OUKA
The University of Osaka
Institutional Knowledge Archive

Title	A Study on Effective Generation and Solution of Matrices in the Process of Numerical Simulation
Author(s)	佐川, 暢俊
Citation	大阪大学, 2015, 博士論文
Version Type	VoR
URL	https://doi.org/10.18910/52023
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

A Study on Effective Generation and Solution of Matrices in the Process of Numerical Simulation

January 2015

Nobutoshi SAGAWA

A Study on Effective Generation and Solution of Matrices in the Process of Numerical Simulation

Submitted to
Graduate School of Information Science and Technology
Osaka University

January 2015

Nobutoshi SAGAWA

List of Publications

A. Journal Papers

1. Nobutoshi Sagawa, Chisato Konno, and Yukio Umetani : “Numerical Simulation Language DEQSOL”, *IPSJ Journal*, Vol.30, No.1, pp.36-45 (1989). (in Japanese)

2. Li Hui, Kensuke Amemiya, and Nobutoshi Sagawa : “Simulation of the Eddy Current in the Thermal Flying Height Control Slider”, *IEEE Transactions on Magnetics*, Vol. 47, No. 7, pp.1809-1812 (2011).

3. Li Hui and Nobutoshi Sagawa : “A Novel Active-Head Slider With a Shear-Mode PZT Actuator and Dual Thermal Actuator”, *IEEE Transactions on Magnetics*, Vol. 49, No. 7, pp.3771-3774 (2013).

4. Nobutoshi Sagawa, Norihisa Komoda, and Ken Naono : “Proposal of a Genetic Algorithm-applied GMRES(*m*) method with Automatic Subspace Parameter Optimization”, *IEEJ Transactions on Electronics, Information and Systems*, Vol. 135, No.6 (2015). (to appear)

B. International Conference Papers

1. Chisato Konno, Miyuki Saji, Nobutoshi Sagawa, and Yukio Umetani : “Advanced Implicit-Solution Function of DEQSOL and its Evaluation”, in *Proceedings of 1986 ACM Fall joint computer conference*, pp.1026-1033 (1986).

2. Nobutoshi Sagawa, Donal Finn, and Neil Hurley : “DEQSOL/AI-DEQSOL: Towards an Integrated Problem Solving Environment for Numerical Simulation”, in *Proceedings of 8th Information Linkage between Industry and Applied Mathematics*, pp.63-80 (1991).

3. Nobutoshi Sagawa, Donal Finn, and Neil Hurley : “An Integrated Problem Solving Environment for Numerical Simulation of Engineering Problems”, in *Proceedings of IFIP Working Conference on Programming Environments for High-Level Scientific Problem Solving*, pp.125-136 (1991).

4. Hiroyuki Hirayama, Miiko Ikeda, and Nobutoshi Sagawa : “Solution functions of PDEQSOL for fluid problems”, in *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pp.218-227 (1991).

5. Ken Naono, Nordin Zakaria, Takao Skurai, Anindya Jyoti Pal, and Nobutoshi Sagawa : “Genetic Algorithm-based Sparse Matrix Iterative Solvers for Efficient Simulation Platforms”, in *Proceedings of the 1st Asian Conference on Information Systems*, pp.171-176 (2012).

6. Nobutoshi Sagawa, Norihisa Komoda, and Ken Naono : “Improvement in Performance of GMRES(*m*) Method by Applying a Genetic Algorithm to the Restart Process”, in *Proceedings of the 2014 International Conference on Computational Science and Computational Intelligence*, pp.466-471 (2014).

7. Nobutoshi Sagawa, Norihisa Komoda, and Ken Naono : “Performance Improvement of a Genetic Algorithm-applied GMRES(*m*) method with Optimization of Subspace Parameter *m*”, in *Proceedings of the 3rd Asian Conference on Information Systems*, pp.34-40 (2014).

C. Domestic Conference Papers

1. Hiroyuki Hirayama, Tadashi Ohta, Miiko Takeishi, Nobutoshi Sagawa, and Kenji Araki : “Automatic Descretization method in the Fluid Dynamics Feature of the

Finite Element Version of DEQSOL", in *Proceedings of the 37th IPSJ National Conference 2nd half 1988*, Vol.1, pp.40-41 (1988). (in Japanese)

2. Nobutoshi Sagawa, Kenji Araki, Hiroyuki Hirayama, Tadashi Ohta, and Miiko Takeishi : "Functional Design of the Fluid Dynamics Feature of the Finite Element Version of DEQSOL", in *Proceedings of the 37th IPSJ National Conference 2nd half 1988*, Vol.1, pp.42-43 (1988). (in Japanese)

3. Nobutoshi Sagawa and Yukio Umetani : "Functional Design of the simulation language DEQSOL for Parallel Computer", in *Proceedings of the 38th IPSJ National Conference 1st half 1989*, Vol.3, pp.1492-1493 (1989). (in Japanese)

4. Nobutoshi Sagawa and Makoto Ohya : "Harmonious Computing – An IT Concept to Manage Ever-increasing Complicity of Data Center Systems", in *Proceedings of the 2004 JSPE National Academic Conference in Spring 2004*, pp.587-588 (2004). (in Japanese)

Summary

This dissertation presents a study on the effective composition and solution of matrices in the numerical simulation process researched from 1985 to 2015 by the author, who is enrolled at the Graduate School of Information Science and Technology, Osaka University, Japan, and has been with Hitachi, Ltd. and Hitachi Systems, Ltd., Japan.

With the increase in computer performance and their widespread application in the manufacturing industry, numerical simulation of physical problems has gained more importance. Understanding the physical phenomena related to industrial products by simulation is a crucial part in the design process. Matrix computation is at the core of numerical simulations of physical problems. The phenomenon that must be handled by the simulation is normally governed by a partial differential equation (PDE), which is converted to a matrix-solution problem via a discretization process.

To perform a highly efficient numerical simulation, we need to address two phases of the simulation process: matrix formation and solution phases. Considering the first phase, notably, the nature of the physical problem is very diverse such that creating a computer program to handle matrix formation from scratch is a formidable task. The problem is avoided using an already available simulation package. However, if such package is unavailable or insufficient, providing the end-user with a high-level language is desirable so that the end-user can represent the problem in a flexible yet concise manner. In the second phase, as opposed to the first phase, the problem is better defined; i.e., we only solve x of $A \cdot x = b$, where A is the coefficient matrix and b is a known right-hand side vector. Although numerous studies have been conducted in this area, we find few approaches to date that pay attention to readily utilize the power of the abundant resources in the present computational environment using the multi-core CPUs or cloud computing.

In reference to the above-mentioned background, we address the following two issues in this dissertation. The first is the design of a high-level language and a translation system dedicated to the matrix-formation phase of the numerical simulation. The language enables compact description of the end-users' problem, whereas the translator converts it into a FORTRAN program suitable for a super computer. The second is the introduction of the genetic-algorithm (GA) concept to the matrix-solution phase. The compatibility of GA with a multi-thread or multi-process computational environment allows the user to exploit the available abundant computational resources.

This dissertation consists of five chapters.

Chapter 1 describes the general process of numerical simulation and overview of the existing method and techniques to handle the process.

Chapter 2 proposes a new language and translation system, which we named as the "Differential EQuation SOLver (DEQSOL)" system, dedicated to forming matrices corresponding to the numerical simulation described by PDEs. We first propose the design principle and specification of the language and then explain the translator architecture that converts the high-level description into a FORTRAN code to form a coefficient matrix. Finally, we evaluate the proposed matrix-formation system. The system enables description of a physical program by one-tenth program lines, and the vectorization ratio of the generated code exceeds 90% and typically reaches 95%.

Chapter 3 and Chapter 4 address the second issue, which is the effective solution of the matrix. An iterative matrix-solution method is chosen as the basis of our study, and GA is introduced where the iteration process starts from multiple initial vectors (chromosomes), and upon stagnation during convergence, they are crossed over. We evaluate the GA performance using several example test matrices and demonstrate that it can shorten the execution time to 1/2–1/6 compared with that of a conventional non-GA case. In the proposed method, the end-user must still specify the calculation parameters that cannot be known a priori. We then try to address this problem of adjusting the parameters with minimum input from the end-user. We propose a technique that captures the stagnation of convergence in a chromosome-wide manner and a method to dynamically adjust the parameters upon such stagnation. We evaluate the proposed dynamic approach against the existing methods that rely on a local stagnation to demonstrate that it yields further 30% performance improvement.

Finally, Chapter 5 concludes our study by summarizing the findings and outlining directions for future research.

Contents

Chapter 1 Introduction 1
1.1 Background 1
1.2 Related Work 5
1.2.1 Effective Environment to Support Matrix Formation for Numerical Simulation 5
1.2.2 Effective Methods for Large-Scale, Sparse, and Ill-conditioned Matrix Solution 7
1.3 Research Strategies 9
1.4 Outline of the Dissertation 12
Chapter 2 Development of a Dedicated Language and a Translation System for Effective Matrix Generation 15
2.1 Introduction 15
2.2 Matrix Formation in Numerical Simulation of Physical Problems 17
2.3 Configuration of the DEQSOL system 19
2.4 Specifications of the DEQSOL Language 19
2.4.1 Language Design 19
2.4.2 Demonstration of DEQSOL Description 22
2.5 Code Generation Method of the DEQSOL Translator 29
2.5.1 Outline of the Translator and the Structure of the Generated Code ... 29
2.5.2 Code Generation for FDM 30
2.5.3 Code Generation for FEM 34
2.6 Advanced Solution Functions 38
2.6.1 Implicit solution function for simultaneous PDE 38
2.6.2 FEM Capabilities to Generate Matrices for Fluid Dynamics

Problems 40
2.7 Evaluation 43
2.7.1 Evaluation for Convection−diffusion Problems 43
2.7.2 Evaluation for Fluid Dynamics Problems 45
2.7.3 Comparison with Other PSEs 47
2.8 Conclusion 49
Chapter 3 Improvement in Performance of GMRES(*m*) Method by Applying Genetic-Algorithm Inspired Enhancement to Restart Process 51
3.1 Introduction 51
3.2 GMRES Method and Look-back Modification 52
3.2.1 GMRES(*m*) Method 52
3.2.2 Look-back GMRES(*m*) Method 54
3.3 GA Application to GMRES(*m*) Method 55
3.3.1 Conventional GA application to Matrix Computing 55
3.3.2 The Algorithm of GA Application 56
3.3.3 Influence of the Initial Vector to GMRES(*m*) Convergence 58
3.3.4 Crossover Algorithm and Preliminary Experimentation 62
3.4 Evaluation 66
3.4.1 Experiment Settings 66
3.4.2 Results 67
3.4.3 Combination of GA–GMRES(*m*) with Look-back Method 72
3.5 Conclusion 74
Chapter 4 Automatic Subspace Parameter Optimization for GA–GMRES(*m*) Method 75
4.1 Introduction 75
4.2 Optimization of the Stagnation Threshold Parameter and the Krylov Sub-space Parameter *m* 76
4.2.1 Influence of the Stagnation Threshold Parameter 76
4.2.2 Chromosome-wide Stagnation 78
4.2.3 Stagnation Detection 79
4.3 Evaluation 80
4.3.1 Numerical Experiments with Example Matrices 80
4.3.2 Comparison with Conventional Stagnation Detection Technique 84

4.4 Conclusion 85
Chapter 5 Conclusions 87
5.1 Concluding Remarks 87
5.2 Future Directions 89
Acknowledgements 91
References 93

Chapter 1

Introduction

1.1 Background

The importance of numerical simulation of physical phenomena is continuously increasing because the simulation can help minimize design lead-time and reduce the cost incurred in the engineering and industrial design processes by replacing physical experiments, thus providing a competitive edge to the product designer and manufacturer [1][2]. Also, notably, sometimes, numerical simulation is the only way to reproduce and analyze a physical problem in question because the phenomenon involved is too small or too large in terms of scale or too complicated to be dealt with by physical experiments. A typical example in the small-scale includes the analysis of the behavior of molecules and particles governed by wave equations [3][4], whereas the huge and complex side of the issue can be exemplified in the analysis of weather pattern governed by a combination of complex fluid and heat equations [5][6]. The coverage of such simulation spreads across diverse physical phenomena, including heat convection, fluid dynamics, magnetics, structural problems, and combinations of these factors. Recent improvement in the underlying computer hardware/software technology has largely contributed to the calculation of these complex physical phenomena to be carried out in a practical and realistic manner within a given time constraint and cost range. During the early stages of scientific numerical simulation in the 80s, the typical performance parameters of scientific computers used in the Japan Metrological Agency (HITAC M200H) used to have merely a 23.8 MFLOPS processing speed and a main memory of 16 MB, whereas the latest machine used in the same agency as of 2012

(HITAC SR-16000M1) boasts of parameters with an 847 TFOPS processing speed and a main memory of 108 TB*. Notably, approximately an increase of 10^6 to 10^7 in the processing speed and memory space has been achieved in the past 30 years. As the competition among industry players becomes stiffer and the computer performance continues to improve, the nature of the simulation becomes complete in scale, more diverse, and multi-disciplinary while the demand for its efficiency remains high.

Matrix computation is at the core of numerical simulations of physical problems. The process to be handled by the simulation is normally governed by a partial differential equation (PDE), which is defined on a one-, two-, or three-dimensional domain where the phenomenon occurs. A number of PDEs are available that are often used to describe typical physical phenomena. One of these examples is the convection–diffusion equation that defines the transfer of particles or energy through a physical system due to the difference in density (from a dense to a lean area—diffusion) and the movement of a substance surrounding an entity (convection). This type of phenomenon frequently appears in a temperature-analysis problem when designing the efficiency of a heat exchanger or in impurity diffusion process when designing a substance-immersion step in semiconductor production. An important variation of the convection–diffusion equation is the Navier–Stokes equation that models the flow of gas or fluid that occurs in weather pattern, ocean current, airflow around a wing, or liquid flow in a pipe, where the transfer effects of the thermal energy, momentum, and mass are simultaneously treated and solved [5][6]. Another well-known PDE is the Maxwell's equation that governs electronic and magnetic fields. This equation plays an important role in the design of such products as electric motor, power generator, hard-disk drive, electric microscope, or even nuclear-related devices [7][8]. In addition to the PDE, boundary conditions need to be imposed on the boundaries of the domain, and an initial condition needs to be set across the domain as the starting value of the simulation to make the solution unique to a specific problem.

The combination of these equations and conditions defines the mathematical model of the problem; thus, analytically or numerically solving these equations provides a solution to the problem in question. Because the analytical method of solving a PDE is available only for the simplest cases, we normally need to rely on numerical simulations to obtain an approximate evaluation of the equation with the help of a computer. Here, the

* http://www.hitachi.co.jp/New/cnews/month/2012/05/0524c.html

continuous value, equation, and domain related to the phenomena cannot be directly handled by a computer, thus necessitating a discretization process to come into play in which the domain is represented by a set of large number of points scattered within and on the boundary of the domain. The continuous value or the parameter in the domain can then be represented as a series of values (i.e., a vector) defined on these points. Together with the values and domain, the PDE and the boundary conditions must be discretized, i.e., these equations are converted into a linear equation at each point to represent the relation between the unknown value on that point and those on the surrounding points. The manner in which the points are scattered, the number of surrounding points referenced, and the method to convert corresponding PDE or boundary condition into a linear equation depends on the discretization technique. Two of the most well-known and widely used techniques are the finite difference method (FDM) [9] and the finite element method (FEM) [10]-[13]. In the FDM, the domain is covered by a mesh consisting of square or rectangular (in a three-dimensional case, cubic or cuboid) grids, where the cross points of the grid are used as the representative points. A simplified mesh topology makes the discretization of the domain and equation rather straightforward while leaving the adaptability of the mesh to the complex boundary shape of the domain and the ability to finely control the mesh density relatively poor. FEM is a generalized version of FDM, which allows the mesh to comprise arbitrary shapes of elements, such as rectangles or quadrilaterals (in a three-dimensional case, tetrahedrons or polyhedrons). This condition in turn provides better fit and control of the mesh to the boundary shape or the resolution of the domain while causing the discretization process to become more complicated. In either method, the number of unknown variables and the linear equations are made identical to that of the representing points, resulting in the collection of linear equations to form a simultaneous linear equation system. It allows the coefficients of the linear equations to be treated as a square matrix, which is commonly referred to as a coefficient matrix. Thus, the solution of a physical problem can eventually be converted to a matrix-solution problem.

To convert a physical problem into a matrix problem, we need to go through two distinctive phases: generating a coefficient matrix and then solving it. A typical process flow of a numerical simulation is shown in Fig. 1.1. The end-user of the numerical simulation first creates a simulation program to generate the corresponding coefficient matrices and to define the control structure as well as the input data to define the domain shape (mesh) and the other parameters such as the material constants and boundary

conditions. The program is then compiled to an executable binary code by a compiler and linked with a matrix-solution library of the end-user choice. The binary code is executed together with the input data, and the simulation result is obtained. The output simulation result is originally presented in a numerical format; however, a graphical visualization system is often used to understand the result in an intuitive manner. In this figure, the matrix-formation phase is directed by the simulation program, whereas the matrix-solution phase is handled by the matrix library.

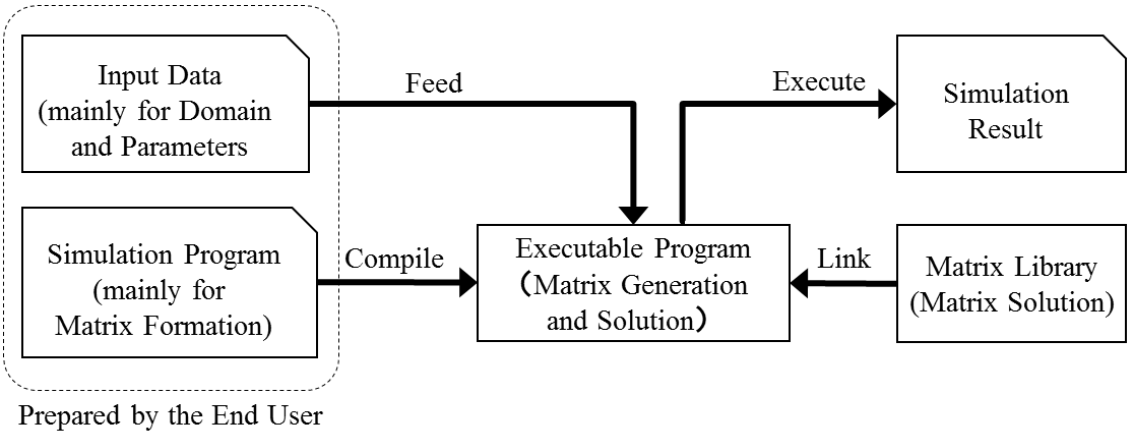


Fig. 1.1　Flow of Numerical Simulation

To meet the demand of carrying out a complex numerical simulation without sacrificing efficiency, we have addressed the aforementioned two phases of matrix operation (formation and solution) in a consistent manner.

Until now, the end-user has been able to rely on an existing simulation package, which generally combines the matrix formation and solution phases, to solve the problem when the problem meets the capability of the package. In this case, the situation is much simplified because the simulation can be performed without the burden of creating a computational program.

However, when no such package suited to solve the end-user problem is available or if the end-user wishes to try a new algorithm or solution scheme that is not implemented in the package, fundamental difficulty in the numerical simulation arises, i.e., no specifically clear method is available to perform the matrix-formation process. The end-user must discretize the PDE by applying a discretization method such as FEM or FDM and manually code a program according to the resultant discrete simultaneous linear equations. The end-user must also develop an algorithm to cope with the inherent

complexity of the equation. For example, the PDE in question may comprise many sub-equations where some are time dependent and non-linear while others are stationary. In a complicated situation, the end-user is responsible for addressing the plurality, time dependence, and non-linearity to choose an appropriate combination of algorithms prior to creating a simulation program. To name a few, the user needs to choose the explicit, implicit, or the Crank–Nicholson method [14] to resolve the time dependence or an alternative substitution or the Newton–Raphson method [15] to handle the non-linearity. This type of difficulty implies that we will be significantly affecting the enhancement of the overall efficiency of the matrix-formation phase by developing the means to describe the PDE, along with the algorithm to handle it.

On the other hand, for the matrix-solution phase, the problem to be addressed is more clearly defined; i.e., solving x of $A \cdot x = b$, where A is the coefficient matrix formed in the previous phase and b is the known right-hand side (RHS) vector. Therefore, solution of the matrices have been one of the major research topics since the advent of the modern digital computers in the 60s, and numerous methods have been proposed to cope with many different aspects of the matrices [16]-[19]. A number of matrix-solution libraries exist that can be linked to user-coded matrix-formation programs. The issues to be addressed in the matrix-solution phase in turn would be on how to solve large-scale, sparse, and often ill-conditioned matrices that appear in contemporary numerical simulations in a more effective manner than those in existing conventional libraries.

In particular, the existing matrix-solution methods are not necessarily fully compatible with the abundant computer resources made available by the technologies, such as multi/many-core CPUs or cloud computing [20]. In the current study, we intend to investigate how such abundant resources can be utilized to obtain even better performance in existing algorithms and to propose a new matrix-solution architecture that is highly compatible with today's multi-threading or multi-processing computational environment.

1.2 Related Work

1.2.1 Effective Environment to Support Matrix Formation for Numerical Simulation

When the physical problem that an end-user faces falls in a common category, an existing simulation package can possibly be applied to solve this problem. Although the

corresponding coefficient matrices are formed internally and solved, the operations at the matrix level are kept invisible from the user. The user only needs to feed the input data to the package that describes the topology, setting the material constants, and specifying the parameters for the initial and boundary conditions. Some of the well-known examples of these simulation packages include commercial products such as the LS-DYNA [21] (for structural deformation problem found in car crash analyses) and Moldflow [22] (for slow viscous fluid problem utilized in injection molding analyses). There are other simulation packages that are publicly available, e.g., open-source simulation packages such as the OpenFOAM [23] for fluid-dynamic problems that can be used to optimize airfoil design. However, these packages suffer from distinct limitations in the applicable physical phenomena and the adopted numerical algorithms because the scope of such packages is predetermined and inflexible. Additionally, users do not have complete control over the numerical algorithms in detail, and understanding or utilizing the wealth of functions and interfaces of the package is not always easy.

Under these circumstances, the end-user must hand-code a simulation program. The development of a simulation program to form a coefficient matrix in a conventional language such as FORTRAN and C is plagued with the following problems. A prolonged leading time may result in the development of even a simple matrix-formation program, and the end-user must be familiar with special knowledge of the numerical analysis methods of discretization such as FDM and FEM. In addition, a specialized programming technique is required to exploit the performance of scientific simulation-oriented vector/parallel processors. Even if such a program can be created, it tends to be very lengthy and complicated such that it is not suitable for maintenance and modifications at a later stage. To address these limitations, several simulation systems have been developed on the basis of a higher level description of a physical problem than the computer program such as FORTRAN or C. Altogether, this level of systems based on specialized language, script, or interface to describe and solve the problem is called the problem-solving environment (PSE) [24][25]. Another approach to solve a mathematical problem using a digital computer is through symbolic computation of mathematical equations. Macsyma [26], and more recently, Mathematica [27], are representative systems in this domain. The symbolic systems are suitable to handle equations at an expression level, rather than to solve them at a numerical level, which limits its applicability to solve PDEs.

1.2.2 Effective Methods for Large-Scale, Sparse, and Ill-conditioned Matrix Solution

A number of matrix-solution methods have been developed along with the evolution of digital computers. The matrix-solution methods can roughly be classified in the following three categories.

(1) Direct Method

This category of methods has the longest history and is still known to be the most robust algorithm of all the categories. These methods try to solve a system of linear equations by eliminating the unknown variables one at a time. They work well for all types of matrices regardless of their sparseness, symmetry, and condition number. The significant drawbacks of these methods are that they are not memory effective and are computationally expensive, thus limiting their application to relatively small- or mid-sized problem scales.

(2) Simple Iterative Method

This category of solvers tries to solve the matrix by improving the initial guess through recurrence progression. It requires the smallest memory and simplest calculation at each stage, whereas the robustness and convergence speed tend to be weak, especially when the matrix is not well conditioned.

(3) Gradient method

This category is a type of iterative method where, instead of directly handling the equation $A \cdot x = b$, attempts are made to minimize its integral form $f(x) = (x, A \cdot x)/2 - (b, x)$.

To search the minimizing points in an iterative manner, search vector concept is introduced. After an initial guess wherein x is set to an arbitrary vector, the search vector for the initial iteration is calculated. The initial guess is then advanced in the direction of the search vector to fulfill a pre-set criterion. From the second iteration, the next search vector is sought to be, in some sense, orthogonal to the preceding vectors. A number of variants exist for this method by changing the minimizing criteria and definition of orthogonality. The characteristics of the gradient method can be positioned in between the previous two categories. It resembles the simple iterative methods because it tries to reach the solution by improving the initial guess. It shares some commonality with the direct methods that it is proven to reach the solution after a predetermined number of iterations.

The summary of the characteristics of the above three categories of matrix-solution

methods is listed in Table 1.1. Among these methods, the generalized minimal residual (GMRES) method, which is a variation of the gradient method, is developed as a promising method to solve large-scale sparse matrices with better convergence and lower memory consumption [28][29]. In this method, all search vectors in the previous iteration stages are recorded in the memory, and the search vector for the next stage is calculated so that it becomes orthogonal to all the previous ones. Because of these characteristics, the GMRES method demands extra memory space and orthogonal vector calculation whose cost increases as the iterations become larger. Since the advent of the GMRES method, a number of modifications have been proposed to minimize this drawback and improve the memory and computational efficiency. Although many methods among them turned out to be effective, no absolute modification has been developed so far, leaving rooms for further improvement.

Table 1.1 Category of Matrix Solution Methods

Type	Outline	Memory use	Calc. amount	Robust-ness	Typical Methods
Direct	Solve a matrix by eliminating unknowns one at a time	Large	Large	Good	Gaussian -elimination
Simple Iterative	repeat $x_k = x_{k-1} + c*(b - Ax_{k-1})$ until convergence	Small	Small	Fair	Jacobi SOR
Gradient	Minimize the integral form $f(x) = (x, Ax)/2 - (x, b)$	Medium	Medium	Medium ~ Good	ICCG, BiCG GMRES

SOR: Successive Over-Relaxation, ICCG: Incomplete Cholesky Conjugate Gradient, BiCG: Biconjugate Gradient, GMRES: Generalized Minimal RESidual method

Alternatively, the recent advent of many/multi-core CPUs and cloud computing has paved way for using abundant computational resources to accelerate simulations. One promising approach highly conformable with such resources is the use of GA because of its compatibility with parallelism, i.e., chromosomes belonging to the same generation in a GA calculation can be readily distributed and processed in parallel [30][31]. Several proposals have been put forth to apply GA to numerical simulation and matrix computation [35][36]. However, most of these efforts have only concentrated on the optimum choice of matrix computation parameters, such as the type of solver and preconditioner, the Krylov subspace dimension, and the number of execution threads. A fundamental study to apply the GA to a body of matrix computing remains as an unexplored domain.

1.3 Research Strategies

The overall objective of this study is to provide a numerical simulation's end user with a means to execute the simulation effectively. Here, we define effectiveness as the total time required starting from development of the simulation program until an understanding of the physics in question. The typical numerical simulation process can be broken down into two steps, i.e., simulation program development and simulation execution, as schematized in Fig. 1.2. We denote the time consumed in each step as t_{pd} and t_{ex}, respectively. Then, the total time t_{all} to perform the simulation can be approximated as follows:

$$t_{all} = (t_{pd} + t_{ex} \cdot l) \cdot k \tag{1.1}$$

where k is the number of iterations to refine and tune the simulation program, and l is the number of iterations to execute the simulation with a variety of parameter settings. The following relationships are normally observed among these parameters.

$$l \gg k, \qquad t_{pd} \gg t_{ex} \tag{1.2}$$

This suggests that t_{pd} and $(t_{ex} \cdot l)$ in (1.1) are likely to possess the same order of impact in t_{all}. Therefore, to achieve the objective of minimizing t_{all}, we must aim at reducing t_{pd} and t_{ex} in a consistent manner.

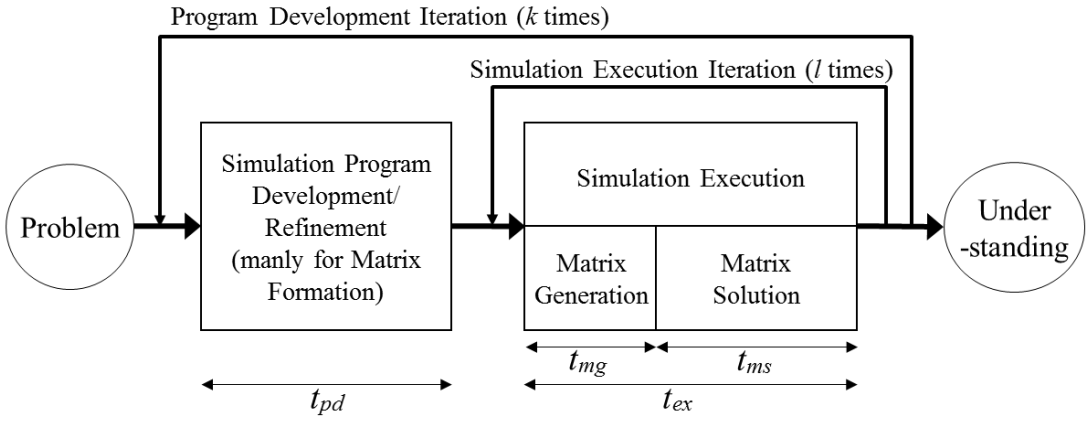


Fig. 1.2 Breakdown of the Numerical Simulation Process

The simulation execution time t_{ex} can be further broken down to the sub-steps of matrix generation time (t_{mg}) and matrix solution time (t_{ms}), where $t_{ms} > t_{mg}$ typically holds. This suggests that paying more attention to t_{ms} than t_{mg} will be rewarding relative to achieving the objective; however, reduction of t_{mg} remains an issue.

With the above procedural analysis in mind, we set forth the following two goals to achieve the above objective:

(1) For the matrix formation phase;

the most time consuming task is the creation of a computational program that discretizes the PDE. We aim to realize an effective simulation environment that allows the end user to describe the physical problem in one-tenth of the FORTRAN code lines (to reduce t_{pd}), while enabling enhancement of the performance for matrix generation (to reduce t_{mg}).

(2) For the matrix solution phase;

we aim to realize an even more efficient matrix solution method than existing methods by providing a means to exploit the abundant computational resources that have been made available by multi/many-core CPUs and cloud computing (to reduce t_{ms}).

Fig. 1.3 shows the scope of the following chapters in the overall structure of the problem domain. The outline of the research strategies to fulfill these goals is as follows.

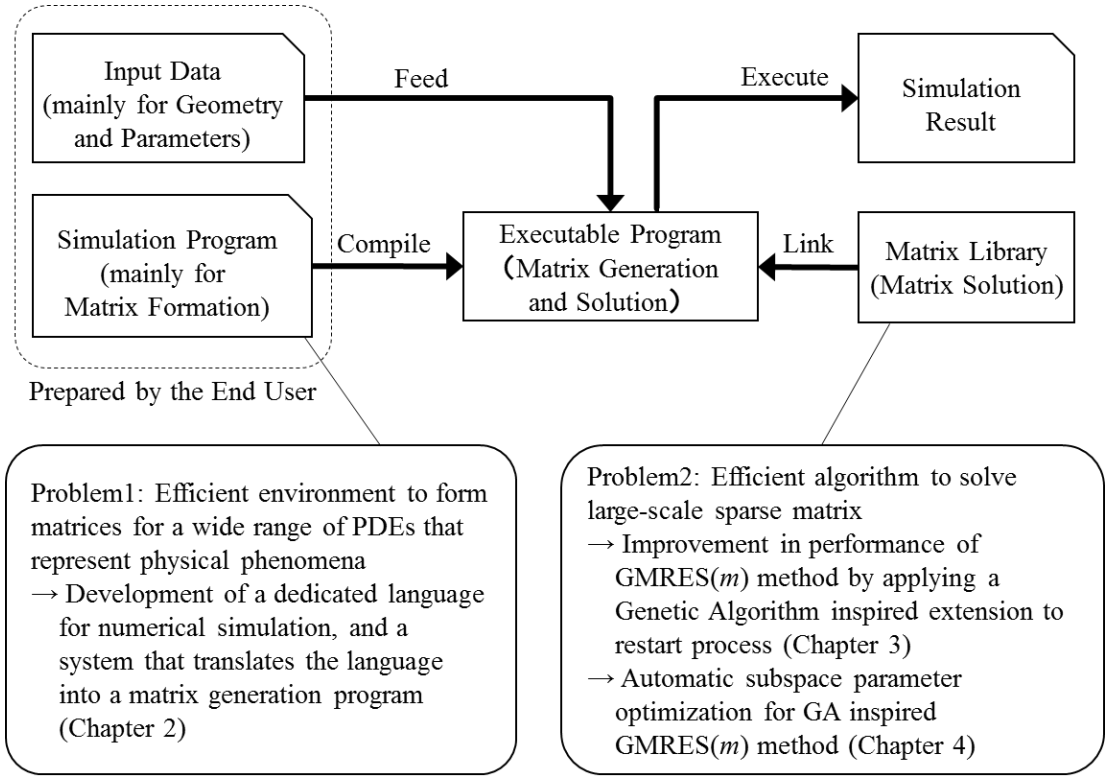


Fig. 1.3 Efficient Environment for Matrix Generation and Solution

(1) Efficient environment to generate coefficient matrices for a wide range of numerical simulation

To cope with the problem of effectively forming coefficient matrices for a wide range of numerical simulation of physical phenomena, we develop a dedicated language suited to describe the physical phenomenon to be solved and a translation system from the description to the program that eventually generates the matrix. The language is designed such that the end-user can describe the physical phenomenon at a level that is as high as the PDE, rather than that at the level of matrix operation or linear equation. The translation system subsequently converts the description to a computer program (such as in FORTRAN), which in turn forms a matrix corresponding to the PDE. The computer program is compiled and linked with one of the existing matrix-solution library. Executing the resultant binary code yields the simulation result. The main issue in designing the simulation language is to minimize the end-user effort to describe the physical phenomena while maintaining the ability to control the simulation method as flexible and the range of the describable PDE as wide as possible. Considering the translation system, a mechanism must be developed as the core technology to automatically discretize the specified PDE using a symbolic formula translation and to create the corresponding matrix-generation program.

(2) Efficient algorithm to solve large-scale sparse matrix

For realizing an efficient algorithm to solve a large-scale sparse matrix more effectively than before, we choose the GMRES(m) method as the basis of our approach [37]. This method is a modification of the GMRES method to minimize its drawbacks associated with the orthogonal search vector creation by limiting the GMRES iteration up to m times (m is user specified) and forcing the GMRES process to restart with a new starting point. We propose a method to further improve the GMRES(m) efficiency by introducing a method inspired from the genetic-algorithm (GA). Instead of using one initial vector as the GMRES(m) starting point, our approach uses multiple starting vectors and crosses over the results when the convergence stagnates. With the help of the GA, the GMRES(m) process can readily leverage the power of the abundant CPU resources to turn it even more efficient in modern computer environment.

In the GA-inspired GMRES(m) method discussed above, the end-user still needs to supply an appropriate parameter m suitable to solve the given matrix. Most of the conventional methods of automatically choosing m rely on a simple approach of

measuring the residual $||b - A \cdot x||$ as the calculation method and increase or decrease the value of m when the reduction of the residual is not sufficiently large or small. This approach often fails to meet the expected effect because the decrease in the residual is neither necessarily monotonous nor stable. We propose a method to automatically adjust m by observing the convergence process of multiple GA threads so that its value can be increased or decreased according to the progress in the global convergence process.

1.4 Outline of the Dissertation

The remainder of this dissertation is organized as follows :

Chapter 2 proposes a new language and translation system, which we named the "DEQSOL" system, dedicated to forming the matrices corresponding to the numerical simulation described by the PDEs [38]-[45]. DEQSOL consists of two components: the language and the translator. This chapter starts by describing the conventional approaches of forming coefficient matrices in the numerical simulation and indicates that matrix formation is a formidable task if no existing numerical simulation package is available suited to an end-user problem. The end-user must manually discretize the PDE to create a computer program that generates the corresponding matrices of the end-user problem. The situation worsens when the end-user wishes to utilize a super computer because it requires the end-user to create a computational program, taking special care of the inherent parallelism of the physical problem and the solution algorithm. We first propose the design principle and specification of the DEQSOL language that allow the end-user to describe the physical problem in the PDE level to alleviate the burden of writing lengthy matrix-formation programs. We then explain the architecture of the DEQSOL translator that converts the PDE-level description into a FORTRAN code that forms a coefficient matrix while exploiting the computational power of a vector super computer. Finally, we evaluate the proposed matrix-formation system in terms of the description coverage, productivity enhancement, and execution performance relative to the vectorization ratio.

Chapter 3 and Chapter 4 address the other sides of the issues: effective solution of the matrix [46]-[50]. The GMRES(m) method is introduced as the basis of our study, and existing methods are reviewed to enhance its convergence efficiency. Then, we pay special attention to the fact that the convergence is influenced by the choice of the initial vector. This preliminary study leads us to an approach where the GMRES(m) process

starts from two or more initial vectors (chromosomes), and upon convergence stagnation, they can be crossed over, taking a hint from the GA to overcome the stagnation. For the crossover algorithm, we examine several possible choices through a preliminary numerical study, which shows that the "weighted average" algorithm has a particular advantage over what the other algorithms can achieve. We name this method (i.e., the combination of the GA application and the adoption of the weighted average for crossover) as the "GA–GMRES(*m*)" method and evaluate its performance using several exemplary test matrices. In the GA–GMRES(*m*) method, the end-user still needs to specify the appropriate value of *m* that cannot be known a priori. Then, we try to address this problem of choosing and adjusting the value of *m* with minimum input from the end-user. The concept of "chromosome-wide stagnation," which is an indicator of convergence stagnation across chromosomes, is introduced, and the method to adjust *m* on the stagnation is proposed. We evaluate the chromosome-wide stagnation approach against a conventional method to adjust *m* on the local stagnation. The numerical experiment using test matrices shows that the new approach exhibits more favorable performance, where *m* can be modified only at the necessary points of stagnation, in contrast to the conventional method where the value of *m* tends to be changed at an unnecessary frequency.

Finally, we conclude the study by summarizing the findings and show the directions for future research in Chapter 5.

Chapter 2

Development of a Dedicated Language and a Translation System for Effective Matrix Generation

2.1 Introduction

To perform numerical simulation of physical phenomena effectively, the formation of the coefficient matrix plays a crucial role. This chapter proposes a dedicated high-level language for numerical simulation and a translation system that converts a description in the proposed language to a simulation program to form a coefficient matrix [38]-[45].

Most of the physical phenomena dealt with in today's industry problems are defined in terms of PDEs. To solve a PDE using a digital computer, we must first discretize the PDE, boundary conditions, and the domain in which the phenomenon takes place. We must then solve the resultant simultaneous linear equations by means of a matrix solution. No general-purpose software package for discretizing PDEs is available; thus, user-written simulation code is typically required. However, such a task is time-consuming and often prone to implementation errors. Therefore, addressing this situation by providing the end user with a means to describe the problem at a higher level than computer programing languages, such as FORTRAN or C, is desirable.

There have been several attempts to provide such a simulation language and supporting system. This category of systems is collectively referred to as a Problem Solving Environment (PSE) [25]. Some typical examples of PSE implementations are SALEM [51], PDEL [52], FIDISOL [53], ALPAL [54], DISTRAN [55], and ELLPACK

[56]-[58]. In particular, ELLPACK is an open source system that can deal with a wide range of physical phenomena using a PDE level description. It has ample problem solving capabilities due to its extensive library, and the number of libraries has been expanded by community members to meet the demands of interactive and parallel computer environments. The ELLPACK system can be considered a sophisticated driver system to the embedded solution library for a wide range of PDE templates. This particular feature of ELLPACK also has its own limitation, i.e., to widen the coverage of physical problems, ELLPACK must add an appropriate library and template to handle them. Another drawback of this approach is that the numerical scheme to handle a PDE is prefixed in the template. Although this feature makes the description of the problem simpler, it is difficult for the end user to examine alternative numerical schemes of their choice. A commercial software package in line with the PDE template-based approach is FlexPDE [59][60], which equips extended functionalities to deal with moving boundary or surface contact problems, appearing in the structural analysis or fluid dynamics domains. In more recent years, MATLAB [63] has gained popularity among engineers and scientists who intend to carry out various numerical analyses, including matrix generation and solution. However, although MATLAB provides wide coverage of numerical and matrix oriented operations and can readily combine a variety of analytic components using user-interactive specification, its ability to generate matrices from a diverse range of PDEs and describe numerous solution algorithms is limited. For example, MATLAB can only express a one-dimensional PDE; however, typically two- or three-dimensional PDEs appear in realistic modern industry design problems.

To alleviate the difficulties associated with matrix formation as described above, we propose an approach of designing a dedicated language to describe physical phenomena at a PDE level. In addition, we propose a translation system from the description into a FORTRAN program that composes a corresponding matrix. We refer to the dedicated simulation language as the Differential EQuation SOLver (DEQSOL) language, and the translator as the DEQSOL translator. The language and translator are collectively referred to as the DEQSOL system. The system has two objectives:

(1) To enhance programming efficiency by designing a suitable language that describes the physical phenomenon in a simple and architecture independent manner.

(2) To enhance execution efficiency by providing a translator that generates highly vectorizable FORTRAN codes in which a corresponding matrix is composed from the DEQSOL description utilizing intrinsic parallelism in the problem.

In contrast to ELLPACK and other template-based systems, DEQSOL is a code generation system from a high-level description of the problem using a dedicated language into a FORTRAN code that composes the corresponding matrix. The extension of its applicable field is realized by broadening the description capability of the language rather than adding more templates and libraries to the system. The code generation approach is currently gaining attention as a part of domain specific language, due to its expandability for a wide range of physical problems and adaptability to exploit the characteristics of the target computational architecture (e.g., parallel, vector, and multi-core) [61]. DESOL can be considered one of the earliest and most successful examples of such work [62], equipping a parser and a code generator as elaborated in Section 2.5.1.

The remainder of this chapter is organized as follows. Section 2.2 provides the basic background of matrix formation in the numerical simulation process. Section 2.3 explains the overall architecture of the DEQSOL system. Section 2.4 describes the design principle and specifications of the DEQSOL language, and demonstrates its descriptive power through examples. Section 2.5 discusses the code generation techniques in the DEQSOL translator for both FDM and FEM functionalities. Section 2.6 provides further explanation regarding the advanced features that DEQSOL equips, especially for fluid dynamics problems. In Section 2.7, we evaluate the DEQSOL system in terms of the descriptive efficiency of the language and the execution performance of the generated code. Finally, we conclude this chapter in Section 2.8.

2.2 Matrix Formation in Numerical Simulation of Physical Problems

In this section, we briefly describe how the matrices are formed in a numerical simulation process of physical problem with reference to the schematic diagram of Fig. 2.1. Here the PDE governing the physical phenomena is denoted as $\Psi(x) = b$ (Ψ : differential operator) defined on a continuous domain Ω, and its solution is a continuous two dimensional function x over the same domain. The aim of the simulation is that "given Ψ and b, obtain the function x". Since a digital computer cannot handle a continuous problem directly, the PDE and domain need to be discretized prior to obtaining the solution. Domain discretization means to represent the domain with a set

of points as is depicted in the top right diagram of Fig. 2.1. Then the continuous function x over the domain can be represented as a series of values x_i (i = 1 to n, n: the number of points) defined on these points. On each point, the PDE is converted to a linear equation that defines the relationship of the function value on this point to the values on the points around it. Take point 4 in Fig. 2.1 top right diagram (marked in red) as an example, the linear equation obtained from the discretization takes the form of:

$$a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 + a_{45}x_5 + a_{46}x_6 = b_4 \tag{2.1}$$

if the points of 1, 2, 3, 5, and 6 are referenced as the surrounding points of 4. Note that how the points are distributed on the domain, how many points are referenced, and how the coefficients a_{ij} are calculated vary depending on the discretization method such as FDM or FEM. In FDM square mesh (grid) is adopted whereas in FEM the location of the points is arbitrary. Since the formation of the linear equation (2.1) can be repeated for all the points on the domain Ω, the resulting equations will form an n dimensional

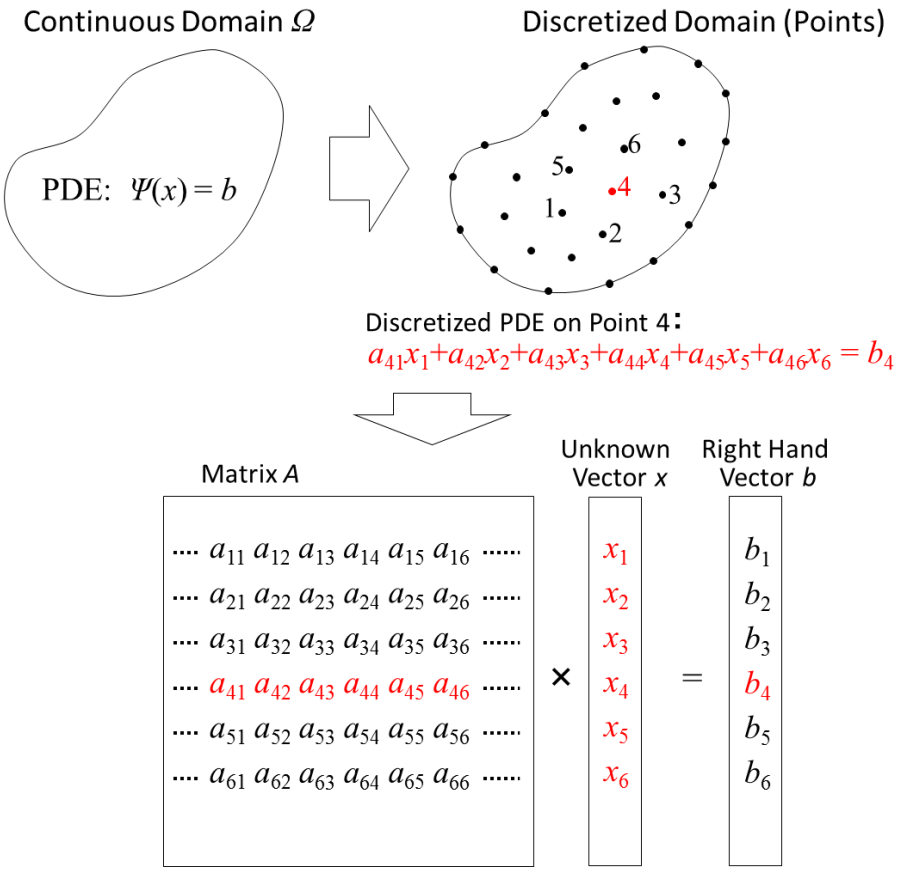


Fig. 2.1 Discretization of the Domain and PDE

simultaneous linear equation (i.e., having n equations and n variables). As is well known, this kind of simultaneous linear equation system can be treated as a matrix-solution problem of $A \cdot x = b$ where A is the coefficient matrix, b is the RHS vector, and x is the unknown vector as is shown in the lower part of Fig. 2.1.

2.3 Configuration of the DEQSOL system

The overall configuration of the DEQSOL system is illustrated in Fig. 2.2. A DEQSOL description defining the physical problem is first prepared by the end-user in accordance with the DEQSOL language specification. DEQSOL translator then reads in the description and converts it into a corresponding FORTRAN code together with input data, e.g., mesh topology and material constants. The user can choose either FEM or FDM as the discretization method depending on the characteristics of the problem. The FORTRAN code is compiled and linked with an external matrix-solution library, to form an executable program. By executing the program with the input data finally yields the simulation result.

DEQSOL includes the ability to form matrices for various kinds of PDEs by discretizing them by either FDM or FEM. In this chapter, we firstly intend to describe the basic feature, design principle, and specifications of the DEQSOL language. Then the automatic PDE discretization method adopted in the translator is disclosed paying special attention to generate highly vectorizable FORTRAN program suited to a vector super computer. The following section of this chapter describes the advanced features of DEQSOL to handle PDEs appearing in structural analysis and fluid dynamics simulations; e.g., matrix formation for simultaneous PDEs, matrix lumping, and up-winding. In the last section, the description ability of DEQSOL language and the performance characteristics of the generated FORTRAN code are experimentally evaluated.

2.4 Specifications of the DEQSOL Language

2.4.1 Language Design

The basic principle of the language design is to provide a high-level interface for the end-user of the simulation enabling simple, intuitive, yet flexible way to express the

physical phenomena; thus realizing supreme productivity, readability, expandability, and ease of maintenance of the simulation code.

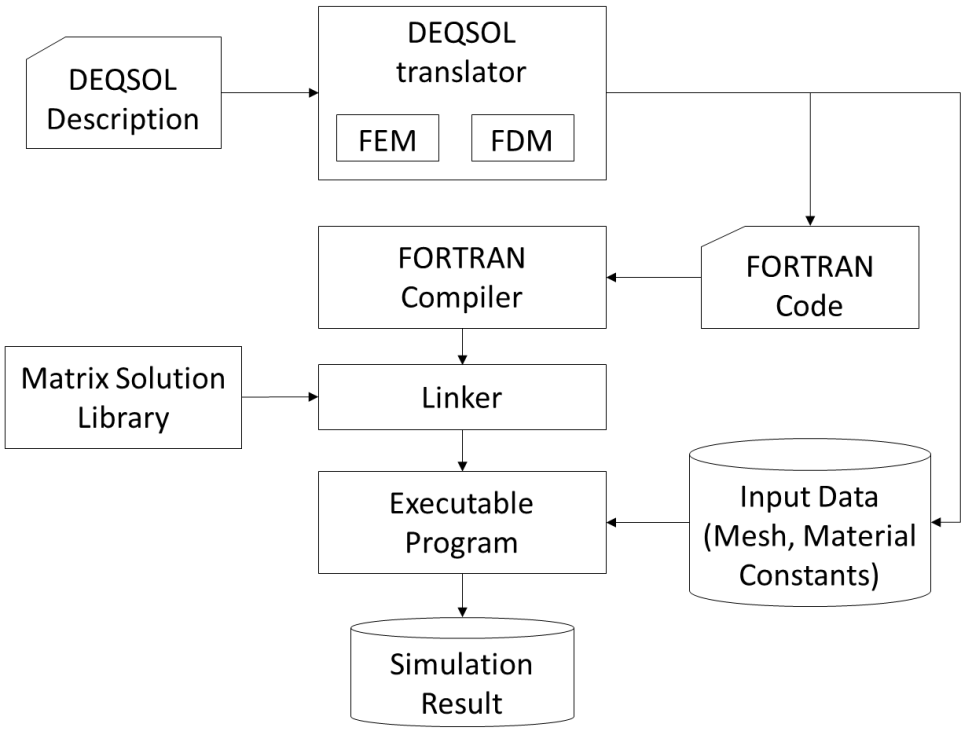


Fig. 2.2 DEQSOL System Configuration

This includes the following features.

(1) Introduction of concise mathematical notations

The end-user should be able to describe the problem using a PDE level description of the physical phenomenon. Mathematical algorithms to deal with the characteristics of PDEs such as time-dependency, non-linearity, and plurality should also be described in a concise manner. To enable these design goals, the following features are incorporated in the language specification.

(a) Multiple types of variables are introduced to express the physical phenomena in a natural way. That includes scalar, vector and tensor variables, and their combinations.

(b) Differential operators, such as the first order derivatives (DX, DY, DZ), second order derivatives (DXX, DYY, DZZ) are introduced. Combined operators such as divergent (DIV), gradient (GRAD), rotation (ROT), and laplacian (LAPL) operators are also incorporated to express a diverse range of PDEs in a natural and concise manner.

The notations prepared in DEQSOL language are shown in Table 2.1. As an example,

the anisotropic convection−diffusion equation (*K*: tensor, *V*: vector, D: scalar)

$$\frac{\partial D}{\partial t} = div(K \cdot grad(D) - V \cdot D) + Q \qquad (2.2)$$

is described as (hereafter notation in DEQSOL is indicated in `Courier` Font)

```
DT(D) = DIV(K..GRAD(D) - V*D) + Q
```

using the DEQSOL language, which possesses intuitive similarity to the original differential equation.

(2) Clear separation of physical model description and numerical algorithm

We make a clear distinction between physical model description (e.g., domain shape, material constants, and boundary conditions) and description of the solution algorithm (denoted as scheme-block) in DEQSOL language. By this way the end-user is able to make frequent modifications to the scheme without touching the description of the physical model, and vice versa.

(3) Capability to represent computational algorithms in the scheme-block

The basic language components to control the execution flow of a program are embedded in the language specification; e.g., sequential execution, conditional branch (IF .. THEN .. ELSE), and iteration (ITER .. END ITER).

(4) Capability to handle domain shape and mesh structure

Not only the physical model but the domain shape and mesh structure can be defined in DEQSOL language so that the comprehensible system can be offered to the user and he does not have to rely on any external system. In addition to this, the DEQSOL system provides an interface where the shape and domain data can be imported from a third party CAD system. This feature is particularly useful in case the domain shape is complex and/or three-dimensional, or fine control of the mesh density is required.

(5) Automatic extraction of intrinsic parallelism

The language does not provide the means to explicitly specify the parallelization as opposed to the directives in a parallel/vector compiler. In turn, the DEQSOL translator extracts the parallelism inherently existing in the algorithm and domain to generate high vectorizable code without end-user's intervention.

Table 2.1 Mathematical notation in DEQSOL

	Mathematical Expression	DEQSOL Notation
Data Types	Scalar : p	`VAR P;`
	Vector : $V = (v_x, v_y, v_z)$	`VEC V;`
	Tensor : $T = ((t_{xx}, t_{xy}, t_{xz}),$ $(t_{yx}, t_{yy}, t_{yz}),$ $(t_{zx}, t_{zy}, t_{zz}))$	`TENS T;`
Multiplication Operators	inner product $U \cdot V$	`V1..V2`
	tensor vector product $T \cdot V$	`T..V`
Differential Operators	$\frac{\partial p}{\partial x}, \frac{\partial^2 p}{\partial x^2}$	`DX(P), DXX(P)`
	$div(V)$	`DIV(V)`
	$grad(p)$	`GRAD(P)`
	$rot(V)$	`ROT(V)`
	$lapl(p)$	`LAPL(P)`
	$div(T \cdot grad(p))$	`DIV(T..GRAD(P))`

The statements used in DEQSOL specification for FDM is listed in Table 2.2. Every DEQSOL statement starts with one of the declarative keywords and ends with a semicolon (;). A collection of the statements comprises a DEQSOL description.

The specification for FEM is fundamentally the same as that in FDM except that there are extra keywords supplied to define complex domain shapes and mesh structure. In FEM, the domain shape is described in a hierarchical manner; the feature points are first specified, then the lines connecting those points are defined, and the area is defined referring to those lines. The mesh size can be controlled by specifying the number of divisions on each line. As was mentioned in (4), it is also possible to import a mesh generated by an external system, which makes DEQSOL more applicable to the problems occurring in a complicated three-dimensional domain.

2.4.2 Demonstration of DEQSOL Description

Combination of the statements in the notation in Table 2.1 and the statements in Table 2.2 provides the end-user with a powerful tool to express various differential equations in an effective way. Examples of DEQSOL descriptions for two problems are

Table 2.2 DEQSOL statements

No	Declarative Keyword	Function
1	PROG	Declare the program name
2	METOD	Specify the discretization method (FDM/FEM)
3	DOMAIN	Specify the physical domain, also indicate whether the problem is 1-, 2- or 3-dimensional
4	TIME	Specify the time domain (start/end time)
5	MESH	Specify the spatial mesh density
6	TSTEP	Specify the time step
7	REGION	Define sub-region that is referred to for setting up boundary conditions, material constants etc.
8	CONST	Define the material constants
9	VAR	Declare the scalar variables to be used
10	VEC	Declare the vector variables to be used
11	TENS	Declare the tensor variables to be used
12	BCOND	Specify the boundary conditions
13	ICOND	Specify the initial condition
14	COUNT	Declare counter variables used in the iteration loop
15	SCHEME, END SCHEME	Declare the beginning/end of the algorithm scheme-block
16	(var) = PDE;	Assignment statement. Discretize and evaluate the PDE on right side and assign the result to the left side (var)
17	SOLVE (var) of PDE;	Discretize and solve the PDE with regard to (var)
18	ITER, END ITER	Definition of iteration loops
19	IF THEN ELSE ENDIF	Definition of if-then-else branch
20	CALL	Call external function
21	PRINT	Print out the numerical values of the variable
22	WRITE	Write the numerical values of the variable to a file
23	END	Declare the end of the program

presented to demonstrate the use of the DEQSOL language for a simple heat conductivity problem and non-linear impurity diffusion problem.

(1) Description example 1: Heat Conductivity Problem

This trivial example is chosen for the sake of simplicity to present the minimum feature of DEQSOL language. As illustrated in Fig. 2.3, the temperature $T_p(x,y,t)$ satisfies the diffusion equation,

$$\frac{\partial T_p}{\partial t} = a \cdot lapl\left(T_p\right) \tag{2.3}$$

on the rectangle domain of ($0.0<x<1.0$, $0.0<y<1.2$) and time domain ($0.0<t<1.0$) with initial condition,

$$T_p(x,y,0) = 100$$

and boundary conditions,

$$T_p = 0.0 \text{ on BOTTOM}, \quad T_p = 100.0 \text{ on TOP},$$
$$\partial T_p/\partial n = 0 \text{ on LEFT and RIGHT},$$

where *lapl*(..) denotes Laplacian, (∂/∂n) denotes normal gradient (gradient perpendicular to the boundary), and the thermal conductivity a is set to 0.5.

The DEQSOL descriptions that describe this problem are shown in Fig. 2.4 when FDM is chosen, and in Fig. 2.5 when FEM is adopted.

In the FDM case of Fig. 2.4, the statements are configured as follows. The statement (1) declares the problem name, statement (2) selects the method of discretization (in this case FDM), statements (3) to (7) specify the spatial and time domain and mesh, statements (8) and (9) declare the variables and constants, statements (10) and (11) define the boundary and initial conditions. The section comprising the lines (12) to (19), surrounded by SCHEME and END SCHEME statements, is referred to as "SCHEME block" that defines the numerical scheme to solve the problem. Since this example is time dependent, the time differential ($\partial T_p/\partial t$) is treated as

```
(TP-TO)/DLT
```

where `TO` is the temperature of the previous time step, `TP` is that of the current time step, and `DLT` stands for delta-T; the time step size.
and the original PDE of $\partial T_p/\partial t = a \cdot lapl(T_p)$ is converted to the following expression.

```
(TP-TO)/DLT = A*LAPL(TP)
```

The SOLVE statement of (14) to (16) indicates that the above expression is to be

discretized and the matrix is to be formed regarding TP as the unknown variable, then the matrix is to be solved using a matrix-solution library of “ICCG”. This kind of treatment of time dependency is known as an implicit method since the value TP to be solved appears on the both sides of the equation. Note that the matrix-solution library, the main issue in Chapter 3, is outside the scope of DEQSOL system so that various kinds of external libraries can be linked and utilized at a later stage.

The same PDE can be handled by an Euler explicit method by replacing the SOLVE statement with the following assignment statement.

```
TP = TO + DLT*A*LAPL(TO);
```

In this case, simple assignment statement suffices because the equation has been transformed so that the unknown variable TP exists only on the left side, thus avoiding a matrix solution to come into play. It is known that the Euler's explicit method necessitates less computation per iteration because of absence of the matrix formation and solution, but we need to rely on a smaller time step for the iteration because otherwise the result is going to diverge. On the contrary, although the implicit method mandates rather expensive matrix operation per each iteration, it is proven to be convergent regardless of the time step. As can be seen even from this simple example, the end-user can describe the preferred numerical scheme in a concise manner in DEQSOL language, and if it is proven unsatisfactory, the scheme can be readily modified by replacing a few lines of the description at a later stage. This feature is particularly useful when the numerical scheme to be used is not well established, and the user needs to take a try-and-error approach before he reaches the conclusive method.

The situation holds almost the same in the FEM description in Fig. 2.5, apart from the topology related statements such as (6) POINT statement to specify the feature points of a region, (8) REGION statement to define a 2- or 3- dimensional geometrical domain, and (11) MESH statement to define a mesh topology comprising polygons or polyhedra on which FEM calculation is performed. In case of FEM, a data file that contains a mesh consisting of triangle or quadrilateral elements is created by the DEQSOL translator, together with a FORTRAN program. The mesh density is controlled by specifying the number of divisions on each lateral line in the MESH statement. The SHEME block is identical to that of FDM description.

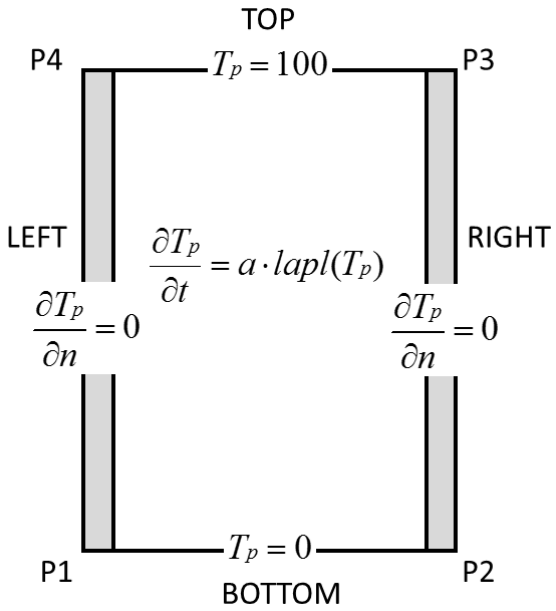


Fig. 2.3 Heat Conductivity Problem

```
(1)  PROG     HEAT;
(2)  METHOD   FDM;
(3)  TIME     T=[0:1];
(4)  DOMAIN   X=[0:1], Y=[0:1.2];
(5)  MESH     X=[0:1;5], Y=[0:1.2:6];
(6)  TSTEP    DLT=[0(0.02)1.0];
(7)  REGION   RIGHT=[1,*], LEFT=[0,*],
              TOP=[*:1.2], BOTTOM=[*,0];
(8)  VAR      TP, TO;
(9)  CONST    A=0.5;
(10) BCOND    TP=0.0 AT BOTTOM,
              TP=100.0 AT TOP,
              DX(T)=0.0 AT RIGHT+LEFT;
(11) ICOND    TO=0.0;
(12) SCHEME;
(13)      ITER NT UNTIL NT GT 50;
(14)           SOLVE  TP  OF
(15)               (TP-TO)/DLT = A*LAPL(TP)
(16)                  BY 'ICCG';
(17)           TO=TP;
(18)      END ITER;
(19) END SCHEME;
(20) END;
```

Fig. 2.4 DEQSOL Description for Heat Diffusion Problem by FDM

```
(1)  PROG     HEAT;
(2)  METHOD   FEM
(3)  DOMAIN   X=[0:1.0],Y=[0:1.2];
(4)  TIME     T=[0:100];
(5)  TSTEP    DLT=[0(0.02)1.0];
(6)  POINT    P1=(0.0,0.0),P2=(1.0.0,0),
(7)           P3=(1.0,1.2),P4=(0.0,1.2);
(8)  REGION   BOTTOM=LN(P1 ,P2), TOP=LN(P4,P3),
(9)           LEFT=LN(P1,P4), RIGHT=LN(P2,P3);
(10) BOUND    BOTTOM+TOP+LEH+RIGHT;
(11) MESH     BOTTOM=D(10), TOP=D(10),
(12)          RIGHT=D(10), LEFT=D(10);
(13) VAR      TP,TO;
(14) CONST    A=0.5;
(15) ICOND    TO=1OO;
(16) BCOND    TP=200
(17)          NGRAD(TP)=O AT LEFT+ RIGHT,
(18)          NGRAD(TO)=LMD*TOLD+MU AT TOP;
(19) SCHEME;
(20)     ITER NT UNTIL NT GT 50;
(21)          TO=TP;
(22)          SOLVE  TP  OF
(23)              (TP-TO)/DLT = A*LAPL(TP)
(24)                 BY 'ICCG';
(25)     END ITER;
(26) END SCHEME;
(27) END;
```

Fig. 2.5 DEQSOL Description for Heat Diffusion Problem by FEM

(2) Description Example 2: Non-linear Impurity Diffusion Problem

This is a more complex and realistic problem than the above heat conductivity example, which appears in an impurity diffusion simulation of the LSI process CAD. The impure material doped on the surface of an LSI spreads into the silicon body. The impurity density $N(x,y,t)$ satisfies the following equation:

$$\frac{\partial N}{\partial t} = div\left(D \cdot grad(N)\right) \pm div\left(\frac{D+N}{\sqrt{N^2 + 4 \cdot Ni^2}} \cdot grad(N)\right)$$
$$D = Di \cdot \left(1.0 + \frac{N}{Ni}\right) + Di^2 \cdot \left(\frac{N}{Ni}\right)^2 \qquad (2.4)$$

where Ni is the intrinsic career density, and D is the diffusion coefficient.

The equation (2.4) is non-linear with respect to *N* because the diffusion coefficient *D* depends on the impurity density *N*. The user does not have to take the non-linearity into account as long as the explicit method is adopted because the value of *N* in the previous iteration can be used to evaluate the right hand side of the equation. However, as was mentioned in the previous example, since the explicit method often fails to exhibit satisfactory convergence for complex problems, we have to rely on a more efficient implicit type of algorithm to solve a non-linear PDE like this. One of the algorithms known to be effective to deal with strong non-linearity is the Newton–Raphson method. We first calculate the derivative of $\partial D/\partial N$, and denote the resulting equation as *DP*.

$$\frac{\partial D}{\partial N} = Di \cdot \left(\frac{N}{Ni} \right) + 2 \cdot Di^2 \cdot \frac{N}{Ni^2} = DP \tag{2.5}$$

Let δN be a variation variable of *N*, then $D(N+\delta N)$ can be approximated by taking the first term of the Taylor's expansion as follows.

$$D(N+\delta N) \approx D+(dD/dN)\cdot\delta N = D+DP\cdot\delta N \tag{2.6}$$

```
(1)  SCHEME;
(2)    ITER CNT0 UNTIL CNT0 GT 20;
(3)      DO=DI*(1.0+N/NI)+D1**2*(N/NI)**2;
(4)      ITER CNT1 UNTIL VNORM LT EPS1;
(5)        D=DI*(1.0+NNEW/NI)+Di**2*(NNEW/NI)**2;
(6)        DP=DI/NI+2*DI**2* NNEW/NI**2;
(7)        SOLVE DN OF
(8)           (NNEW+DN-N)*DLT
(9)           =1/2*(DIV(D*GRAD(NNEW))
(10)          +DIV(DP*DN*GRAD(NNEW))
(11)          +DIV(D*GRAD(DN))+DIV(DO*GRAD(N)))
(12)              BY 'ILUBCG' WITH (EPS(1.0D-8));
(13)       NNEW=NNEW+DN;
(14)       CALL NORM2(S1,NNEW);
(15)       CALL NORM2(S2,DN);
(16)       VNORM=S2/S1;
(17)     END ITER;
(18)     N=NNEW;
(19)     PRINT N AT WHOLE EVERY 5 TIMES;
(20)   END ITER;
(21) END SCHEME:
```

Fig. 2.6 Impurity Diffusion Problem

Substituting (2.6) to the diffusion term of (2.4) yields

$$div(D(N+\delta N)\cdot grad(N+\delta N)) = div(D\cdot grad(N))+div(DP\cdot\delta N\cdot grad(N)) \\ +div(D\cdot grad(\delta N))+div(DP\cdot\delta N\cdot grad(\delta N)) \qquad (2.7)$$

By eliminating the negligible last term of (2.7), a linier PDE with regard to the variation variable δN is obtained. The SCHEME block based on the Newton–Raphson applied diffusion equation is shown in Fig. 2.6. In order to deal with time dependency, the Crank–Nicholson method is also introduced. The iteration loop (2) is to push time step forward by updating the value of `N`, whereas the loop (4) is to obtain a converged result of `DN` with Newton–Raphson method to update the value of `NNEW` for each time step. At each iteration inside the double loop, the SOLVE statement (7) specifies the corresponding matrix is formed with regard to `DN` and solved by the bi-conjugate gradient matrix-solution library with preconditioning by an incomplete LU factorization (ILUBCG).

2.5 Code Generation Method of the DEQSOL Translator

2.5.1 Outline of the Translator and the Structure of the Generated Code

The DEQSOL translator consists of two main components; PARSE and CODEGEN. PARSE reads in a DEQSOL description, checks the syntax of the statements, analyses the consistency of the variables, diagnoses the compatibility of the boundary and initial conditions with the PDE, and finally converts the description to an internal data structure called a “dictionary”. CODEGEN then looks up the dictionary and generates a FORTRAN code.

The structure of the generated code is shown in Fig. 2.7. The generated code starts with CONST, VAR and INIT statements for declaration of the variables and setting up the initial values, then looks up the SCHEME block to generate an overall control structure such as the time loop or non-linear loop corresponding to the ITER statements. Code fragments for simple assignment statements that do not include differential operators are also generated and directly placed in the corresponding loop structure. When CODEGEN encounters an assignment statement or SOLVE statement including one or more differential operators, it generates a subroutine in which the equation is handled. For the SOLVE statement in particular, CODEGEN discretizes the PDE in accordance with the

specified discretization method and generates code that calculates the value of the matrix and constant vector component and issue a call statement to a matrix-solution library.

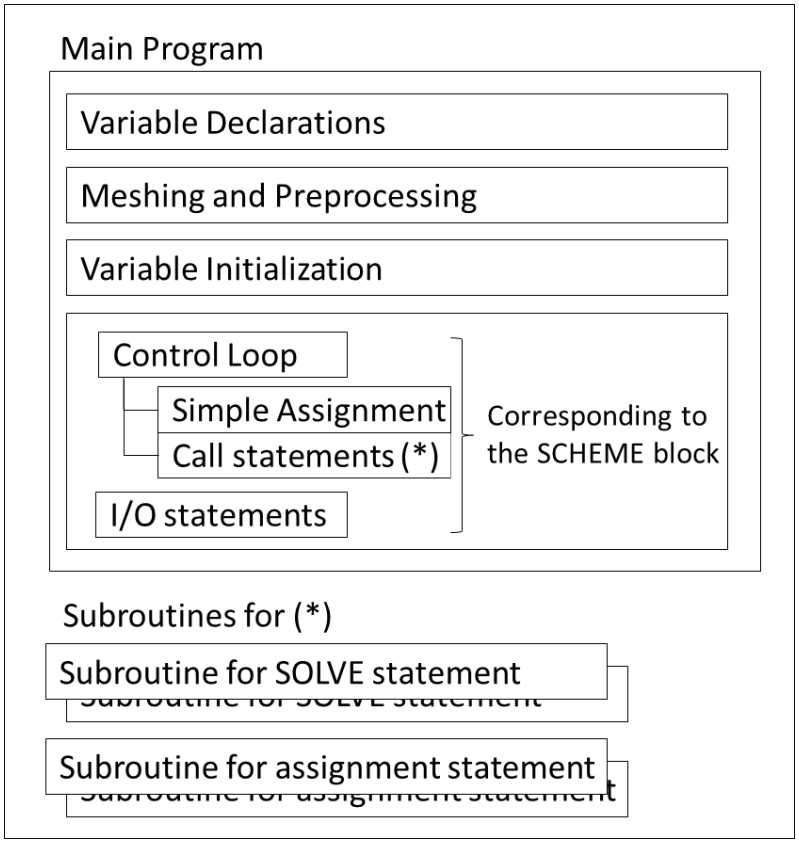


Fig. 2.7 Structure of generated code

2.5.2 Code Generation for FDM

The DEQSOL translator discretized PDEs appearing in the assignment statement or SOLVE statement by using the built-in discretization rule. The fundamental functions of the translator are symbolic manipulation of the differential operators according to the discretization rule as well as the break down of the region to compose maximum sized sub-regions in which the discretized equation becomes identical.

The symbolic manipulation of the differential operators takes place as follows in the DEQSOL FDM translator:

(1) Discretization is implemented as the recursive procedure on the given differential operators by applying the rule from the inner operator to outer one, thus all orders of

PDEs can be handled in a consistent manner.

(2) Internal discretization rule is based on the 1/2 index technique;

$dp/dx = (p(i+1/2) - p(i-1/2)) / \Delta x$

rather than

$dp/dx = (p(i+1) - p(i-1)) / 2 \cdot \Delta x$

which contributes the discretized equation to become identical to what resulting from the control volume method that is known to have a preferable nature of preserving the quantity such as energy, momentum or mass.

The typical flow of the discretization method is shown in Fig. 2.8 by taking a diffusion equation as an example. A term in a given PDE such as $div(D \cdot grad(N))$ is examined in turns, and the discretization rule is applied recursively from the inner operator to the outer. In Fig. 2.8, the internal operator $D \cdot grad(N)$ is first discretized, then discretization process proceeds to the outer $div(...)$ operator. The discretization is concluded by replacing the remaining 1/2 indices with the average of both sides of the index. Therefore 1/2 index has been used as an expedient to derive the final discretized form that does not involve the half indices.

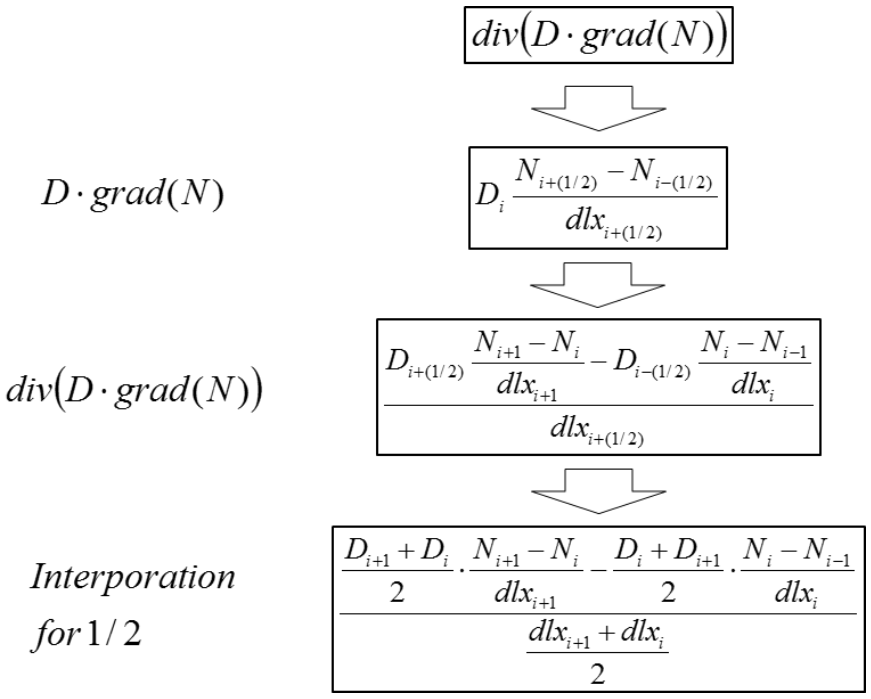


Fig. 2.8 Nested Discretization for Differential Operators

Another key function of the code generation is the break down of regions. This procedure is demonstrated using the case with a SOLVE statement in Fig. 2.9. First, the region is divided into several sub-regions according to the regional shape and the kinds of boundary conditions. The domain in Fig. 2.9 is divided into 6 sub-regions, 5 of them are for the boundary points and the remaining is for the inner ones. Each sub-region comprises a unit where the identical pattern of linear equations is formed by discretizing the PDE on every mesh point. Therefore calculation of the matrix and the RHS vector components can be handled by generating single DO loop for each sub-region.

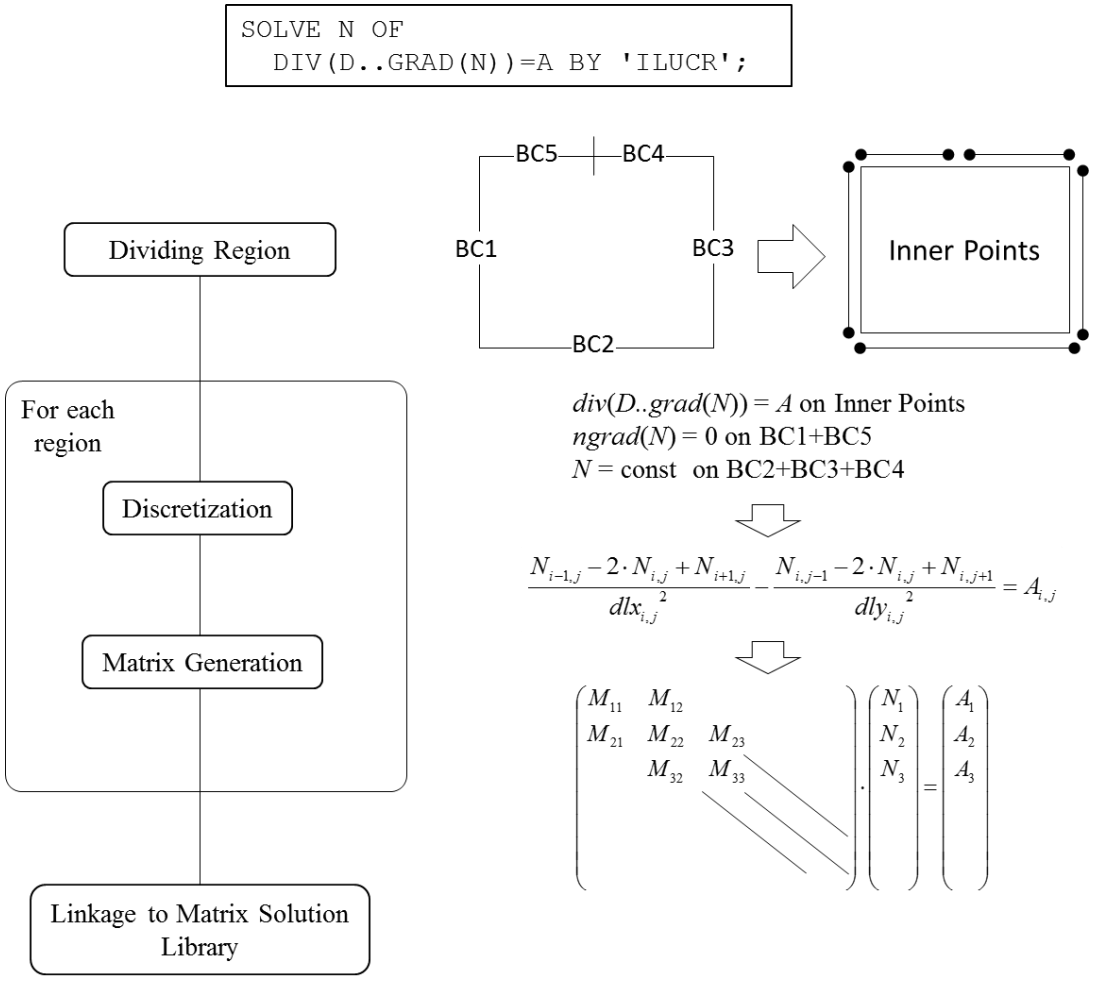


Fig. 2.9 Discretization and Matrix Formation in FDM

The DEQSOL translator discretizes the PDE according to the rule as described above. It then generates a part of the linear equation and creates the FORTRAN code that calculates the corresponding part of the total matrix and RHS vector elements of the

linear equation system. After the whole matrix and the RHS vector are comprised, it is handed over to an external matrix solver of the end-user's choice for the result.

A part of the generated FORTRAN code consisting of the matrix formation and linkage to the matrix-solution is shown Fig. 2.10. Each of the innermost DO loops can be readily vectorized by the vector compiler since there is no data dependency or no conflicting data assignment taking place in each loop.

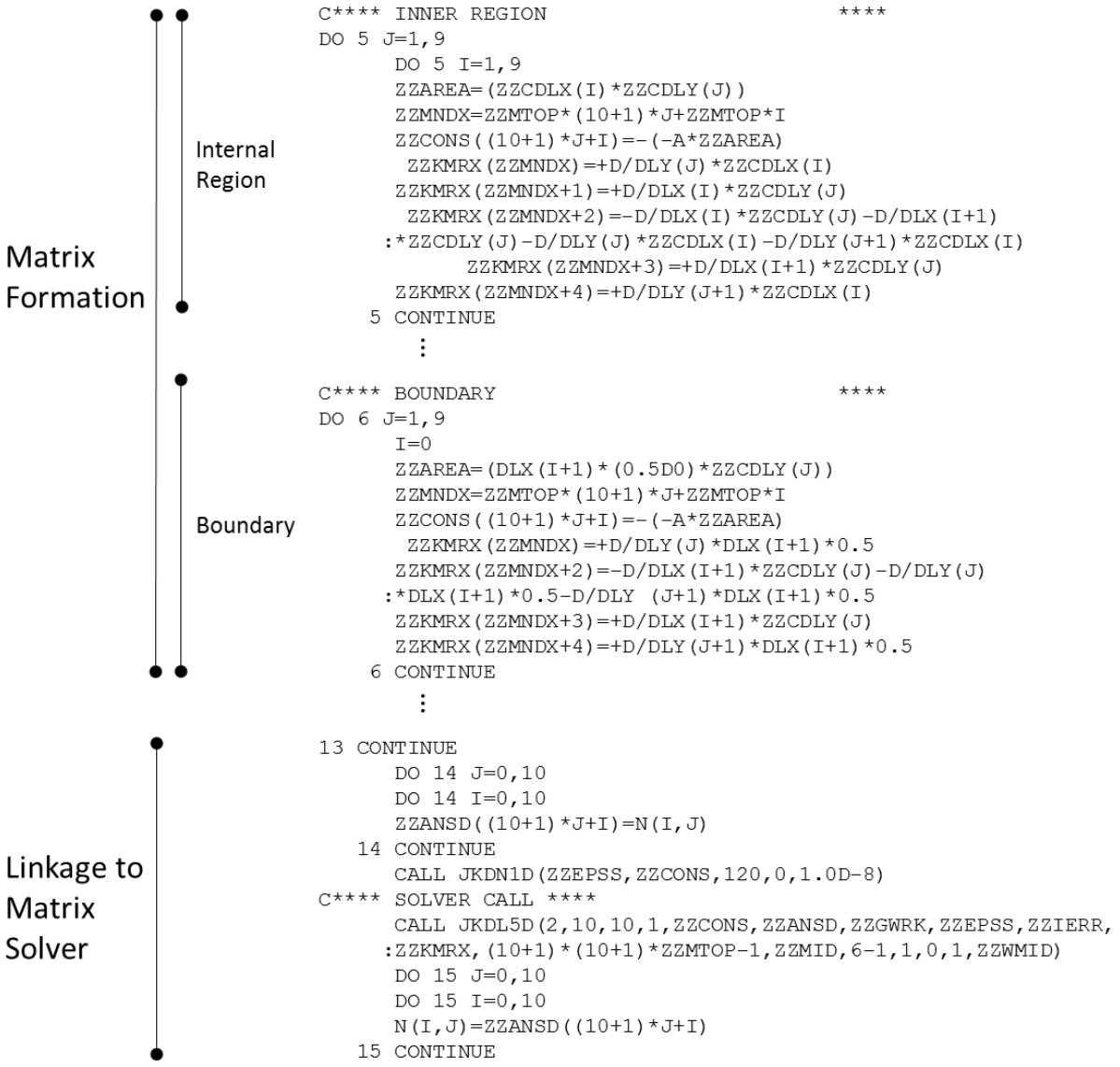


Fig. 2.10 Example of generated FORTRAN code for FDM

2.5.3 Code Generation for FEM

(1) Overall Process

A detailed discussion of the automatic discretization process for FEM is presented in this Section. Instead of applying replacement rules to each differential operator as was done in FDM discretization, a discretization method known as the "Galerkin method of weighted residuals" is applied to the PDE by means of symbolic manipulation to obtain the evaluation equation of the matrix. The discretization is thus targeted at PDEs that can be treated as the generalized convection–diffusion equations because most of the physical phenomena effectively formulated by the Galerkin method can be broken down to a combination of this type of equation.

Consider the generalized convective diffusion equation for a two-dimensional steady problem that can be expressed as

$$div(-k(x,y)\cdot grad(u)+\boldsymbol{b}(x,y)\cdot u) + c(x,y)\cdot u = f(x,y) \tag{2.8}$$

where k, c and f are scalar functions, and $\boldsymbol{b}$ is a vector function to be specified by the user. Equation (2.8) is defined in the region Q and satisfies the following boundary conditions:

$$u = 0 \quad \text{(Dirichlet boundary condition)} \quad \text{on } G1 \tag{2.9}$$

$$(-k\cdot grad(u)+\boldsymbol{b}\cdot u)\cdot \boldsymbol{n} = B \quad \text{(Neumann boundary condition)} \quad \text{on } G2 \tag{2.10}$$

where $\boldsymbol{n}$ is the normal vector to the boundary G2. An approximate solution of the variable u is expressed to be

$$u = \sum_{j=1}^{n} u_j \varphi_j(x,y) \tag{2.11}$$

where φ_j is the function belonging to a set that is complete in Q that is known to be an interpolation function, and u_j is the value of the variable to be solved at the j-th node. By multiplying both sides of equation (2.8) by φ_i that is known to be a weight function, and integrating it over the region Q, we can get a weighted residual form of the PDE. Since normally the interpolation function φ_j is piece wise linear, 2nd order derivative becomes indefinite on the edge of the function. Application of the Green's theorem to the residual form can convert the 2nd order derivative to a combination of 1st order derivative and boundary integration to reach a weak form of the equation (2.12). Conveniently also, the resulting boundary integration term normally coincides with the Neumann type of boundary condition (2.10), which largely simplifies the evaluation of this term. At this

stage, since all the derivatives regarding φ have been converted to those of 1st order, we can avoid the problem of the derivative becoming indefinite. It should also be noted that the sets of interpolation functions and the weight functions are chosen to be identical so that the resulting matrix becomes square and symmetrical (in absence of convection term).

$$\begin{aligned} &-\int_Q ((-k \cdot grad(u) + \boldsymbol{b} \cdot u) \cdot grad(\varphi_i))dQ \\ &+\int_G ((-k \cdot grad(u) + \boldsymbol{b} \cdot u)\varphi_i) \cdot \boldsymbol{n}\, dG + \int_Q (cu\,\varphi_i)\, dQ = \int_Q (f\varphi_i)\, dQ \end{aligned} \tag{2.12}$$

Substituting equation (2.10) and (2.11) into equation (2.12) yields

$$\begin{aligned} &\sum_{j=1}^{n} [\int_Q (k \cdot grad(\varphi_j) \cdot grad(\varphi_i) - \boldsymbol{b}\varphi_j grad(\varphi_i) + c\varphi_j\varphi_i)dQ] \cdot u_j \\ &= -\int_{G2} BdG + \int_Q f\varphi_i dQ \end{aligned} \tag{2.13}$$

Equation (2.13) can then be written in a matrix form as $A \cdot u = q$ where.

$$\begin{aligned} A_{ij} &= \int_Q (k \cdot grad(\varphi_j) \cdot grad(\varphi_i) - \boldsymbol{b}\varphi_j grad(\varphi_i) + c\varphi_j\varphi_i)dQ \\ q_i &= -\int_{G2} BdG + \int_Q f\varphi_i dQ \end{aligned} \tag{2.14}$$

The DEQSOL translator performs the above Galerkin discretization process symbolically to obtain the discretized equation (2.14). In order to determine all the element values of *A* and *q*, the integrals defined in equation (2.14) can be evaluated using the Gauss-Legendre numerical integration procedure that is performed element-by-element. The DEQSOL translator generates FORTRAN code to evaluate the above integrations that eventually form matrix *A* and RHS vector *q*. Once these are completed, an external matrix solver of the user's choice is called to obtain the result *u*.

The symbolic manipulation explained above has been achieved in the DEQSOL translator by making use of following key functionalities:

(A) Multiplication

This functionality is typically activated in the process of multiplying the PDE with the weight function as in (2.12).

(B) Pattern matching

In case applying Greens theorem or incorporating Neumann boundary condition to the integrated form, it is necessary to search for and pinpoint a particular pattern out of

the terms in the equation. The symbolic manipulation in the translator is equipped with a pattern matching functionality that works to find out special patterns to which the theorems to be adopted.

(C) Replacement

When the pattern is matched, the appropriate theorems need to be applied. Such action is generally performed by replacing the term in question with another specified in the theorem.

The basic idea for the code generation is similar to that of FDM. Total matrix A_{ij} is constructed by the contributions from the region integral terms and the boundary integral ones. The former are related to all nodes, whereas the latter is concerned with only the nodes on the boundary. Also the Dirichlet boundary condition needs to be imposed by replacing the relevant row and column to with 0.0, diagonal element with 1.0 and the RHS vector element with the specified value. Hence force, the contribution of the region integral terms can be calculated with a single DO loop for all nodes, and the contribution of the boundary integral terms needs to be calculated with separate DO loop for each boundary condition.

(2) Vectorization Technique of the Generated Fortran Code

Since FEM program requires more complex loop structure than FDM, we need to take special consideration to how to arrange the loops and the data structure suited to it. The procedure for forming matrix A with no consideration for the vectorization may be presented as follows:

```
(Procedure 1)
for all elements do begin
    for i:=1 to N_nd do begin
        for j:=l to N_nd do begin
            for k:=l to N_sp do begin
                Evaluate the discretized equation corresponding to
                    the element (i,j) of the element matrix on the k-th sampling point
                Add the resulting value to the element matrix;
            end;
        end;
    end;
    Add the element matrix to the global matrix;
end,
```

where N_{nd} is the number of nodes for each element, and
N_{sp} is the number of Gauss-Legendre sampling points for numerical integration.

In procedure 1, the innermost loop is not long enough to exploit the performance of vector processors since the typical number of integration points N_{sp} is merely 4 or 9 in two dimensional case and 8 or 27 in the three dimensional case. It is much preferable that the loop for all elements, that is normally order of 10^2 or more depending on the problem size, be placed within the innermost one. However, the simple exchange of loops causes unsuitable data dependency in the innermost loop. The above data dependency occurs only in the assembly of the global matrix, and the element matrices themselves can be calculated concurrently. Taking all these facts into account, the following loop structure is developed:

```
(Procedure 2)
for k:=l to N_np do begin
    for i:=1 to N_nd do begin
        for j:=l to N_nd do begin
            for all elements do begin
                Evaluate the discretized equation corresponding to
                    the element (i,j) of the element matrix on the k-th sampling point
                Add the resulting value into the work array for
                    the specific element (i,j) of all element matrices;
            end;
            Add data of the work array to the global matrix; *
        end;
    end;
end.
```

In procedure 2, the specific element (i,j) of all element matrices is first stored in a temporary work array. Assembly into the global matrix is carried out after the innermost loop exited (as denoted * in Procedure 2). In this manner, calculations of all element matrices are vectorizable. Procedure 2 can be highly vectorized, though the matrix assembly process remains un-vectorizable. This is because the number of computational operations required in the matrix assembly is much smaller than that in calculations of element matrices for most of the numerical schemes.

2.6 Advanced Solution Functions

In order to generate matrices for wider ranges of PDEs especially those appearing in the domains of structural analysis and fluid dynamics, we equipped DEQSOL with advanced functions that can handle simultaneous PDEs and additional techniques such as up-winding and lumping so that the system could form appropriate matrices for complex PDEs.

2.6.1 Implicit solution function for simultaneous PDE

An example of simultaneous PDE consisting of three equations and three variables can be written as

$$F(A,B,C) = 0,\ G(A,B,C) = 0,\ H(A,B,C) = 0$$

where F, G, and H are differential operators, and A, B, and C are the valuables to be solved. One possible numerical scheme to solve this PDE is shown in Fig. 2.11. Each equation is solved successively and iteratively until a convergence condition is met. If the coupling relationship among the variables is weak, this scheme can successfully reach an appropriate solution within a reasonable period of time. However, when the coupling relationship is strong or the dominating variable in each PDE is not clear, this iteration necessitates long time for convergence, or does not converge at all.

```
SCHEME;
     AOLD=AO; /*INITIAL SETTING*/
     BOLD=BO; /*INITIAL SETTING*/
     COLD=CO; /*INITIAL SETTING*/
     ITER Nl UNTIL NORY LT eps;
         SOLVE A OF F(A,BOLD,COLD)=0  BY 'ILUBCG';
         SOLVE B OF G(AOLD,B,COLD)=0  BY 'ILUBCG';
         SOLVE C OF H(AOLD,BOLD,C)=0  BY 'ILUBCG';
         CALL NORM2(A,B,C,AOLD,BOLD,CDLD,NORM);
         AOLD=A;
         BOLD=B;
         COLD=C;
     END ITER;
END SCHEME;
```

Fig. 2.11 Numerical scheme for simultaneous PDEs (successive assignment)

To cope with such a situation, the implicit solution function is extended such that plural coupled variables and PDEs are specifiable in a single SOLVE statement with appropriate boundary conditions specified using the UNDER clause. This is shown in Fig. 2.12. By using this clause, the coupled boundary conditions can definitely be assigned to each SOLVE statement.

```
SCHEME;
     SOLVE A,B,C OF
         F(A,B,C) = 0,
         G(A,B,C) = 0,
         H(A,B,C) = 0
              UNDER BCS1
              BY 'ILUBCG';
END SCHEME;
```

Fig. 2.12 Numerical scheme for simultaneous PDEs (simultaneous SOLVE)

The advanced implicit solution function discretizes the coupled PDE by treating them as one system of linear equations for coupled variables and generates an extended matrix with regard to all the variables specified in the SOLVE statement. In this case, the form of the generated matrix depends on the order in which the linear equations and discrete variables are arranged. Suppose F_i, G_i, H_i and A_i, B_i, C_i are discretized equations and variables at point i for the PDE. The DEQSOL translator generates a code where the indices are arranged in the order of (F_i, G_i, H_i) (i=1 to N) and (A_i, B_i, C_i) (i=1 to N). This arrangement of the indices will create the matrix with block diagonal form that is shown in Fig. 2.13. This formation helps to maintain the dominance of diagonal elements that is a preferable characteristic for iterative methods, such as InComplete Conjugate Gradient (ICCG), Bi-Conjugate Gradient (BCG), or Generalized Minimum RESidual (GMRES) to converge. Furthermore, the generated matrix is manipulated into the compressed form which retains only non-zero elements to save the memory space to store the extended matrix.

The DEQSOL translator calculates the necessary memory size to contain the integrated matrix. This size depends on the number of coupled systems involved for each SOLVE statement in the given numerical algorithm. Only the maximum region is secured, and it is shared alternatively among all SOLVE statements.

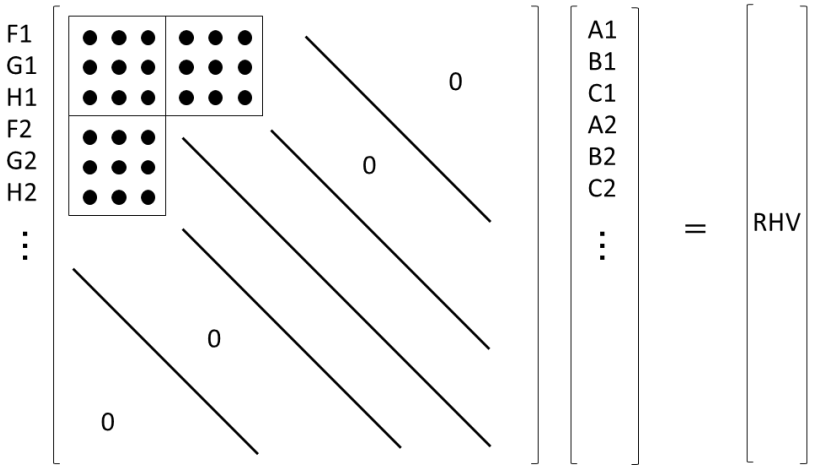


Fig. 2.13 Matrix formation for simultaneous SOLVE statement

2.6.2 FEM Capabilities to Generate Matrices for Fluid Dynamics Problems

Fluid mechanics is one of the most important applications of numerical simulation that appears not only in the design process of automobiles, ships, and aircrafts. Since the nature of the governing PDE of fluid dynamics, Navier–Stokes equation, is known to be mathematically complex, and the relationship among the variables is intertwined and non-linear, a variety of numerical schemes have been invented and need to be examined depending on the characteristics of the problem. Because of the complex nature of the PDE, FDM used to be commonly used to discretize the PDE and to generate the corresponding matrices. However, as the structure of the industrial products becomes sophisticated and precision of the simulation for such products becomes critical, FEM is gaining popularity as a preferable method because it has more compatibility and adaptability with complex region shapes and mesh topology. To cope with this trend, the necessary functionalities required to describe fluid dynamics problems in FEM were studied and the following features have been extracted.

(1) Mixed-order Interpolation

The mixed-order interpolation is used in primitive variable (velocity and pressure) formulations to eliminate the spurious pressure oscillations (that appears as a checkerboard pressure pattern) [64]-[66]. Mixed-order interpolation means the order of the interpolation function for pressure should be lowered than that for velocity.

(2) Upwind Approximation

Upwind approximations are used to solve convection-dominated problems with numerical stability. Numerous approximation techniques have been proposed among which the most promising in the fluid domain is the streamline upwind approximation [67][68]. This enables highly accurate simulation by controlling the amount of the artificial diffusion so that it affects only in the direction of the flow streamline.

(3) Matrix Lumping

The coefficient matrix is generated as the result of the finite element discretization of an assignment statement. Lumping technique diagonalizes the mass matrix and enables the assignment statement to be solved explicitly without intervention of matrix solution. Matrix lumping is an essential numerical technique for the fast processing of assignment statement. Lumping is also effective to numerically stabilize the explicit schemes.

In order to support the numerical techniques presented above, DEQSOL is equipped with the following capabilities:

(A) Function for Mixed-ordered Interpolation

Second-order isoparametric elements are introduced in addition to the conventional linear elements. The ELMTYPE statement is introduced in DEQSOL to specify interpolation functions for each problem's variables. For example, when quadratic elements for the velocity variable v and linear elements for the pressure variable p are used, that is expressed as,

```
ELMTYPE QUADRIC FOR V LINEAR FOR P;
```

When this statement is omitted, all variables are interpolated using linear elements.

(B) Function for Upwind Approximation

The streamline upwind approximation is introduced. This approximation modifies the weighting function in the conventional Galerkin formulation to get more influence from the upwind side. The weighting function φ is shifted to $\varphi+v_e \cdot grad(\varphi)$, where "$v_e$" is the vector that controls the direction and amount of shifting. The upwind approximation can be specified in the SOLVE statement or the assignment statement. For example, the upwind approximation is specified for the convection term of the transport equation,

$$\partial N/\partial t = -div(-k \cdot grad(N)) - v \cdot grad(N) \qquad (2.15)$$

This equation is expressed in the assignment statement as,

```
NNEW=NOLD-DLT*(DIV(-K*GRAD(NOLD))
                    +V..GRAD(NOLD)[UP(VE)]);
```

where “UP” is the key word indicating the upwind approximation. VE is the vector that controls the direction and amount of the upwind approximation. An appropriate value for VE can be set in the earlier part of the DEQSOL description.

(C) Function for Matrix Lumping

Lumping is specified for variables in the SOLVE or the assignment statement. When lumping is applied for variables in the transport equation (2.15), it is expressed as,

```
NNEW[LM]=NOLD[LM]
     −DLT*((DIV(-K*GRAD(NOLD))+V..GRAD(NOLD));
```

where the keyword “LM” indicates that lumping is applied to the variable.

```
VAR       U,V,P,UO,VO,PO,UM,VM,UT,VT,PHI;
VECT      VV=(U,V), VE=(UT,VT);
ELMTYPE   LINEAR FOR U,V,UO,VO,UM,VM,PHI
          FLAT FOR P,PO;
SCHEME;
     ITER NT UNTIL NT EQ 10000;
         UT=0.5*DLT*UO; VT=0.5*DLT*VO;
         UM[LM]=UO[LM]-DLT*(VV..GRAD(UO))[UP(VE)]
              -DLT*(1/RHO*DX(PO)-NU*LAPL(UO));
         VM[LM]=VO[LM]-DLT*(VV..GRAD(VO))[UP(VE)]
              -DLT*(1/RHO*DY(PO)-NU*LAPL(VO));
         SOLVE PHI OF LAPL(PHI)=-DX(UM)-DY(VM)
              BY 'ILUBCG';
         U[LM]=UM[LM]+DX(PHI);
         V[LM]=VM[LM]+DY(PHI);
         P=PO-RHO*PHI/DLT;
         UO=U; VO=V; PO=P;
     END ITER;
     PRINT U,V,P;
END SCHEME;
```

Fig. 2.14 DEQSOL Description for SMAC Method in FEM using Advanced Functions of Mixed-ordered Interpolation [ELMTYPE], Matrix lumping [LM], and Upwinding Approximation [UP]

An example is presented to illustrate the application of the above functions to practical flow problems. Consider a two-dimensional unsteady incompressible viscous flow. The governing equations, a variation of the Navier–Stokes equation, are:

$$
\begin{aligned}
&\partial u/\partial t = -\boldsymbol{V}\cdot grad(u) - 1/\rho\cdot(\partial p/\partial x) + \nu\cdot lapl(u)\ ,\\
&\partial v/\partial t = -\boldsymbol{V}\cdot grad(v) - 1/\rho\cdot(\partial p/\partial y) + \nu\cdot lapl(v)\ ,\\
&div(\boldsymbol{V}) = 0,
\end{aligned}
\tag{2.16}
$$

where $\boldsymbol{V}$ is the velocity (u,v), ρ is the density, and ν is the kinematic viscosity. Here, the simplified Marker and Cell (SMAC) scheme is used to advance time steps. The SMAC scheme is one of the most successful algorithms in fluid analysis to handle the cases with high Reynolds Number [65]. The variable definition statements and the SCHEME statement for this problem are shown in Fig. 2.14. A mixture of 1st order linear element and 0th order constant element is used for velocity (u,v) and pressure p. Matrix lumping is used to accelerate updating the velocity components u and v. As illustrated in the figure, the complex algorithm is expressed in a natural manner using DEQSOL's advanced functionalities for fluid dynamics.

2.7 Evaluation

In this section, an evaluation of DEQSOL, such as programming productivity and vectorization ratio of the generated code, is given by the application of DEQSOL to practical physical problems.

DEQSOL and its processing system have been in use since 1985 and applied to numerous practical problems. Applied fields include thermal conduction problems, convection and diffusion problems for LSI manufacturing, electric or magnetic field analysis, fluid dynamics analysis, and many others. From the viewpoint of numerical schema, explicit, implicit, and mixed methods for time dependent problems, Newton–Raphson's method for non-linear PDEs, successive assignment, and a collective solution method for simultaneous PDEs can be expressed, and the FORTRAN programs to form corresponding matrices can be generated successfully using the DEQSOL system.

2.7.1 Evaluation for Convection−diffusion Problems

Some evaluations using convection–diffusion type problems are shown in Table 2.3, where the descriptive efficiency ratio of DEQSOL (code lines) and the runtime efficiency of the generated FORTRAN code (vectorization ratio, acceleration ratio using

vector processor) for each problem are illustrated. In the table, the acceleration ratio refers to the speed-up rate for the generated FORTRAN code executed with a vector processor on the HITAC S-820 supercomputer against that when vector processing is turned off (i.e., the execution time with the scalar processor only).

(1) Productivity

Productivity when measured by the required code lines has been improved by approximately one order of magnitude over FORTRAN programming. The FORTRAN code lines in the table are those for hand crafted programming rather than code generated by the DEQSOL translator.

(2) Vectorization ratio

The vectorization ratio shown in Table 2.3 is measured on the HITAC S-820 supercomputer. Most of the generated FORTRAN codes have satisfactorily high vectorization ratio (90%–96%). When the code is executed on the vector processor, execution is accelerated 3.8 to 9.4 times faster than execution by scalar processing. The vectorization and acceleration ratios in the table do not include the contribution from the matrix library that was linked with the generated code. By binding the code with an effective matrix-solution library, we can expect even higher vectorization and acceleration ratios.

Table 2.3 Evaluation of DEQSOL for Convection–diffusion Problems

Method		FDM			FEM		
Problem Number		I	II	III	IV	V	VI
Problem Name		Impurity Diffusion	Beam in CRT	LSI Chip Cooling	JJ Device Cooling	Disk Head Magnetics	Electric Voltage Analysis
Dimension		2	2	2	2	3	3
Code Lines	DEQSOL	79	35	34	96	904	147
	FORTRAN	1312	370	347	1078	11588	1198
	Ratio	16.6	10.6	10.2	11.2	12.8	8.2
Perfor-mance	Vectorization Ratio	96	94	98	93	94	96
	Acceleration Ratio	8.2	5.7	6.1	5.1	7.7	7.6

2.7.2 Evaluation for Fluid Dynamics Problems

We conducted further evaluation to understand the enhancement in productivity and performance of matrix formation by DEQSOL using samples from the fluid dynamics domain, which tend to be significantly more complex than the convection–diffusion problem examined in Section 2.7.1. Note that the advanced FEM functionalities described in Section 2.6.2 were fully utilized. Two types of problems were chosen, i.e., stokes flow solved by direct method (problem numbers I and II), and incompressible flow solved by the SMAC method (problem numbers III and IV). For each type, we selected two- and three-dimensional cases. The characteristics of the problems are summarized in Table 2.4.

Table 2.4 Evaluation of DEEQSOL for Fluid Dynamics Problems using Advanced FEM Functionality

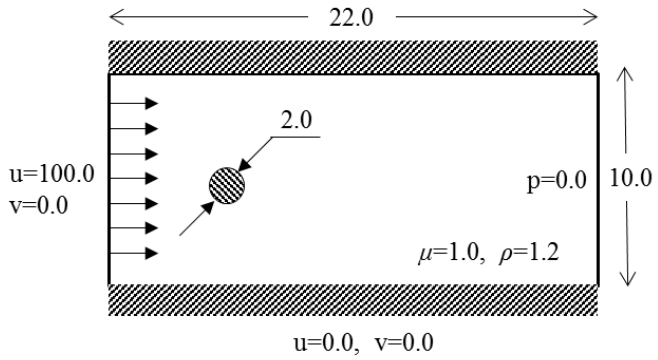
Problem Number	VII	VIII	IX	X
Problem Name	Backward Facing Step Flow	Backward Facing Step Flow	Flow Passing a Cylinder	Flow in a Bent Duct
Dimension	2	3	2	3
Time Dependency	Steady	Steady	non-steady	non-steady
Type of Flow	Stokes' flow	Stokes' flow	Incompressible viscous flow	Incompressible viscous flow
Reynolds Number	-	-	240	260
Solution Scheme	Simultaneous Solve	Simultaneous Solve	SMAC	SMAC

Table 2.5 shows the results of the performance evaluation of DEQSOL and its translator with respect to the aforementioned benchmark problems. The definition of the acceleration ratio and vectorization ratio are the same as in Section 2.7.1. The performance measurement is again performed on the HITAC-S820 supercomputer.

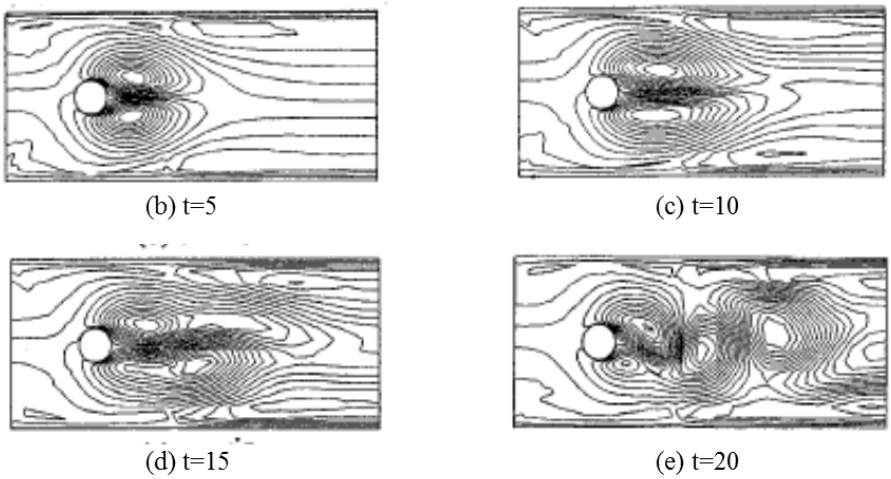
For all the problems examined, the vectorization ratio of the generated code was over 90%, and the acceleration ratio ranged from 3.70 to 11.0. According to the analysis of generated FORTRAN codes, all innermost loops were vectorized, with the exception of the assembly of the global matrix and vector, as well as the I/O process. The larger the problem becomes, the smaller the contributions from the above non-vectorizable processes will be. Therefore, the generated FORTRAN codes for larger problems are expected to achieve even higher acceleration ratio than the above cases.

Table 2.5 Evaluation of DEQSOL for Fluid Dynamics Problems

Method		FEM			
Problem Number		VII	VIII	IX	X
Problem Scale	Node	78	212	1008	1078
	Element	19	24	945	756
Lines of Code	DEQSOL	39	74	34	96
	FORTRAN	1021	1820	70	120
	Ratio	26.2	24.6	47.7	59.2
Perfor-mance	Vectorization Ratio	92	97	93	97
	Acceleration Ratio	6.6	7.2	3.7	11.0



(a) Problem Definition



(b) t=5 (c) t=10

(d) t=15 (e) t=20

Fig. 2.15 Example Simulation Result of Fluid Dynamics Problem

Fig. 2.15 shows a typical simulation result from problem IX, graphically depicted in a form of a streamline. In this figure, “t” is the normalized time expressed as the diameter of the cylinder divided by the inlet flow velocity. The streamlines clearly represent how asymmetric von Karman’s vortex street evolves over time.

2.7.3 Comparison with Other PSEs

In this section, we compare the descriptive capabilities of DEQSOL to those of existing systems. As per the benchmarking systems, we choose MATLAB and FlexPDE, because these systems are widely used PSEs for which enhancement efforts are still in progress.

A functional comparison of the systems is provided in Table 2.6. Comparison items include the dimension of solvable PDEs, discretization method, variability of numerical scheme, applicable physical domain, and other features.

Among DEQSOL, FlexPDE, and MATLAB, DEQSOL and FlexPDE can numerically simulate PDEs defined on a 2- or 3-dimensional domain. In contrast, MATLAB possesses more capabilities at a mathematical level, such as symbolic operation and matrix level manipulation. If the end user wishes to analyze the problem at a mathematical level or examine the nature of the matrix in an interactive manner, MATLAB would be an ideal choice.

FlexPDE is equipped with PDE templates with a pre-attached corresponding numerical library together with a fixed numerical scheme. Although this feature simplifies the description by avoiding manual selection or construction of a numerical scheme, it poses a limitation when the user wishes to develop a new or alternative scheme to tackle a problem not previously encountered. One of the clear advantages of FlexPDE is its ability to handle a moving boundary problem. This feature makes it a good candidate to solve a structural problem with large deformation.

When MATLAB is used with an external library, such as the NAG toolbox, the combined capability resembles that of DEQSOL or FlexPDE, as is shown in the last column of Table 2.6. In this situation, MATLAB can solve 1- and 2-dimensional PDEs in the convection−diffusion domain using the algorithm provided in the toolbox. However, as is the case with FlexPDE, the limitation of describing a variety of numerical schemes persists. The applicability of the system to a given problem is bound by the functions provided by the additional toolbox.

Table 2.6 Comparison of PSE Functionalities

		DEQSOL	FlexPDE	MATLAB	MATLAB with NAG toolbox
Approach/ Feature		Code Generation	PDE template + library	Symbolic Manipulation,	PDE template + library
Dimension of PDE		1-, 2-, and 3-dimentional	1-, 2-, and 3-dimentional	1-dimentional	1- and 2-dimentional
Discretization method		FDM/FEM	FEM	FDM	FDM/FEM
Variable numerical simulation scheme		yes	no	no	no
Domain	Convection−Diffusion problem	yes	yes	no	yes
	Fluid dynamics problem	yes	limited	no	limited
	Moving boundary problem	no	yes	no	no
Others	Interactive operation	no	no	yes	yes
	Matrix level manipulation	no	limited	yes	yes
	Symbolic operation of PDEs	no	no	yes	yes

Table 2.7 Applicability of PSEs to the Example Problems

	I	II	III	IV	V	VI	VII	VIII	IX	X
DEQSOL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
FlexPDE	✓	✓	✓	✓	✓	✓	–	–	–	–
MATLAB	–	–	–	–	–	–	–	–	–	–
MATLAB with NAG toolbox	✓	✓	✓	✓	–	–	–	–	–	–

The fundamental advantage of DEQSOL lies in the descriptive capability of numerical simulation schemes derived from the code generation approach. A scheme as

complex as the SMAC method can be described at a PDE level (as discussed in Section 2.6), and due to this feature, the corresponding matrix formation code can be generated flexibly. DEQSOL allows the user to examine various numerical schemes. This is particularly useful when an effective scheme to handle the problem is unknown, and the user wishes to explore new schemes with minimal changes to the script.

Table 2.7 evaluates the applicability of the PSEs to problems I–X in Section 2.7.2. Most PSEs can handle the convection−diffusion problems (I–VI) in various ways. DEQSOL shows a clear advantage for 3-dimensional and/or complex fluid dynamics problem domains.

2.8 Conclusion

In this chapter, we have proposed an efficient way to form matrices for physical problems by providing a high-level language and a corresponding translator dedicated to numerical simulation. We refer to the language and translator as the DEQSOL language and the DEQSOL translator, respectively. The design principle of the DEQSOL language and its specifications were explained in detail with examples. The architecture of the DEQSOL translator was explained with emphasis on code generation techniques for FDM and FEM discretization functionalities. The advanced features of DEQSOL to handle structural and fluid dynamics problems were also explained. Finally, we evaluated the DEQSOL system from the perspective of description efficiency and the performance of the generated matrix formation program in terms of vectorization ratio using convection–diffusion and fluid dynamics problems. The evaluation demonstrated that the end user can describe a physical problem in one-tenth of the FORTRAN code lines while gaining more than a 90% (typically 95%) vectorization ratio on a supercomputer.

Chapter 3

Improvement in Performance of GMRES(*m*) Method by Applying Genetic-Algorithm Inspired Enhancement to Restart Process

3.1 Introduction

For effective execution of numerical simulation, improving the performance and usability of the matrix-solution phase has significant importance. In this chapter we propose a matrix-solution method that combines GA and an existing matrix solver to achieve higher convergence performance [46]-[48].

In recent years, the nature of the matrices being treated in a numerical simulation becomes more complex, i.e., larger, sparser and often more ill-conditioned. To cope with such a situation, a number of matrix-solution methods have been proposed, among which the gradient type is known to be a good compromise between robustness, computational complexity, and memory usage as mentioned in Section 1.2.2. The GMRES(*m*) method was invented to be one of the most promising gradient methods to solve large-scale sparse matrices with better convergence and low memory consumption [28][29].

On the other hand, recent advent of the cloud computing is paving the way for abundant use of the computational resources to accelerate a simulation. One of the promising techniques to utilizing such resources is adoption of GA due to its compatibility with embarrassing parallelism, i.e., the chromosomes belonging to the same generation of GA can be readily distributed and processed in parallel. Although

several proposals are made to apply GA to numerical simulation and matrix computing in particular, most of such efforts were concentrating on the optimum choice of the matrix computing parameters such as the kind of the solver and the pre-conditioner, the dimension of Krylov subspace and the number of execution threads [35][36].

The aim of this chapter is applying a GA inspired technique to the choice of initial vectors, not to the calculation parameters, in order to clarify the impact of GA application to the matrix-solution process rather than parameter selection process, which leads to a proposal of an effective algorithm to crossover the chromosomes in the GA process.

The rest of this chapter is organized as follows. Section 3.2 reviews the existing algorithms of GMRES(*m*) and Look-back modification. Section 3.3 explains how GA can be applied to improve the performance of GMRES(*m*) by examining different algorithms for crossover process. Section 3.4 evaluates the proposed GA–GMRES(*m*) method using several example matrices having different characteristics. Finally we conclude this chapter in Section 3.5.

3.2 GMRES Method and Look-back Modification

3.2.1 GMRES(*m*) Method

GMRES method was developed in 1986 and a series of improvements has been invented since then to make it one of the most promising techniques to solve large-scale, non-symmetric, sparse matrices. The GMRES algorithm is described in a pseudo code:

Compute $r_0 = b - A \cdot x_0$, $\beta = \|r_0\|$, and $v_1 = r_0 / \beta$
Define the $(n+1) \times n$ matrix $\underline{H}_n = \{h_{ij}\}_{1 \le i \le (n+1),\ 1 \le j \le n}$, Set $\underline{H}_n = 0$
For j = 1,2, ... n (n: number of iterations till convergence) Do **
  Compute $\omega_j = A \cdot v_j$
  For i = 1,2,..., j Do
    $h_{i,j} = (\omega_j, v_i)$
    $\omega_j = \omega_j - h_{i,j} \cdot v_i$
  End Do
  $h_{j+1,j} = \|\omega j\|$. If $h_{j+1,j} = 0$ go to *
  $v_{j+1} = \omega_j / h_{j+1,j}$
End Do
*Compute $x = x_0 + V_n \cdot y$, where y = arg min $\| \beta e_1 - \underline{H}_n \cdot y \|$

One drawback of GMRES method is that it demands O(n) memory workspace and O(n^2) calculation complexity (n: maximum number of iteration) because it tries to make a new search vector orthogonal to all the previous search vectors, necessitating to remember all those vectors in the past iterations. GMRES(m) modification was proposed to cope with this shortfall. In this modification, the GMRES process is prematurely cut off after every m iterations and the resultant intermediate solution is used as the initial vector for the next restart stage. The GMRES(m) algorithm is formulated as follows:

(a) Choose an initial guess of $x_0^{(1)}$

(b) For l = 1, 2, … until convergence Do

(c) Solve $A \cdot x = b$ using m iterations of the GMRES method

(i.e., let $n := m$ in the loop ** in the above GMRES pseudo code)

with initial guess $x_0^{(l)} := x_m^{(l-1)}$

(d) End Do

The cost of each restart stage is limited within O(m^2) where m can be adjusted to balance the convergence speed and memory requirement. However, the introduction of the restart process gives rise to another problem of finding an appropriate value of m according to the characteristics of the matrix to be solved. The schematic of the GMRES(m) method is shown in Fig. 3.1 taking the first two restart stages as an example, where $x_a^{(b)}$ denotes an intermediate solution vector x at the a-th GMRES iteration in the b-th restart stage. Note that the initial guess of the Stage 2 (i.e., $x_0^{(2)}$) is carried over from the immediately previous result of the Stage 1 (i.e., $x_m^{(1)}$).

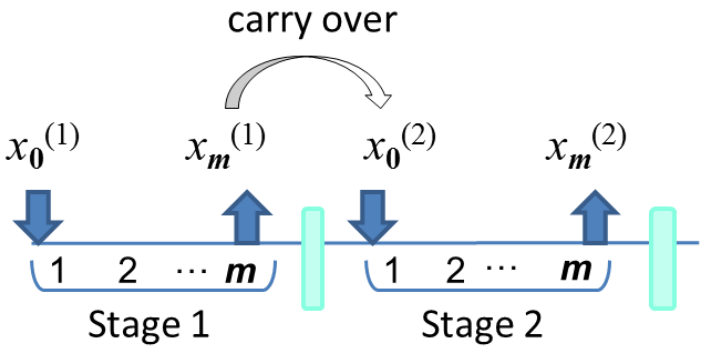


Fig. 3.1 Schematic of the GMRES(m) Method

3.2.2 Look-back GMRES(*m*) Method

As can be seen from the pseudo code line (c) in Section 3.3.1, the GMRES(*m*) iteration process consists of the following two phases:

I. Solve $A \cdot x = b$ by m iterations of GMRES calculations with the initial guess $x_0^{(l)}$ to get an approximate solution $x_m^{(l)}$.

II. Update the initial guess $x_0^{(l+1)} := x_m^{(l)}$ for next restart cycle.

There have been several approaches proposed to improve the efficiency of phase I, such as introduction of deflation techniques and techniques based on augmented Krylov subspace [69]. In addition to these, it is advantageous to use a better intermediate initial vector for each restart stage to accelerate convergence of a GMRES(*m*) calculation at phase II. Instead of using the solution of the previous stage directly as the next initial vector, several modifications are being proposed and it has become one of the recent focal points to improve the convergence performance of GMRES(*m*) method. A typical example of such modifications is the Look-back type of GMRES(*m*) method [70]-[72]. This method tries to improve the initial vector for each restart stage by utilizing the multiple intermediate results rather than by utilizing the intermediate result of the immediately previous stage only. The simplest algorithm using two previous results is obtained by replacing the step (c) of the GMRES(*m*) method with the following (c)'.

(c)' Solve $A \cdot x = b$ with m iterations of GMRES method
with initial guess $x_0^{(l)} = x_m^{(l-1)} + y^{(l-1)}$,
where

$$y^{(l)} = \mu^{(l)} \Delta x^{(l)}$$

$$\mu^{(l)} = \arg\min \| r_m^{(l)} - \mu A \cdot \Delta x^{(l)} \|$$

$$= (r_m^{(l)}, A \cdot \Delta x^{(l)}) / (A \cdot \Delta x^{(l)}, A \cdot \Delta x^{(l)})$$

Fig. 3.2 depicts the schematic of the Look-back modified GMRES(*m*) method taking the first three restart stages as an example, where $x_a^{(b)}$ denotes an intermediate solution vector x at the a-th GMRES iteration in the b-th restart stage. In order to obtain the initial guess of the Stage 3, the previous two intermediate results (i.e., $x_m^{(1)}$ and $x_m^{(2)}$) are referenced, μ is calculated with the formula in (c)', and $x_0^{(3)}$ is determined as an extrapolation or interpolation of $x_m^{(1)}$ and $x_m^{(2)}$ as shown in the lower part of Fig. 3.2.

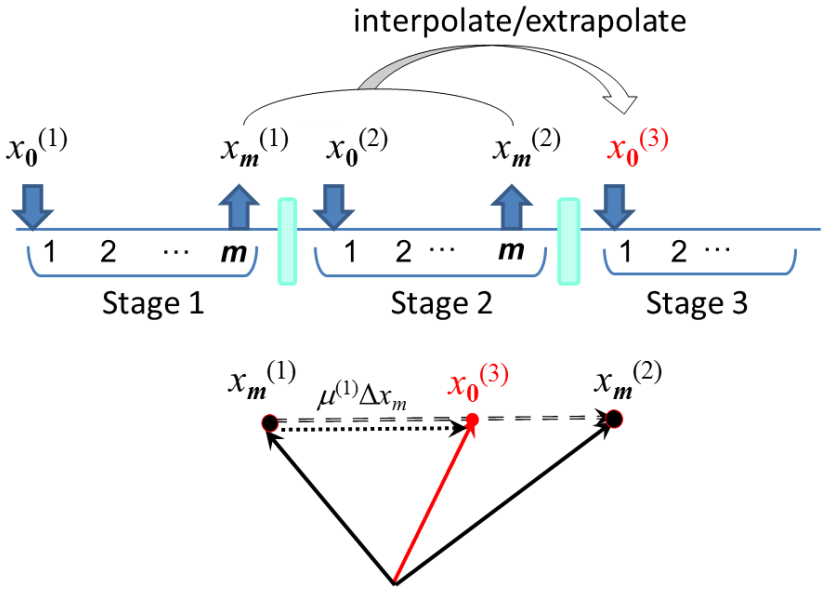


Fig. 3.2 Schematic of Look-back GMRES(*m*) Method

3.3 GA Application to GMRES(*m*) Method

3.3.1 Conventional GA application to Matrix Computing

Application of GA to the sparse matrix solvers has been proposed for the combination of calculation parameters such as the kind of solvers and pre-conditioners. Reference [35] proposes on-the-fly optimization of the simulator performance by dynamically adjusting the simulation parameters. The system has been combined with ExxonMobil's reservoir simulator and has successfully shown the performance improvement by 30% or more. Another attempt is presented where GA has been used to find the optimum combination of sparse solver and its parameters [36]. Supervised learning has been adopted to find the function of the sparse matrix features, which gives rise to a necessity of preparing large number of training sets although the implementation can be highly parallelized with 2048 computation cores.

In the preceding study [46], GA has been applied for the first time to the combination not only of the calculation parameters but of the initial vectors. This approach has presented successful convergence of GMRES(*m*) method with smaller *m* than the cases without GA. It also eliminates the necessity of prior learning because the choice of the parameters and initial vector can be achieved while GA process is on-going. However, since both the parameter and the initial vector are handled in GA process at the same time,

the impact of applying GA to the initial vector only was not made clear. Furthermore, to cross the chromosomes (i.e., initial vectors), simple one-point crossover algorithm was used where we thought there would be further opportunities of performance improvement. Therefore, we aim to clarify the impact of GA on creation of better initial vectors for the restart process as well as to propose an effective crossover algorithm to accelerate the convergence of the matrix solution by taking a hint from the Look-back modification of the GMRES(*m*) method.

3.3.2 The Algorithm of GA Application

In the proposed application of GA to the GMRES(*m*), the initial vectors of each restart step are regarded as chromosomes. *Nc* sets of initial chromosomes are randomly generated as the starting points. GMRES processes are executed until the solution vectors become stagnant, and the new initial vectors will be generated by crossing the resultant vectors of the previous stage.

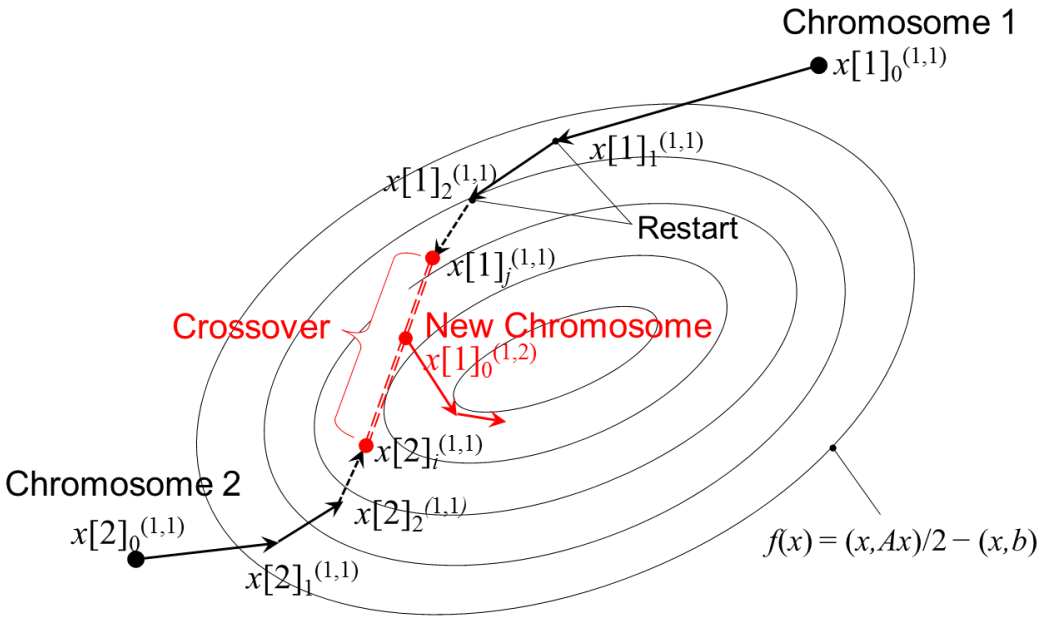


Fig. 3.3 Schematic of Crossover Process

The basic idea of this approach is shown schematically in Fig. 3.3. The solution process of a linear equation $A \cdot x = b$ is defined alternatively as a minimization of the functional:

$$f(x) = (x, A\cdot x)/2 - (x,b) \tag{3.1}$$

where (x, y) denotes the inner product of two vectors x and y. The contour of $f(x)$ = *constant* forms a hyper-elliptic surface in an n-dimensional space. In Fig. 3.3, the n-dimensional space collapses to a two-dimensional plane, and the number of chromosomes is limited to two (i.e., $x[1]_0^{(1,1)}$ and $x[2]_0^{(1,1)}$) for the sake of simplicity. Here $x[d]_a^{(b,c)}$ denotes an intermediate solution vector x at the a-th GMRES iteration in the b-th restart stage of the c-th GA generation with chromosome number d. Suppose the intermediate results derived from $x[1]_0^{(1,1)}$ and $x[2]_0^{(1,1)}$ will develop into $x[1]_i^{(1,1)}$ and $x[2]_j^{(1,1)}$ after the i-th and j-th GMRES stages, respectively, to experience stagnation. This is the point at which a GA process comes into play to cross $x[1]_i^{(1,1)}$ and $x[2]_j^{(1,1)}$) to generate a new chromosome $x[1]_0^{(1,2)}$. Using $x[1]_0^{(1,2)}$ and $x[2]_0^{(1,2)}$ (generated from another pair of chromosomes) as the initial vectors for the next generation, the calculation continues until the result x that minimizes $f(x)$ is reached.

The algorithm of the proposed method is described as below:

(a) Populate initial guess vectors $x[k]_0^{(1,1)}$
    with random numbers, where $k = 1\ldots Nc$
(b) For $i = 1, 2\ldots$ until convergence Do
(c) For $k = 1\ldots Nc$ Do
(d) For $l = 1, 2,\ldots$ until stagnation Do
(e) Solve $A\cdot x = b$
    using m iterations of the GMRES method
    with initial guess $x[k]_0^{(l,\, i)}$
(f) End Do
(g) End Do
(h) Cross $x[k]_m^{(l,\, i)}$ $[k = 1 \ldots Nc]$ to obtain new $x[k]_0^{(1,\, i+1)}$
(i) End Do

The loop (b) pushes the chromosome generation forward. The loop (c) is to scan the chromosomes. The loop (d) corresponds to the restart loop of the GMRES(m) calculation. Here, attention should be paid that the loop (c) can be readily parallelized in terms of k by taking advantage of the virtue of GA. A schematic of the algorithm is provided in Fig. 3.4.

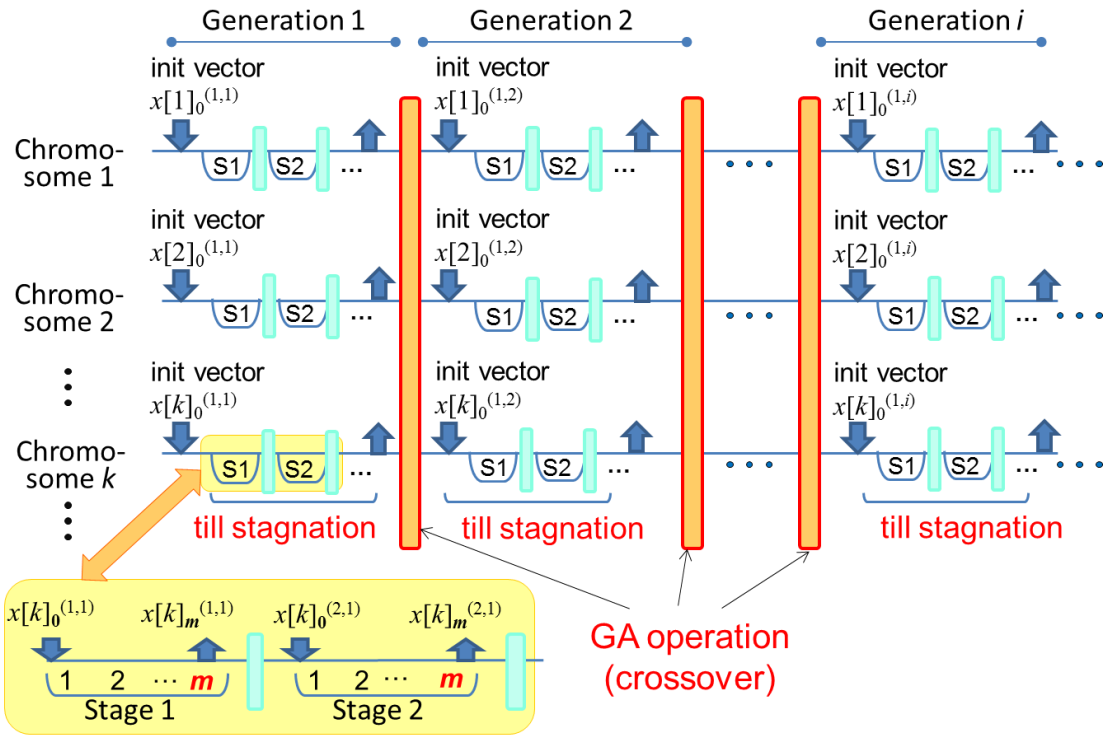


Fig. 3.4 Schematic of the GA–GMRES(*m*) Method

A possible alternative way for GA application is to let GA have the sole responsibility to evolve the chromosomes without having to rely on any matrix operation such as GMRES. The bottleneck with this approach is the sheer length of the chromosomes, turning it into a very massively high-dimensional problem beyond the reach of GA, unless a very large population (as also implied in [32]), and subsequently a very long computation time is deployed. Hence, the approach adopted here is not a pure GA, but rather a hybrid GA or a memetic algorithm. Note that hybrid GA has in fact seen a surge in recent years in other fields than matrix computing [33][34].

3.3.3 Influence of the Initial Vector to GMRES(*m*) Convergence

Prior to examining the impact of GA application to GMRES(*m*), we carried out a simple study about how much impact the choice of different initial vector would have upon the convergence process of GMRES(*m*) method.

The configuration of the numerical experiment is as follows. The test matrix is torso1 taken from the University of Florida sparse matrix collection*. The GMRES solver

* University of Florida sparse matrix collection homepage, http://www.cise.ufl.edu/research/ /matrixes/index.html.

is the one implemented in Xabclib library developed by Tokyo University*. The computer is Dell Inspiron with Core i5 2.5GHz CPU equipping 8GB memory, while the operating system is Ubuntu 10.4 with Intel Fortran compiler. The pre-conditioner is set to be ILU(0). To make the effect of adopting different initial vectors distinctive, the auto tuning feature of m in Xabclib (increasing m upon stagnation) has been turned off; i.e., a pre-fixed value of m is maintained throughout each execution. The RHS vector b of $A \cdot x = b$ is determined so that x is an n-dimensional vector of (1.0 ... 1.0) and b is the product of A and x. The components of the randomly generated initial vectors are evenly spread between −1.0E+1 and 1.0E+1. Four different values of m, namely 30, 50, 180 and 200, were chosen. For each value of m, five randomly generated initial vectors are prepared and GMRES(m) were performed.

The result of the experiment is shown in Fig. 3.5 to Fig. 3.8 in a form of convergence curves; i.e., the vertical axis indicates the normalized residual $||b-A \cdot x||/||b||$ at each restart stage whereas the horizontal axis corresponds to the number of the restart iterations. As can be seen from Fig. 3.5 and Fig. 3.6, the tendency of the convergence does not change in the smaller values of m of 30 and 50. The calculation becomes stagnant around 1.0E−4 for all the randomly generated initial vectors.

However, the convergence curve shows different nature with larger m of 180 in Fig. 3.7 and 200 in Fig. 3.8, where the calculation progresses without stagnation by gradually reducing the residual. Increased value of m results in faster convergence.

As well as the influence of the value of m, what to be noted in this experiment is that the degree of convergence is affected by the choice of the initial vector. Be reminded that the five curves of different colors in each graph correspond to the five trials using randomly generated initial chromosomes. In case of m=200 in Fig. 3.8, the best initial vector hits the convergence of 1.0E−8 after 14 iterations, whereas the worst one necessitates 24 iterations. This result implies that careful choice of initial vector would play an important role for better GMRES(m) performance. A casual approach of using the 0-vector as the initial guess, as is often observed in the use of iterative matrix solvers, may not always yield good convergence.

* Xabclib homepage, http://www.abc-lib.org/ Xabclib/index.html.

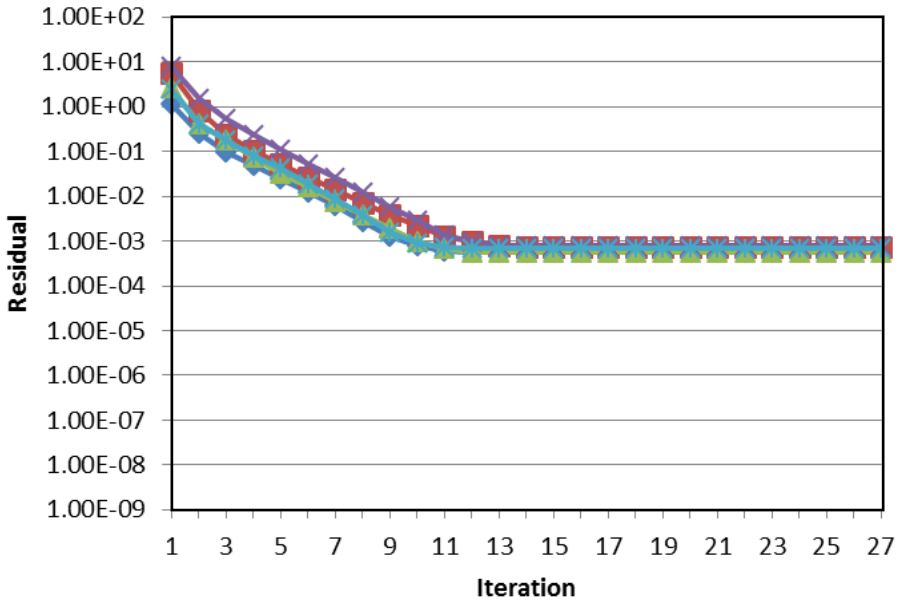


Fig. 3.5 Convergence Curve of GMRES(*m*) Method with Randomly Generated Initial Vectors for *m*=30

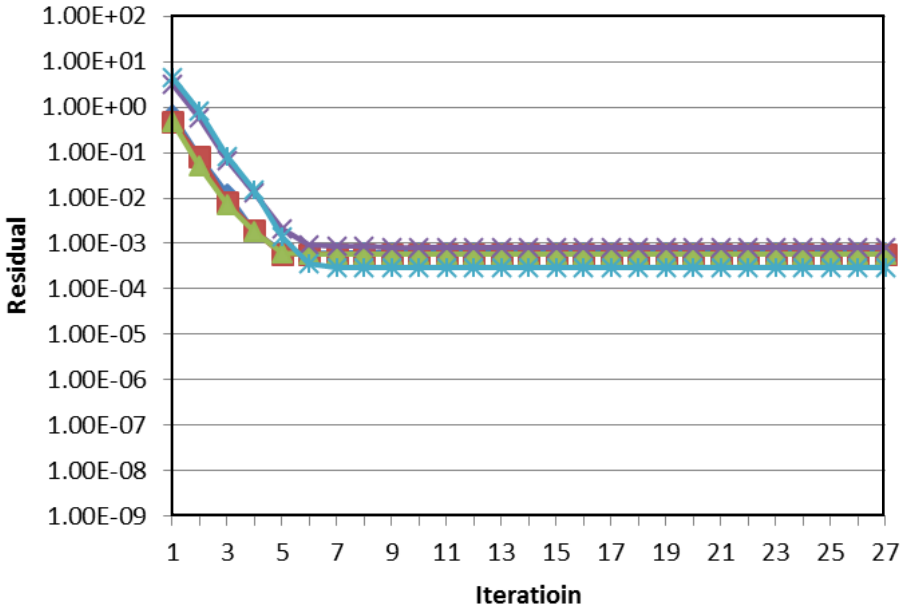


Fig. 3.6 Convergence Curve of GMRES(*m*) Method with Randomly Generated Initial Vectors for *m*=50

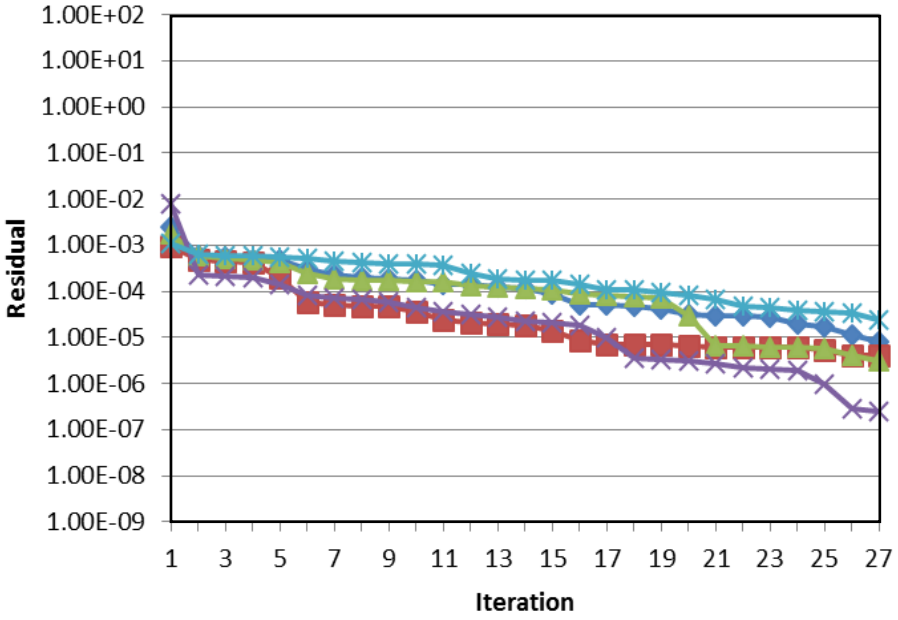


Fig. 3.7 Convergence Curve of GMRES(*m*) Method with Randomly Generated Initial Vectors for *m*=180

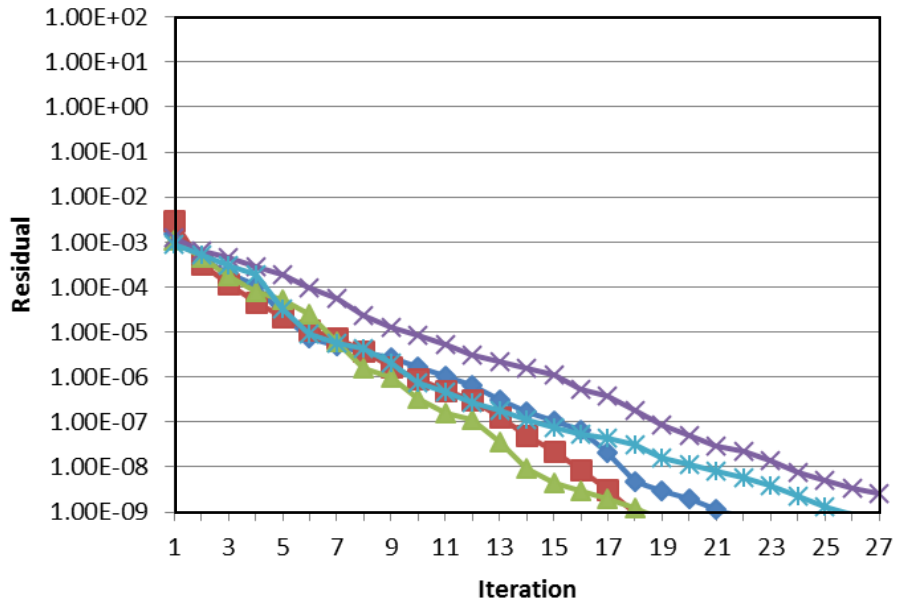


Fig. 3.8 Convergence Curve of GMRES(*m*) Method with Randomly Generated Initial Vectors *m*=200

3.3.4 Crossover Algorithm and Preliminary Experimentation

In order to obtain satisfactory results from the GA process, choice of an appropriate crossover algorithm at the step (h) of Section 3.2.2 plays a crucial role. We have made a preliminary study of three different crossover algorithms by actually solving a matrix using them.

The following four cases were examined:

(A) Simple average: use the mathematical average of the two pairing vectors.
(B) Single point crossover: use the upper and lower halves of the pairing vectors and combine them.
(C) Mutated simple average: use the same simple average algorithm as in (A), when a new kind of mutation is introduced. As the means of the mutation, a few of the newly generated chromosomes are chosen at each crossover process, and a randomly generated disturbance is added to their components. The magnitude of the random disturbance is set to 1/10 of the residual $||b - Ax||$.
(D) Weighted average: taking a cue from the look-back GMRES(m) algorithm, calculate a new vector from the paired vectors, as shown in Step (c)′ of Section 2.2, where $\Delta x^{(l-1)}$ is interpreted as the subtraction of one of the paired vectors from the other at the $(l-1)^{\text{th}}$ restart stage. Note that the weighted average will be identical to the simple average when μ is fixed to 0.5.

In this study, the GA used four chromosomes of randomly generated initial vectors. The chromosomes were numbered from 1 to Nc (i.e., 4) in order of their corresponding residuals (with 1 being the smallest), then (1,2), (1,3), (1,4), and (2,3) were paired to create the chromosomes for the next generation. The manner of chromosome pairing is depicted in Fig. 3.9.

The restart parameter m was set to 50. Special attention should be drawn to the fact that this parameter is quite small considering that successful convergence for torso1 typically occurs when m exceeds 180, even with a good preconditioner.

Fig. 3.10 to Fig. 3.14 show the convergence curves of the numerical experiment using crossover algorithms (A), (B), (C), and (D), respectively. As a reference, a trial with the conventional non-GA calculation was also executed and plotted in Fig. 3.10. The five lines indicated in different colors in each graph show the convergence curves in five trials starting from different sets of randomly generated initial vectors. The convergence curves

in Fig. 3.10 and Fig. 3.11 reveal that both the conventional non-GA case and the simple average case of (A) ended with stagnation, i.e., the residual did not decrease below 1.0E−4. The algorithm with the single-point crossover (B) also did not show significant improvement, as is shown in Fig. 3.12. The convergence curve oscillates upward when a crossover takes place, but the residual remains above 1.0E−4 throughout the calculation.

However, from Fig. 3.13 it can be observed that convergence did occur when mutation was introduced in addition to the simple average algorithm. This is an interesting observation, because introducing disturbance in the crossover process, thereby virtually injecting more diversity in the chromosome pool, appears to have a meaningful impact on the conversion process. An obvious drawback of this algorithm is that the speed of convergence depends on "serendipity" of the random mutation. In the best case, the residual reached 1.0E−8 criteria after 45 iterations, but the worst case required two or more times that number of iterations.

Among the four algorithms examined, the most promising is weighted average algorithm (D), whose convergence curve is shown in Fig. 3.14. This approach demonstrated not only a satisfactory convergence with m = 50, which could not be achieved by using the conventional non-GA algorithms, but also much faster and more stable convergence speed than the mutated average algorithm (C). Convergence to 1.0E−8 can be reached after 14 iterations in the best case, while only 18 iterations were required even in the worst case. It is noteworthy that the convergence curves show similar convergence tendency to each other although the initial guess was started from randomly generated chromosomes.

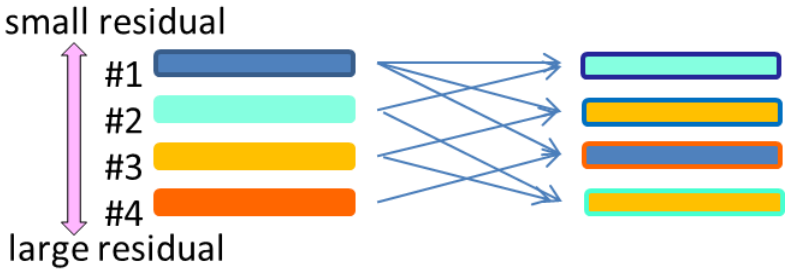


Fig. 3.9 Paring Chromosomes in case with Nc = 4

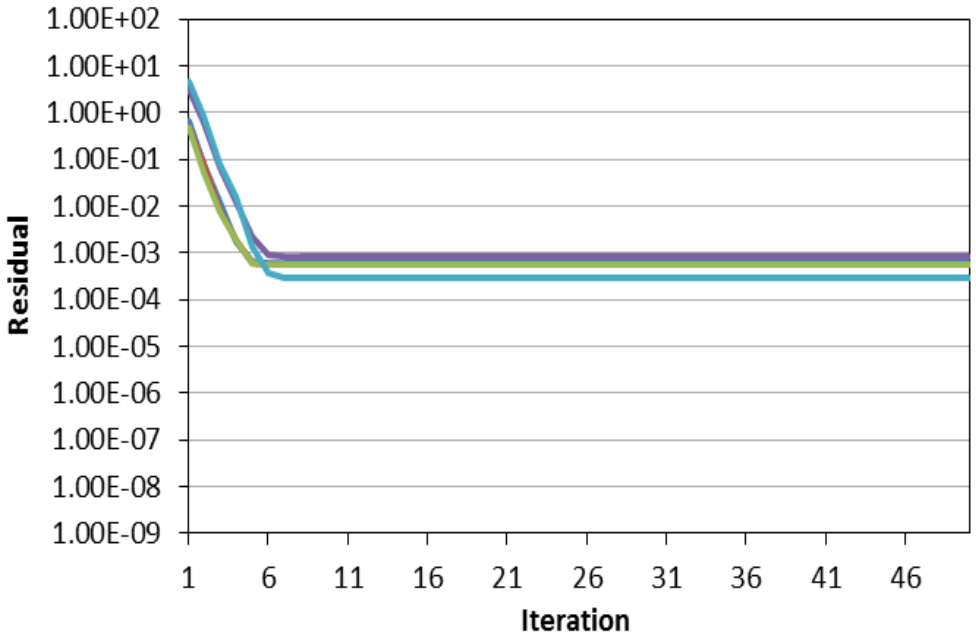


Fig. 3.10 Convergence Curve for non-GA case (for comparison)

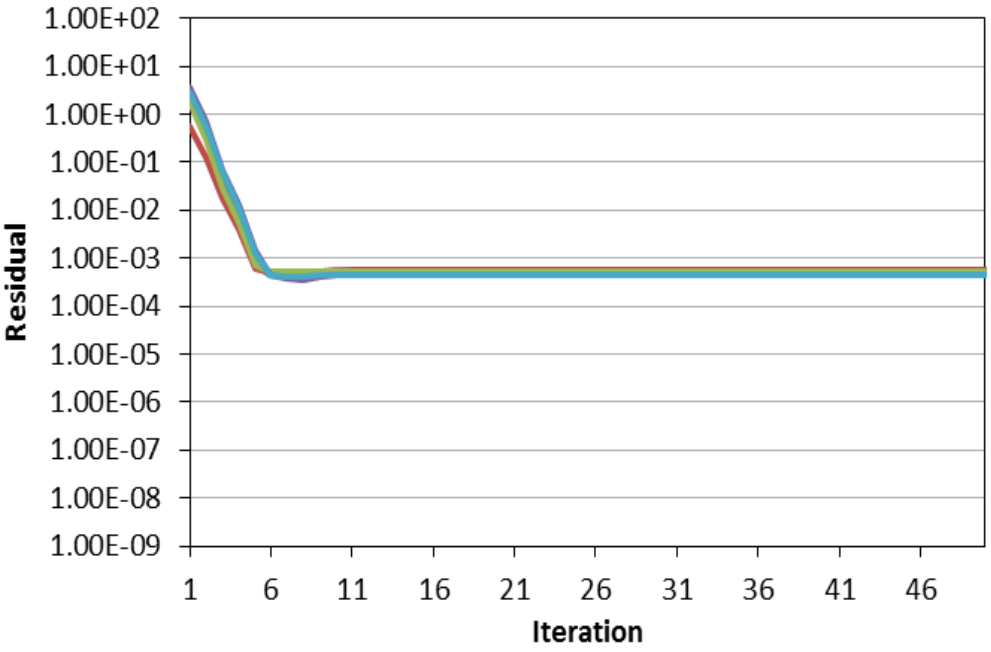


Fig. 3.11 Convergence Curve for Simple Average Crossover Algorithm

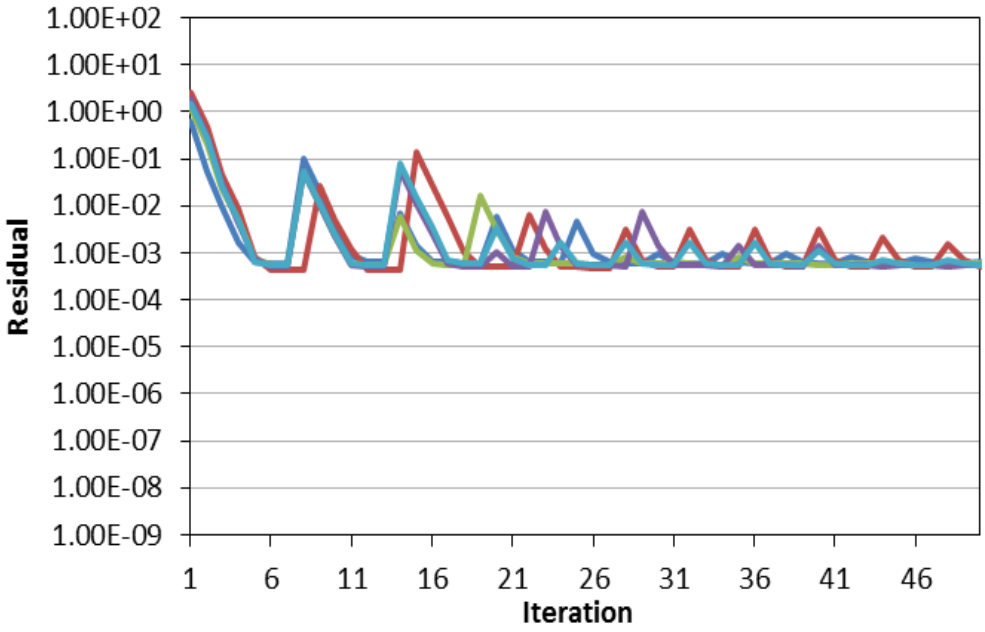


Fig. 3.12 Convergence Curve for Single Point Crossover Algorithm

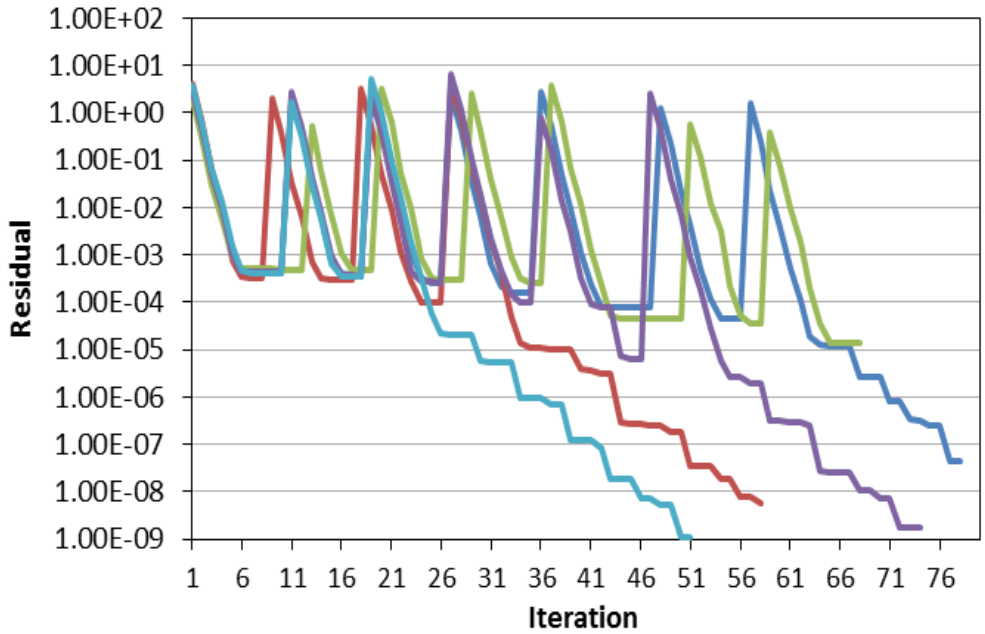


Fig. 3.13 Convergence Curve for Mutated Simple Average Crossover Algorithm

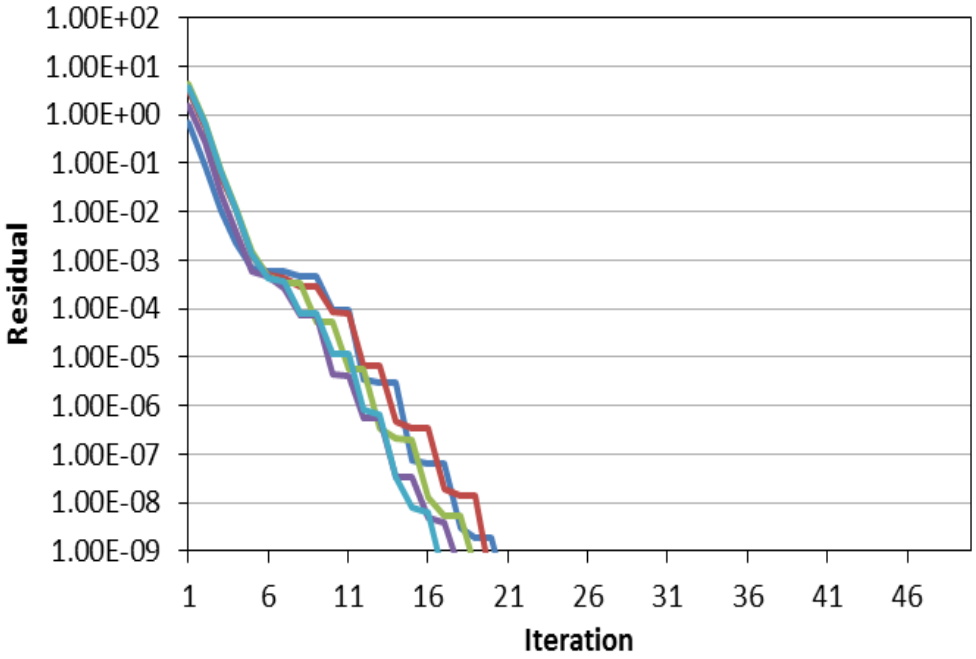


Fig. 3.14 Convergence Curve for Weighted Average Crossover Algorithm

3.4 Evaluation

By taking four exemplary matrices from University of Florida sparse matrix collection, we carried out more diversified numerical experiment of the proposed GA based GMRES(*m*) method with the weighted average crossover algorithm.

3.4.1 Experiment Settings

The matrices selected are torso1, sme3Da, xenon1, and comsol. The configuration of the experiment is the same as the one in Section 3.3.3. The number of chromosomes *Nc* is set to be 1, 4 and 8. Three different restart parameters *m* are selected for each matrix. The iterations in every GA generation are executed until stagnation that is defined as the decrease of the residual becoming less than 3% of the previous iteration. After all chromosomes have stagnated, the crossover algorithm is activated to create the chromosomes for the next generation.

To determine the crossover pair, the chromosomes are numbered from 1 to *Nc* in the order of the corresponding residual (1 smallest). For *Nc*=4, (1,2), (1,3), (1,4), and (2,3)

are paired (as in Fig. 3.9), while (1,2), (1,3), (1,4), (1,5), (2,3), (2,4), (2,5), and (3,4) are paired for *Nc*=8. For each combination of *m* and *Nc*, five trials are made using randomly generated initial chromosomes. A trial is regarded as converged when the convergence criterion $||A \cdot x - b||/||b|| < 1.0\text{E}{-8}$ is met. When the criterion is not satisfied after 500 iterations, the execution is deemed to have failed to converge.

3.4.2 Results

The results from the experiments are shown in Table 3.1 and Table 3.2. The number in the "Iteration" column is the total restarting iteration count needed to obtain a converged solution, i.e., the normalized residual became less than 1.0E−8. This number corresponds to a multiplication of iterations in steps (b) and (d) in the algorithm shown in Section 3.3.2. The displayed number is the rounded average of the five trials that started from randomly generated initial vectors. In this column, "NCV" stands for "no convergence". The "e-time" column shows the execution time in seconds (I/O excluded) for each trial until convergence. The execution time corresponding to the best performance for each matrix is indicated with underlined boldface.

By comparing the Iteration and e-time columns for $m = 1$ with those for $m = 4$ and $m = 8$ for the same matrix, the table shows that applying GA is effective in decreasing the iteration count and thus reducing the execution time. Recall that the result for $m = 1$ corresponds to the result for the conventional non-GA execution. However, the impact of applying GA is affected by the nature of the matrix; different characteristics in the convergence pattern occur when values of *m* were changed. There were two types of matrices.

The performance results for torso1 and sme3Da are shown in Table 3.1. These are the matrices in which convergence occurs at a much smaller *m* than the non-GA case. With GA, the execution time becomes six times shorter in torso1 (2.0E+1 vs 1.2E+2), and less than half in sme3Da (6.1E+1 vs 1.4E+2). This tendency is desirable since it can not only contribute to shortening the execution time but also reducing memory use because of smaller *m*. As an example of the obtained results from this type of matrices, the convergence curves of sme3Da for $Nc = 1$ and $Nc = 8$ at $m = 50$ are shown in Fig. 3.15 and Fig. 3.16. It can be observed that although the calculation fails to converge without GA ($Nc = 1$, Fig. 3.15), it results in conversion when GA is applied ($Nc = 8$, Fig. 3.16).

Table 3.2 shows the performance results for the other group of the matrices; xenon1 and comsol, in which there is little improvement in convergence with changes in the value of *m*, but still shows better performance because the iteration count decreased for the same *m*. The improvement in performance was more than two times in xenon1 at *m* = 300 (2.2E+2 vs 5.1E+2) and comsol at *m* = 60 (1.8E0 vs 3.9E0). The convergence curves of xenon1 for *Nc*=1 and *Nc*=4 at *m*=300 are shown in Fig. 3.17 and Fig. 3.18. Although both cases of *Nc*=1 and *Nc*=4 result in convergence, much faster reduction of the residual is observed when GA has been applied (*Nc*=4, Fig. 3.18).

In contrast, the impact of increasing the number of chromosomes is not so clear in either of the groups. Iteration counts for *Nc* = 4 and *Nc* = 8 do not present much difference for most of the matrices, even when the value of *m* was changed.

Table 3.1 Impact of Applying GA to GMRES(*m*) for torso1 and sme3Da

Matrix	*m*	*Nc*	Iteration	e-time
torso1 (n=116,158) (no of non zero elements; 8.5M)	30	1	NCV	-
		4	30	2.5E+1
		8	27	2.3E+1
	50	1	NCV	-
		4	17	**2.0E+1**
		8	19	2.3E+1
	200	1	19	1.2E+2
		4	14	8.7E+1
		8	13	8.1E+1
sme3Da (n=12,504) (no of non zero elements;0.8M)	30	1	NCV	-
		4	110	1.0E+1
		8	65	6.2E0
	50	1	NCV	-
		4	58	9.8E0
		8	36	**6.1E0**
	200	1	18	1.4E+1
		4	13	9.9E0
		8	12	9.2E0

Table 3.2 Impact of Applying GA to GMRES(*m*) for xenon1 and comsol

Matrix	*m*	*Nc*	Iteration	e-time
xenon1 (n=48,600) (no of non zero elements;1.2M)	200	1	NCV	-
		4	254	4.1E+2
		8	219	3.5E+2
	250	1	326	7.7E+2
		4	153	3.6E+2
		8	153	3.6E+2
	300	1	159	5.1E+2
		4	69	**2.2E+2**
		8	72	2.3E+2
Comsol (n=1,500) (no of non zero elements;0.1M)	10	1	565	1.5E0
		4	63	1.6E−1
		8	68	1.8E−1
	30	1	63	5.2E−1
		4	32	2.6E−1
		8	29	2.4E−1
	60	1	22	3.9E−1
		4	10	**1.8E−1**
		8	10	1.9E−1

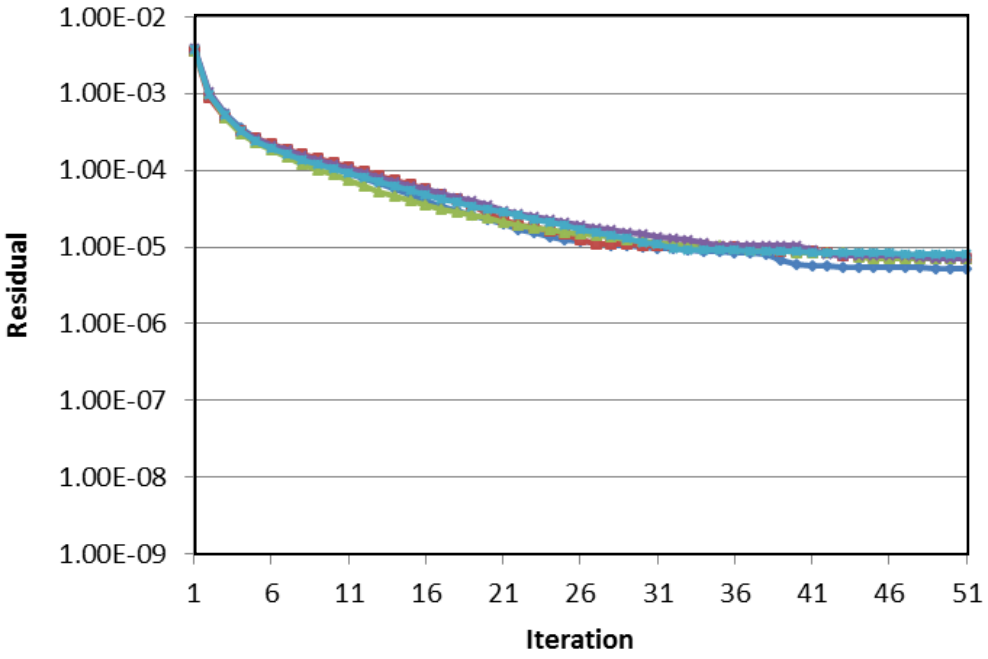


Fig. 3.15 Convergence Curve of sme3Da at m = 50, Nc = 1 (non-GA)

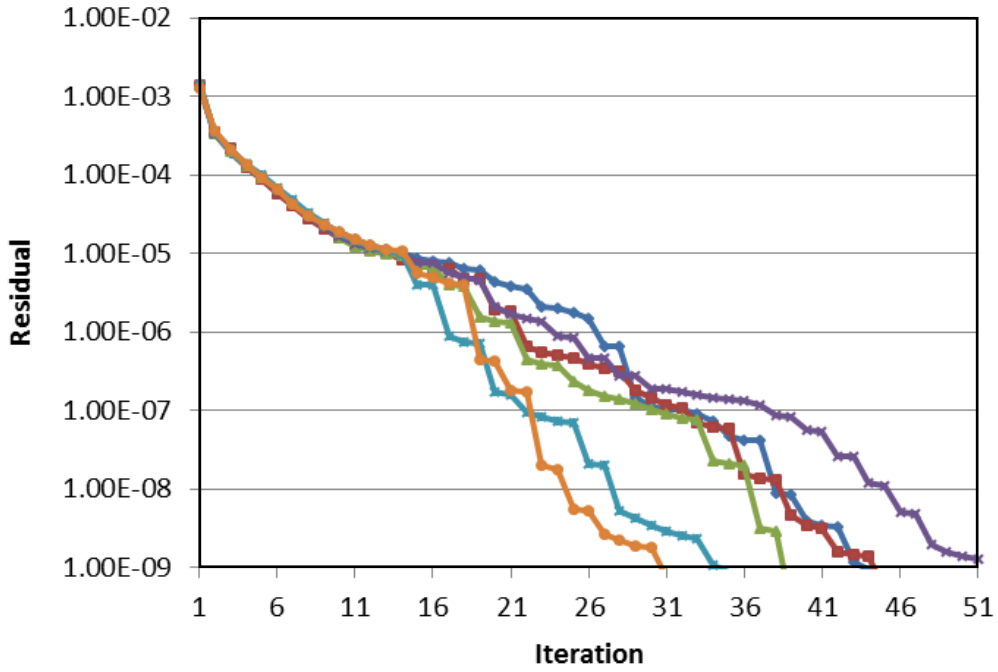


Fig. 3.16 Convergence Curve of sme3Da at m = 50, Nc =8

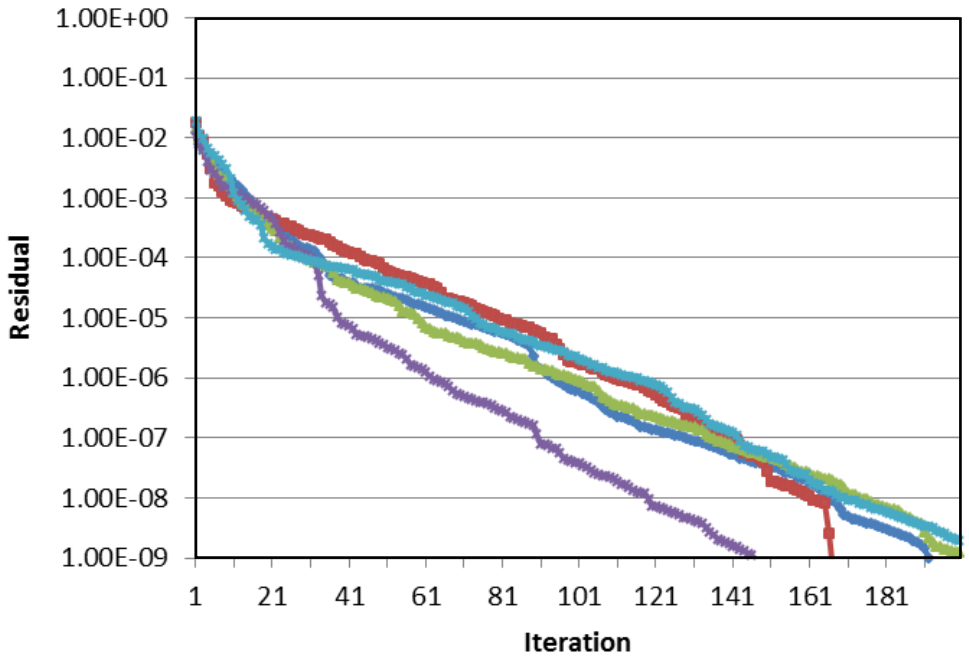


Fig. 3.17 Convergence Curve of xenon1 at m = 300, Nc = 1 (non-GA)

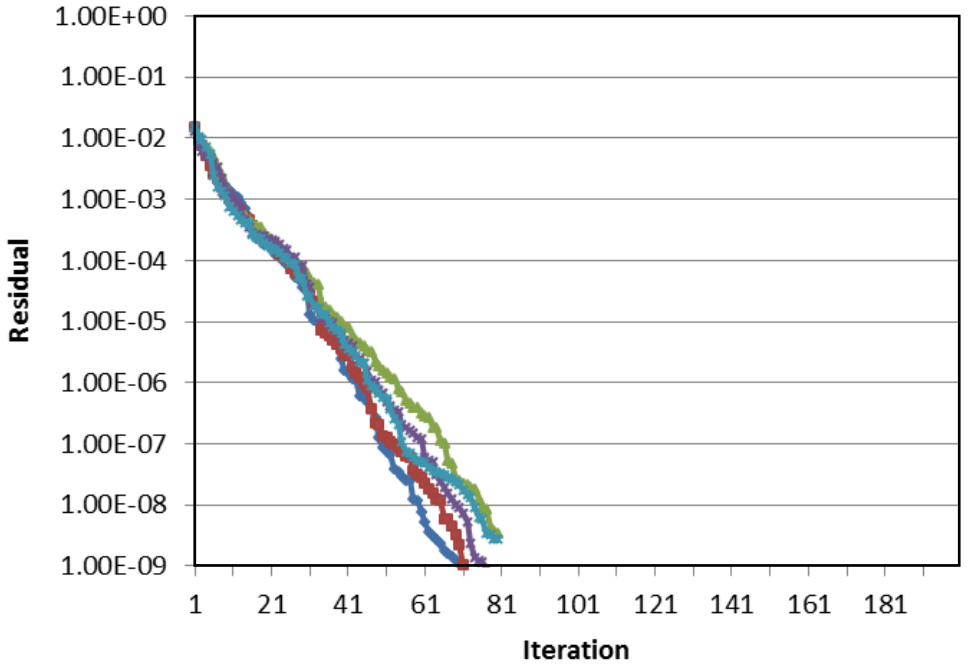


Fig. 3.18 Convergence Curve of xenon1 at m = 300, Nc =4

3.4.3 Combination of GA–GMRES(*m*) with Look-back Method

One of the virtues of the proposed GA–GMRES(*m*) method is that it is orthogonal to most of the existing acceleration techniques which are designed for the use within one thread (i.e., one chromosome), such as the Look-back types of GMRES(*m*) method [70]-[72], whereby allowing our GA based approach to be coupled with those techniques to expect synergetic effects.

In order to demonstrate the above orthogonal nature, we carried out additional numerical experiments in which GA–GMRES(*m*) and a Look-back algorithm were coupled. For each crossover process, the modifications of the intermediate initial vectors were doubly applied; i.e., two intermediate results in the current and previous stages were referred back to in order to improve the next initial vector within each chromosome (as in the Look-back algorithm shown in Section 3.2.2 (c)'), then the pairs of the improved initial vectors across chromosomes were further utilized to refine the vectors using the weighted average algorithm.

The experimental results are shown in Table 3.3. The number in each cell in the table is the restart counts (five times average) till convergence. The columns of "non-GA" and "non-GA+Look-back" show the results for the conventional GMRES(*m*) method with and without Look-back modification. Likewise, the columns "GA" and "GA+Look-back" are the results when GA is applied, with and without Look-back modification. The results show that the effect of the GA application and Look-back modification are orthogonal as expected, thus resultant "GA+Look-back" gives rise to the convoluted performance of the two algorithms. The acceleration ratio of Look-back modification is inserted in the "(a)/(b)" and "(c)/(d)" columns of Table 3.3. Comparison of these two columns shows that the combination of GA and Look-back modification is even more effective than the introduction of the same technique in the conventional case. Out of the tested four matrices, torso1 showed slightly different behavior where the impact of the Look-back modification was not so visible, and the major contribution to the performance improvement appeared to derive from GA application only. The convergence curves for the same experiment with sme3Da matrix are plotted on Fig. 3.19 that corresponds to the second row of Table 3.3. It is visually understandable by comparing the four curves that the combination of GA–GMRES(*m*) and Look-back method indeed contributes to betterment of the convergence performance.

Table 3.3 Combination of GA–GMRES(*m*) with Look-back Method

Stagnation parameter	non -GA (a)	non-GA + Look-back (b)	Ratio (a)/(b)	GA (c)	GA + Look-back (d)	Ratio (c)/(d)
torso1 (*m* = 30)	NCV	NCV	-	30	29	1.0
sme3Da (*m* = 50)	159	120	1.3	58	36	1.6
xenon1 (*m* = 300)	159	93	1.7	69	37	2.0
comsol (*m* = 30)	63	40	1.6	32	18	1.8

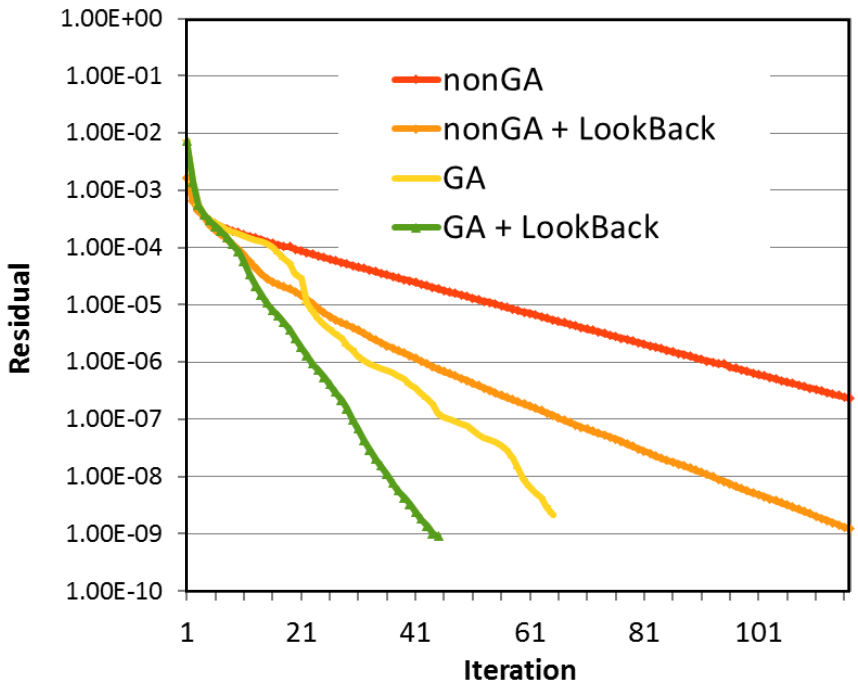


Fig. 3.19 Convergence Curves for GA–GMRES(*m*) with Look-back Method

3.5 Conclusion

An approach to improve the performance of GMRES(m) method by applying GA to its restart process was examined and presented. A number of randomly generated initial vectors were used as the initial chromosomes, and intermediate initial vectors were regarded as succeeding chromosomes. When stagnation was detected across all chromosomes in the GMRES(m) restart process, the GA process performed crossover or the chromosomes for the next restart. A weighted average algorithm, inspired by the Look-back GMRES(m) method, was developed to effectively perform the crossover process. Using numerical experiments, we found that the weighted average algorithm successfully converged, while the single-point crossover and simple average methods did not. The proposed method was applied to four example matrices in the University of Florida sparse matrix collection. We successfully demonstrated that the GA − GMRES(m) shortened the execution time to $1/2 - 1/6$ for all four matrices against the execution time with the conventional non-GA algorithm.

Chapter 4

Automatic Subspace Parameter Optimization for GA–GMRES(*m*) Method

4.1 Introduction

In this chapter, we propose an improvement of GA‑GMRES(*m*) method described in Chapter 3. It is intended to enhance the convergence performance and ease of use of the newly developed GA–GMRES(*m*) [49][50].

Despite the advantage of GA–GMRES(*m*) method that effectively enhances the performance of conventional matrix solver, we noticed there are still two challenges remaining to be addressed. The first is that the proposed method does not necessarily let the crossover process intervene at the right timing, since the stagnation detection threshold is fixed. The second is that if *m* chosen as a starting point is too small or too large, it sometimes leads to a slowdown or failure in convergence, or it may be using unnecessary large memory space or computational time than actually required, both resulting in unsatisfactory performance. In this study, we attempt to address these drawbacks of the proposed GA–GMRES(*m*) method.

Most of the conventional methods of choosing *m* automatically relies on a simple approach of measure the decrease in residual $r = ||b - A \cdot x||$ on the way of calculation, and increase or decrease the value of *m* when the reduction of *r* is too small (convergence is stagnating) or too large (could converge with smaller *m*) [73][74]. However, this approach often fails to exhibit the expected effect. The decrease of the residual is not necessarily monotone or stable, which makes it difficult to determine the right timing

when the *m* increase/decrease process should intervene.

Our approach is to make the adjustment of *m* automatic and flexible so that its value can be increased or decreased according to the status of the overall convergence. In order to enable this kind of flexible adjustment, we take advantage of GA–GMRES(*m*) method that there are multiple threads of convergence process on-going. By comparing the convergence behavior across the chromosomes, the stagnation point of the convergence process can be found more precisely on-the-fly than the conventional method, which enables more confident control of the value of *m* for even more efficiency and ease of use.

The rest of this chapter is organized as follows. Section 4.2 we evaluate the influence of convergence threshold, introduce the concept of "chromosome-wide stagnation", and propose an automatic *m* adjustment technique when this type of stagnation takes place. Section 4.3 evaluates the proposed *m* adjustment technique using several example matrices and compares its effectiveness against the conventional *m* adjustment technique that is based on local stagnation. Finally we conclude this chapter in Section 4.4.

4.2 Optimization of the Stagnation Threshold Parameter and the Krylov Sub-space Parameter *m*

In the GA–GMRES(*m*) method described in Chapter 3, the convergence threshold parameter was fixed at a certain value, viz., the calculation was deemed stagnated if the residual decrease became less than 3% of the previous iteration at each generation. A fixed Krylov subspace value for *m* was also used throughout the calculation. However, a closer observation of the convergence curves reveals that the convergence trend differs depending largely on the type of matrices to be solved. In the sme3Da case as shown in Fig. 3.16, the convergence curve flattens at an early stage, while in the xenon1 case as shown in Fig. 3.18, it tends to maintain a degree of steepness. This suggests that adjusting the occurrence of the GA intervention and GMRES(*m*) subspace *m* to a more appropriate timing and value, respectively, may help GMRES(*m*) process to result in better convergence.

4.2.1 Influence of the Stagnation Threshold Parameter

We conducted a numerical experiment to understand the influence of the stagnation

threshold parameter on convergence. The configuration of the numerical experiment is the same as described in Section 3.3.3. The RHS vector b of $Ax = b$ was determined such that x is an n-dimensional vector of (1.0 ... 1.0) and b is the product of A and x. The components of the randomly generated initial vectors were evenly spread between −1.0E+1 and 1.0E+1. For each trial, five randomly generated initial vectors were prepared and GMRES(m) calculations were performed.

The test matrices were torso1, sme3Da, xenon1, and comsol. The stagnation threshold parameter was scanned from 0.01 to 1.0. Here the threshold 0.01 means that the GMRES restarting process shown in Section 3.3.2 step (d) is deemed stagnant when the difference in residual between the current and previous iteration becomes 1% (=0.01) of the current residual.

The experiment results are shown in Table 4.1. The figures in the table are the five-times average of the number of restart stages included in each calculation, which is proportional to the requested computational time for each test matrix. NCV stands for "no convergence". From Table 4.1 it can be seen that although the stagnation threshold parameter presents a certain influence over the convergence speed; the degree of influence is not very sensitive to the value of the parameter. The iteration count until the final convergence stays relatively stable if the threshold parameter is varied between 0.03 and 1.0. This result suggests that paying the extra cost involved to search for the optimum threshold parameter is unlikely to be rewarded, and it is sufficient that we specify the threshold value between 0.1 and 1.0.

Table 4.1 Influence of the Stagnation Parameter

Stagnation parameter	non -GA	0.01	0.03	0.1	1.0
torso1 (m = 30)	NCV	50	30	30	30
sme3Da (m = 50)	157	129	48	46	48
xenon1 (m = 300)	159	80	75	69	70
comsol (m = 30)	63	75	32	27	27

4.2.2 Chromosome-wide Stagnation

Thus far, stagnation in the calculation process has been described as the status wherein the decrease of the residual becomes less than the prefixed value for each chromosome at each stage of the GA generation. Hereafter, this status is defined as “local stagnation”. As mentioned previously, when local stagnation is detected across all the chromosomes, the crossover process is initiated to resolve the locally stagnated status. However, there are some cases in the convergence process where the residual of the GMRES(m) does not show much progress despite the fact that frequent GA intervention is taking place. This status is defined as “chromosome-wide stagnation.”

Fig. 4.1 shows a typical convergence curve for sme3Da at m = 50. A cross marker on the curve corresponds to an intervention of crossover process due to local stagnation. We can observe several areas in the convergence curve where the convergence does not present much progress despite the heavy crossing intervention. These areas correspond to the aforementioned chromosome-wide stagnation. In the figure, three stagnated areas can be observed, and they are identified as such on the graph.

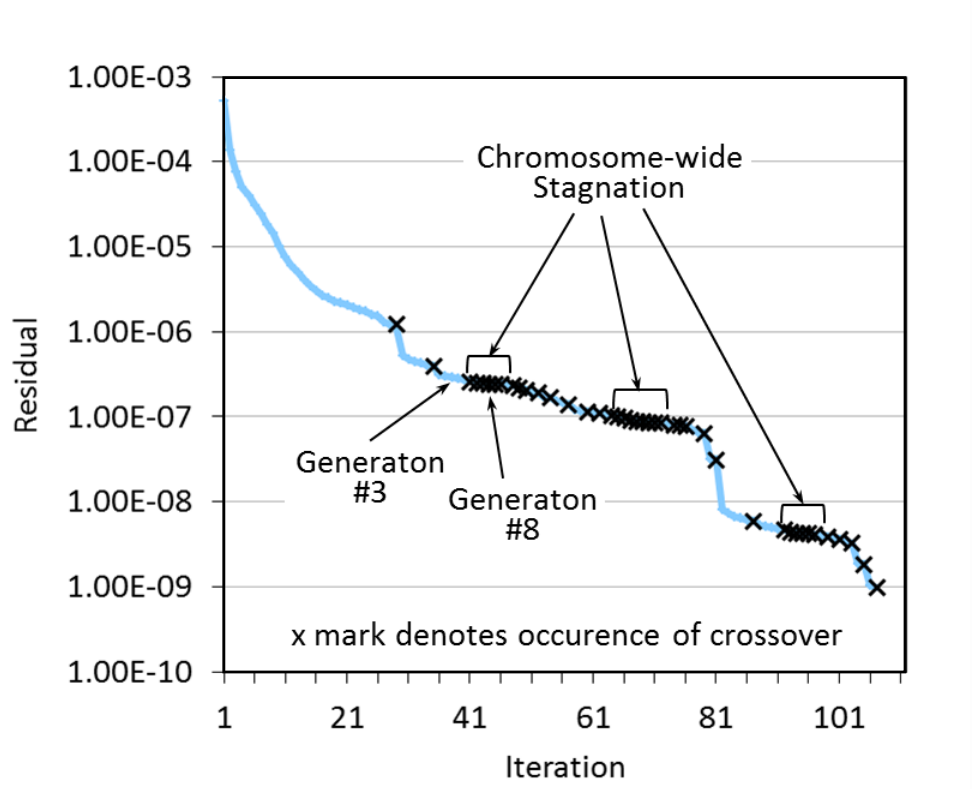


Fig. 4.1 Chromosome-wide Stagnation Appearing on Convergence Curve

4.2.3 Stagnation Detection

To make decisions about timing and the best way for the appropriate crossover process to occur, it is desirable to first detect whether the calculation has fallen into a state of chromosome-wide stagnation. One method is to compare the value of the residual between the current stage and previous stage in one chromosome and then compare the difference in residual for all the chromosomes. However, there is a simpler and less costly means for detecting chromosome-wide stagnation. When all the chromosomes have come to a state of stagnation and the decrease in the residual is stuck after the crossover process, the value of the residual itself (rather than the difference from the previous stage) becomes almost uniform across all the chromosomes. This implies that the nature of all the chromosomes becomes similar, resulting in a less effective crossing over of the chromosomes. Table 4.2 and Table 4.3 show the development of the residual in the numerical experiment, as with the one plotted in Fig. 4.1. Table 4.2 is the residual convergence sequence when no chromosome-wide stagnation is observed (generation #3). We can see that the pattern of the residual reduction is unique for each chromosome. Table 4.3 is the same sequence when chromosome-wide stagnation is present (generation #8). This time we see that the residuals of all chromosomes become almost identical, implying that the characteristics of the chromosomes resemble each other.

Table 4.2 Residual Convergence Progress without Chromosome-wide Stagnation

```
------------------------------
 Generation:      #3
------------------------------
 Chromosome:             No.1
     0.2689048E-06
     0.2638042E-06
 Chromosome:             No.2
     0.3029526E-06
     0.2902925E-06
     0.2810679E-06
     0.2693219E-06
     0.2623717E-06
 Chromosome:             No.3
     0.2959425E-06
     0.2900377E-06
 Chromosome:             No.4
     0.7687083E-06
     0.6848289E-06
     0.6389537E-06
     0.6380865E-06
------------------------------
```

Table 4.3 Residual Convergence Progress with Chromosome-wide Stagnation

```
--------------------------------
 Generation:      #8
--------------------------------
 Chromosome:             No.1
     0.2461232E-06
     0.2455979E-06
 Chromosome:             No.2
     0.2461134E-06
     0.2455717E-06
 Chromosome:             No.3
     0.2460692E-06
     0.2456790E-06
 Chromosome:             No.4
     0.2462488E-06
     0.2457450E-06
--------------------------------
```

4.3 Evaluation

4.3.1 Numerical Experiments with Example Matrices

Taking measures to minimize the impact of chromosome-wide stagnation is believed to be effective in further accelerating the convergence. It allows the use of non-costly calculation when the convergence is occurring, intervening with the more expensive calculation only when the calculation experiences chromosome-wide stagnation. To fulfill this objective, an approach of automatically adjusting the subspace value *m* is examined; i.e., *m* is increased when chromosome-wide stagnation is encountered, and gets reverted to the original when the stagnation has been resolved.

We conducted yet another numerical experiment to investigate the efficiency of this approach. The configuration of the experiment was the same as that specified in Section 3.3.3, with sme3Da, xenon1, and comsol1 chosen as the testing matrices. This time torso1 was excluded from the test matrices because GA–GMRES(*m*) already performed so well on this matrix that there was no chromosome-wide stagnation taking place. The subspace parameter *m* was set to 30 (sme3De), 10 (comsol1), and 200 (xenon1), and then increased to 60, 20, and 300, respectively, when chromosome-wide stagnation was detected. The local stagnation threshold was set to 1.0 for all the testing matrices as learned appropriate through the study in Section 4.2.1.

Fig. 4.2 to Fig. 4.4 show the convergence curves obtained from the sme3Da matrix

experiment. The five lines in each graph show the convergence curves in five trials starting from different sets of randomly generated initial vectors. Fig. 4.2 and Fig. 4.4 correspond to the cases in which *m* is fixed to 30 and 60, respectively, and Fig. 4.3 in the middle is the result of dynamic intervention of *m* as it increased from 30 to 60, while chromosome-wide stagnation occurred. The black marks on the graph indicate that the *m* value increased at those points. It should be noted that the *m* increased only when chromosome-wide stagnation was recognized, thus enabling a smaller *m* to be used whenever convergence progressed without stagnation. It is also evident from a comparison of the three graphs that an increasing *m* contributes to a tendency for accelerated convergence. The steepness of the curve for the dynamic intervention case is much closer to that of the case with *m* fixed to 60.

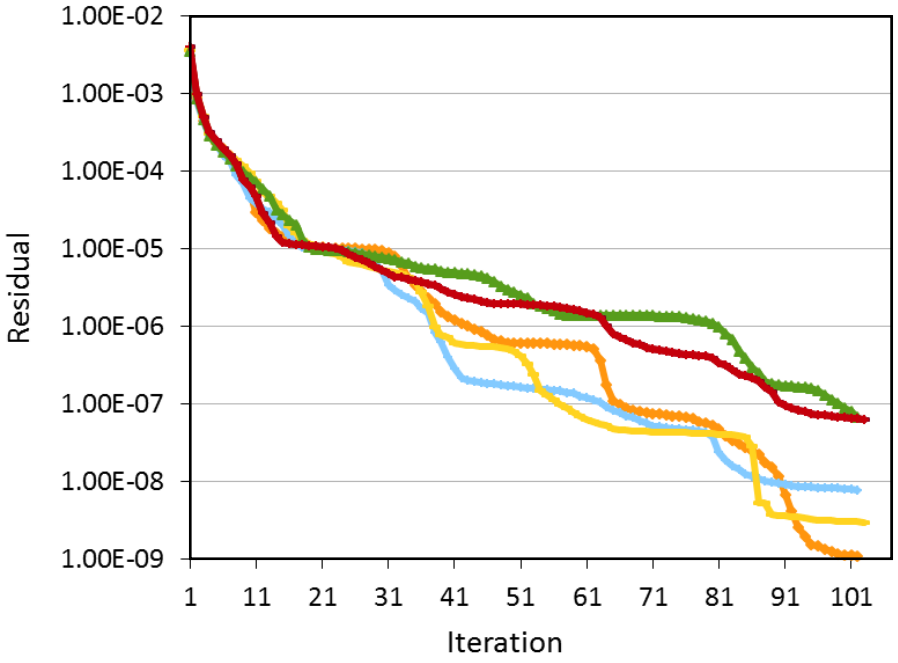


Fig. 4.2 Convergence Curve for sme3Da at *m* = 30

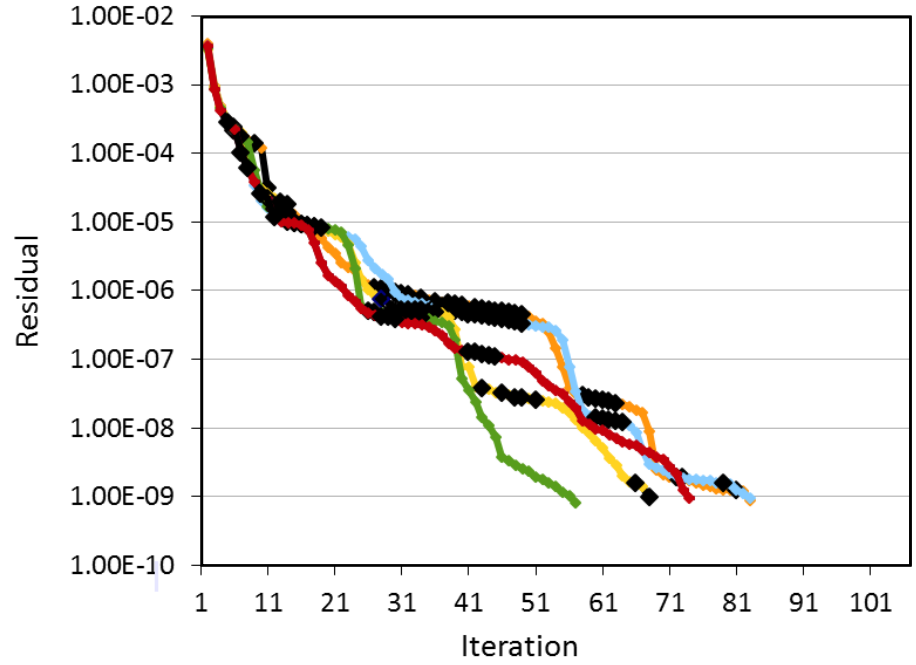


Fig. 4.3 Convergence Curve for sme3Da at m = 30–60

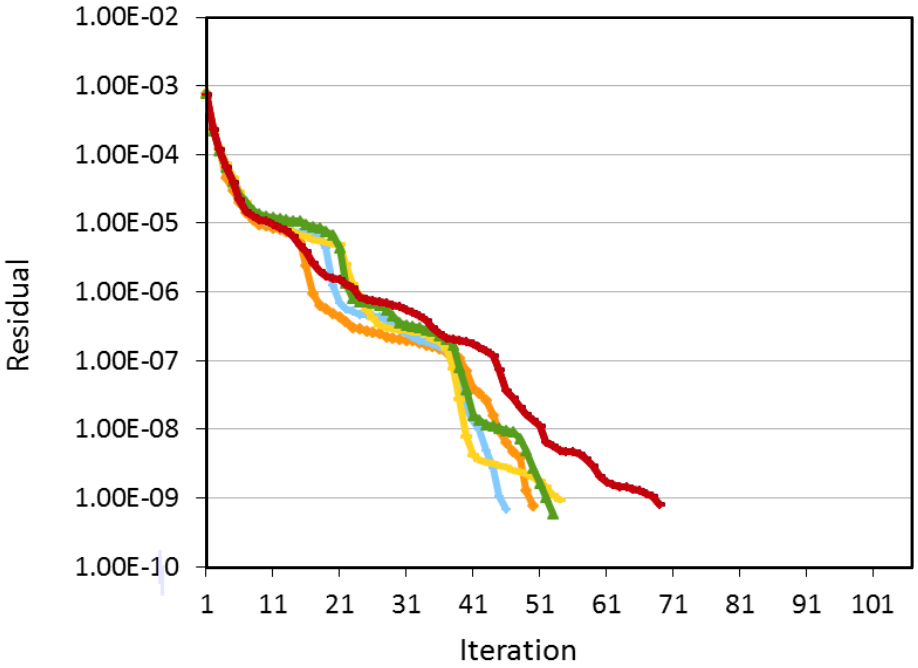


Fig. 4.4 Convergence Curve for sme3Da at m = 60

Table 4.4 to Table 4.6 show the performance results of the numerical experiment using the above approach for all the three tested matrices. In the sme3Da case, as discussed above (Table 4.4, columns 30–60 and 60–120), the value of *m* starts from 30 or 60 and is doubled to 60 or 120, while chromosome-wide stagnation is observed, and it is then pushed back to 30 or 60, when the stagnation is resolved. The results for fixed *m* values (30, 60, and 120) are provided in the same table. From these results we can understand that increasing the value of *m* with appropriate timing is effective. Column 30–120 is the case in which *m* started from 30 and gradually increased to 60, 90, and 120, while chromosome-wide stagnation persisted. This treatment showed the shortest execution time of all, outperforming the fixed *m* = 120 case by 30%. Likewise, the *m* adjustment strategy works effectively for the comsol1 matrix, increasing its performance when compared with the results for fixed values of *m* (10, 20, and 40). In this case also, gradual incrementing of *m* from 10 to 20, 30, and 40 (columns 10–40) exhibited the best performance, outperforming the fixed *m* = 40 case by 30%.

Table 4.4 Convergence performance for sme3Da

m	30	30–60	60	60–120	120	30–120
e-time (sec)	1.0E+2	8.1E+1	9.8E+1	6.7E+1	7.6E+1	5.8E+1
Number of restart stages	101	58	45	23	19	30

Table 4.5 Convergence performance for comsol

m	10	10–20	20	20–40	40	10–40
e-time (sec)	1.6E0	1.3E0	1.4E0	1.2E0	1.3E0	1.0E0
Number of restart stages	92	51	42	33	20	36

Table 4.6 Convergence performance for xenon1

m	200	200–300	300
e-time (sec)	4.0E+1	2.2E+2	1.8E+2
Number of restart stages	254	123	70

In contrast, the effect of increasing m at chromosome-wide stagnation is not as clear in the xenon1 case (Table 4.6) because the dynamic variation of m within 200–300 yields only an intermediate result compared with the results for m = 200 and 300. Even in this case however, it should be noted that adoption of the proposed technique incurred no harm because the performance fell in between the case with m = 200 and 300, and there was no adversary effect observed.

4.3.2 Comparison with Conventional Stagnation Detection Technique

To further clarify the advantage of introducing the concept of the chromosome-wide stagnation, we conducted an experimental comparison of the proposed approach against the conventional m adjustment based on the local stagnation [73]. The conventional method calculates the ratio between the residuals of the previous and current restart stages i.e.;

$$p^{(l)} = ||b^{(l)} - A x^{(l)}|| \,/\, ||b^{(l-1)} - A x^{(l-1)}|| \tag{4.1}$$

and recognizes stagnation if $p^{(l)}$ becomes close to 1.0 within a given threshold.

In Table 4.7, the comparison on the restart iteration counts and execution time obtained from the conventional local stagnation detection and the chromosome-wide detection is shown. The same test matrices were used as in the previous section. We also selected the same value range and incremental step of m for both of the detection methods for each test matrix.

Although we adopted the recommended threshold value to detect the local stagnation, the conventional stagnation detection looking at only in-chromosome convergence appeared to have recognized the stagnation too sensitively because it tended to pick up the local fluctuations in the convergence curve. The “Adjustment Ratio” row of the table reveals that the conventional m adjustment is activated around 80% of the restart stages. Although it contributed to lessen the number of the restart stages, the calculation itself ended up with longer execution time due to too frequent intervention. On the contrary, in our approach of detecting the chromosome-wide stagnation, the intervention of increased m took place more moderate and timely thus it succeeded in giving better performance by choosing more appropriate value of m in accordance with the progress of calculation. The new approach consistently outperforms

the conventional one for all the tested matrices, i.e., 58% for sme3Da, 18% for xenon1, and 30% for comsol.

Table 4.7 Comparison of Adjusting *m* on Local and Chromosome-wide Stagnation

matrix	sme3Da		xenon1		comsol	
m	30−120		200−300		10−40	
Stagnation Detection	local	c-wide	local	c-wide	local	c-wide
e-time (sec)	9.2E+1	5.8E+1	2.6E+2	2.2E+2	1.3E0	1.0E0
Number of restart stages (a)	30	30	115	123	27	36
Number of *m* adjustment (b)	26	14	102	32	20	12
Adjustment Ratio (b)/(a)	0.87	0.47	0.89	0.26	0.74	0.33

local: local stagnation (conventional), c-wide: chromosome wide stagnation

4.4 Conclusion

In this chapter, we presented an approach for further improving the efficiency of solving linear systems by the GA–GMRES(*m*) method while alleviating the end-user from the effort of setting up an appropriate *m* via try and error process. We paid special attention to the fact that the residuals for all the chromosomes became almost identical when overall stagnation was taking place. This implies that the nature of the chromosomes becomes similar thus making the crossover process less effective. We named this phenomenon as “chromosome-wide stagnation” that enabled simple on-the-fly detection of a slowdown in convergence of the GA process. In order to enhance the convergence performance, a technique to adjust the *m* value automatically at the onset of such stagnation was proposed, which contributed a further 30% improvement in the convergence speed of GA–GMRES(*m*) method. The proposed technique also consistently exhibited better convergence performance than the conventional *m* adjustment technique based on local stagnation detection.

Chapter 5

Conclusions

5.1 Concluding Remarks

In this dissertation, we considered the following two proposals pertaining to effective execution of numerical simulation of physical phenomena, i.e., matrix formation and matrix solution. The first is realization of a high-level language and translation system dedicated to the matrix formation phase of the numerical simulation. The second is the introduction of a GA to the matrix solution phase to exploit the abundant resources available in contemporary computational environments.

In Chapter 1, we described the general process of numerical simulation and provided an overview of existing methods and techniques to deal with the process.

In Chapter 2 we proposed a new language and translation system, the Differential EQuation SOLver (DEQSOL) system, dedicated to forming matrices that correspond to numerical simulation described by PDEs. First, we proposed the design principle and specifications of the DEQSOL language, which allows the end user to describe a physical problem at a PDE level to alleviate the burden of writing lengthy matrix formation programs. We then explained the DEQSOL translator architecture, which converts the PDE level description into FORTRAN code to form a coefficient matrix that exploits the computational power of a supercomputer. Finally, we evaluated the proposed matrix formation system in terms of description coverage, productivity enhancement, and execution performance. The evaluation demonstrated that the end user can describe the physical problem in one-tenth of the FORTRAN code lines while gaining more than a 90% (typically 95%) vectorization ratio on a supercomputer.

Chapter 3 and Chapter 4 addressed the second issue, effective solution of the formed matrices. The GMRES(*m*) method was introduced as the basis of our study, and existing methods to enhance its convergence efficiency were reviewed. We considered the fact that convergence is influenced by the choice of the initial vector. A preliminary study led us to apply a GA to matrix solution, where the GMRES(*m*) process starts from multiple initial vectors (chromosomes). Upon stagnation in convergence, they cross over to accelerate convergence. For the crossover algorithm, we proposed a "weighted average" algorithm that has distinct advantage over conventional crossover algorithms and demonstrate particular conformity with the GMRES(*m*) process. We refer to the combination of the GA and the weighted average for crossover as the GA–GMRES(*m*) method. We evaluated the performance of the GA–GMRES(*m*) method using four exemplary test matrices. For all the tested matrices, the proposed method exhibited two to six times faster conversion than conventional non-GA calculation.

In the GA–GMRES(*m*) method, the end user must still specify the appropriate value of *m*, which cannot be predetermined. We then attempted to address this problem of choosing and adjusting the value of *m* with minimum input from the end user. The concept of chromosome-wide stagnation was introduced, and a method to adjust *m* at the onset of stagnation was proposed. We evaluated the chromosome-wide stagnation-based approach against the conventional local stagnation-based approach to demonstrate that the proposed approach exhibits more favorable tuning of *m* (i.e., 20%–30% performance improvement).

By referencing the above achievement, we conclude that two research goals set forth in Section 1.3 have been fulfilled, i.e., (1) enabling an effective matrix forming environment that allows the end-user to describe a physical problem in one-tenth the code lines without deteriorating efficiency and (2) realizing an even more effective matrix solution method by exploiting the abundant computational resources that have been made available by many/multi-core CPUs and cloud computing.

However, there are still some limitations in the developed system and methods. For the matrix formation phase, the proposed DEQSOL system can only deal with physical phenomena whose boundary shapes and mesh topologies are stationary. Physical problems with large deformation typically appearing in structural or fluid dynamics analysis, e.g., car crash simulation or two-phase flow analysis, are beyond the scope of this study. For the matrix solution phase, although the proposed GA−GMRES(*m*) method shows satisfactory performance improvement for the selected test matrices, the

effectiveness of the method may vary significantly depending on the nature of the matrix. A performance measurement using a larger set of test matrices to find a way to determine effectiveness prior to the actual matrix calculation remains an issue.

5.2 Future Directions

Finally, we describe future problems and research directions.

(1) Application of a GA to other matrix solution methods

In Chapter 3 and Chapter 4, we selected the GMRES(*m*) method as the basis of our study, because it is one of the most promising and widely used matrix solvers for large-scale, ill-conditioned matrices. However, there are other iterative matrix solution methods where a GA can be applied in a similar manner to what we did with GMRES(*m*). This type of study would be worth investigating, because the proposed method of GA application is orthogonal to the matrix solution method used inside each chromosome; thus, we could expect a synergetic effect, as demonstrated by the combination of Look-back modification and GA–GMRES(m) in Section 3.4.3. One obvious candidate is the Bi-Conjugate Gradient Stabilized method, which is equally well known and widely used as GMRES(*m*) for complex matrices. A combination of a GA with simple iterative methods, such as Jacobi or SOR, is also a topic of interest.

(2) Utilization of PDE level information in the matrix solution phase

Throughout this dissertation, the problem description in the DEQSOL language is used only in the matrix formation phase, but not in the matrix solution phase. However, if we could draw some high-level information from the nature of the PDE (such as sparseness or symmetricity) and pass it to the matrix solution phase, the choice of matrix solver and related parameters could be performed in a more precise and automated manner.

(3) Application of machine learning techniques to matrix solution

By collecting a large set of problem descriptions (in DEQSOL or otherwise) together with a corresponding solution method and an indicator of success, we will be able to apply machine learning techniques, such as neural network or support vector machine, to generate knowledge that directly connects the problem description to a particular solution method. This would allow the system to predict the best solution method prior to execution of the simulation. As big data analytics technologies advance

in line with expansion of computational capacity and explosive accumulation of log data, this type of brute force approach is expected to become realistic and bare significant importance in not so distant future.

Acknowledgements

I express my sincere gratitude to Professor Norihisa Komoda of the Graduate School of Information Science and Technology at Osaka University for his visionary directions and constructive guidance on this research, and on writing up the thesis and academic papers. Without his support and encouragement, this study could have been neither carried out nor concluded at all.

I would like to thank Professors Makoto Onizuka and Shinji Shimojo of the Graduate School of Information Science and Technology at Osaka University for their numerous suggestions for revising this dissertation.

I also so much appreciate to Professors Shojiro Nishio and Toru Fujiwara, of Graduate School of Information Science and Technology at Osaka University for their valuable comments and supports on this research.

I would like to thank Dr. Ken Naono, Chief Researcher R&D Department, Hitachi Asia Malaysia Ltd. (currently Yokohama Research Laboratory, Hitachi Ltd.), for his valuable contribution to the idea of combining matrix computing and genetic-algorithm. It was because of his suggestion and co-authoring the earlier papers that I chose the research topic and commence the studies described in Chapter 3 and Chapter 4.

My sincere appreciation also goes to Dr. Shigeru Azuhata, Executive Vice President and the General Manager of Research and Development Group (currently Hitachi Fellow) and Senior Corporate Engineer Dr. Akira Yamamoto of Hitachi Ltd. for giving this opportunity to initiate the study in the Graduate School of Information Science and Technology at Osaka University.

I am also very grateful to Dr. Yukio Umetani, Dr. Chisato Konno, Mr. Hiroyuki Hirayama, Ms. Michiru Yamabe, Ms. Miyuki Saji, Mr. Tadashi Ota, Ms. Miiko Takeishi, and Mr. Mitsuyoshi Igai of Central Research Laboratory, Hitachi Ltd., whom I worked

with for so many years together to establish the research domain of problem solving environment that was materialized in the form of DEQSOL.

I would like to express my appreciation to Dr. Yasutsugu Takeda, Dr. Hisashi Horikoshi, Dr. Michiharu Nakamura, Dr. Eiji Takeda, Dr. Tsuneyo Chiba, Dr. Sakae Takahashi, Dr. Shigeo Nagashima, Dr. Mamoru Sugie, and Mr. Yasuhiro Inagami who were the successive managers of Central Research Laboratory, Hitachi Ltd., for their continuous guidance and advices to execute a number of projects related to matrix computing and high performance computing since I joined Hitachi Ltd.

I would like to thank Dr. Michitaka Kosaka (currently Professor at Japan Advanced Institute of Science and Technology), Dr. Akira Maeda, and Dr. Takashi Hotta for their managerial support and leadership during my assignment at Systems Development Laboratory, Hitachi Ltd.

I would also like to extend my sincere thanks to the seniors, colleagues, and juniors of the company.

Last but not least, I must express my gratitude to the current colleagues at Hitachi Systems Ltd., especially Mr. Takashi Tashiro, Mr. Takayuki Hisanaga, and Ms. Megumi Yamaguchi, for providing a lot of support so that I could proceed this study without hindrance.

References

[1] I. Fried : "*Numerical Solution of Differential Equations*", Academic Press, New York (1979).

[2] G.D. Smith : "*Numerical Solution of Partial Differential Equations*", Oxford University Press (1971).

[3] J.M. Haile : "*Molecular Dynamics Simulation*", John Wiley & Sons, Inc. (1992).

[4] D. Frenkel and B. Smit : "*Understanding Molecular Simulation*", Academic Press (1996).

[5] G.J. Haltiner and R.T. Williams : "*Numerical Prediction and Dynamic Meteorology*", Wiley (1980).

[6] K.J. Harper : "*Weather by the Numbers*: *the Genesis of Modern Meteorology*", MIT Press (2008).

[7] L. Hui, K. Amemiya, and N. Sagawa : "Simulation of the Eddy Current in the Thermal Flying Height Control Slider", *IEEE Transactions on Magnetics*, Vol. 47, No. 7, pp.1809-1812 (2011).

[8] L. Hui and N. Sagawa : "A Novel Active-Head Slider With a Shear-Mode PZT Actuator and Dual Thermal Actuator", *IEEE Transactions on Magnetics*, Vol. 49, No. 7, pp.3771-3774 (2013).

[9] R. LeVeque : "*Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*", SIAM (2007).

[10] K.J. Bathe and E.L. Wilson : "*Numerical Methods in Finite Element Analysis*", Prentice-Hall (1976).

[11] O.C. Zienkiewicz and K. Morgan : "*Finite Element Method and Approximation*", A Wiley Inter-science Publication (1983).

[12] T. Miyoshi, M. Shiratori, M. Zako, and S. Sakata, "*Finite Element Method - Analysis of Deformation and Fracture of Structural Element*", Jikkyo Publishing Company (1976). (in Japanese)

[13] T. Miyoshi : "*Introduction to Finite Element Method*", Baifukan (1978). (in Japanese)

[14] J. Crank and P. Nicolson : "A practical method for numerical evaluation of solutions of partial differential equations of the heat conduction type", in *Proceedings of Cambridge Philosophical Society*, Vol. 43, No 1, pp.50–67 (1947).

[15] T.J. Ypma : "Historical development of the Newton–Raphson method", *SIAM Review*, Vol. 37. No.4, pp. 531–551 (1995).

[16] O. Axelsson : "*Iterative Solution Methods*", Cambridge University Press (1994).

[17] J.E. Gentle : "*Matrix Algebra*", Springer (2007).

[18] J.E. Gentle : "*Fundamentals of Matrix Computations*", Wiley (2010).

[19] A. Bjorck : "*Numerical Methods in Matrix Computation*", Springer (2014).

[20] J. Reinders and J. Jeffers : "*High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches*", Morgan Kaufmann (2014).

[21] "*LS-DYNA Keyword User's Manual*", Livermore Software Corporation (2014).

[22] "*Moldflow Design Guide*", Moldflow Corporation (2006).

[23] J. Russell and R. Cohn : "*OpenFOAM*", Book on Demand Ltd. (2012).

[24] G.E. Houstis and J.R. Rice : "Computer as thinker/doer; Problem Solving Environment for Computer Science", *IEEE Computational Science & Engineering*, Vol.1, No.2, pp.11-23 (1994).

[25] S. Kawada, S. Tago, Y. Umetani, and T. Minami : "*PSE BOOK*", Baifukan (2005). (in Japanese)

[26] "*Macsyma (Mathematics and System Reference Manual)*", Macsyma Inc (1995).

[27] P. Wellin : "*Programming with Mathematica*", Cambridge University Press (2013).

[28] Y. Saad and M.H. Schultz : "GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems", *SIAM Journal on Scientific and Statistical Computing*, Vol. 7, No. 3 : pp. 856-869 (1986).

[29] Y. Saad: "*Iterative methods for sparse linear systems*", SIAM (1996).

[30] H. Muehlenbein, M. Schomisch, J. Born : "The parallel genetic algorithm as function optimizer", *Journal of Parallel Computing*, Vol. 17, No. 6, pp. 619-632 (1991).

[31] M. Nowostawski : "Parallel genetic algorithm taxonomy", In *Proceedings of the 3rd International Conference on Knowledg−Based Intelligent Information Engineering Systems*, pp. 88-92 (1999).

[32] M. Urselmann, G. Sand, and S. Engell : "A memetic algorithm for global optimization in chemical process synthesis", In *Proceedings of the 11th conference on Congress on Evolutionary Computation*, IEEE Press, pp. 1721-1728 (2009).

[33] H. R. Topcuoglu, B. Demiroz, and M. Kandemir : "Solving the Register Allocation Problem for Embedded Systems Using a Hybrid Evolutionary Algorithm", *Transactions of Evolutional Computing*, Vol. 11, No. 5, pp. 620-634 (2007).

[34] T. Tometzki and S. Engell : "Systematic Initialization Techniques for Hybrid Evolutionary Algorithms for Solving Two-Stage Stochastic Mixed-Integer Programs", *Transactions of Evolutional Computing*, Vol.15, No. 2, pp. 196-214 (2011).

[35] I. D. Mishev, N. Fedorova, S. Terekhov, B. L. Beckner, A. K. Usadi, M. B. Ray, and O. Diyankov : "Adaptive control for solver performance optimization in

reservoir simulation", *11th European Conference on the Mathematics of Oil Recovery* (2008).

[36] V.Y. Voronov and N.N. Popova : "Automatic performance tuning approach for parallel applications based on sparse linear solvers", *Parallel Computing: from multicores and GPU's to petascale*, B. Chapman et. al. (Eds.), IOS Press, pp.415-422 (2010).

[37] J. Erhel, K. Burrage, and B. Pohl: "Restarted GMRES preconditioned by deflation", *Journal of Computational and Applied Mathematics 69*, pp.303-318 (1996).

[38] N. Sagawa, C. Konno, and Y. Umetani : "Numerical Simulation Language DEQSOL", *IPSJ Journal*, Vol.30, No.1, pp.36-45 (1989). (in Japanese).

[39] C. Konno, M. Saji, N. Sagawa, and Y. Umetani: "Advanced Implicit-Solution Function of DEQSOL and its Evaluation", in *Proceedings of 1986 ACM Fall joint computer conference*, pp.1026-1033 (1986).

[40] N. Sagawa, D. Finn, and N. Hurley : "DEQSOL/AI-DEQSOL: Towards an Integrated Problem Solving Environment for Numerical Simulation", in *Proceedings of 8th Information Linkage between Industry and Applied Mathematics*, pp.63-80 (1991).

[41] N. Sagawa, D. Finn, and N. Hurley: "An Integrated Problem Solving Environment for Numerical Simulation of Engineering Problems", in *Proceedings of IFIP Working Conference on Programming Environments for High-Level Scientific Problem Solving*, pp.125-136 (1991).

[42] H. Hirayama, M. Ikeda, and N. Sagawa: "Solution functions of PDEQSOL for fluid problems", in *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pp.218-227 (1991).

[43] H. Hirayama, T. Ohta, M. Takeishi, N. Sagawa, and K. Araki: "Automatic Descretization method in the Fluid Dynamics Feature of the Finite Element Version of DEQSOL", in *Proceedings of the 37th IPSJ National Conference 2nd half 1988*, No.1, pp.40-41 (1988). (in Japanese)

[44] N. Sagawa, K. Araki, H. Hirayama, T. Ohta, and M. Takeishi: “Functional Design of the Fluid Dynamics Feature of the Finite Element Version of DEQSOL”, in *Proceedings of the 37th IPSJ National Conference 2nd half 1988*, No.1, pp.42-43 (1988). (in Japanese)

[45] N. Sagawa and Y. Umetani: “Functional Design of the simulation language DEQSOL for Parallel Computer”, in *Proceedings of the 38th IPSJ National Conference 1st half 1989*, No.3, pp.1492-1493 (1989). (in Japanese)

[46] K. Naono, N. Zakaria, T. Skurai, A.J. Pal, and N. Sagawa : “Genetic Algorithm-based Sparse Matrix Iterative Solvers for Efficient Simulation Platforms”, in *Proceedings of the 1st Asian Conference on Information Systems*, pp.171-176 (2012).

[47] K. Naono, T. Katagiri, T. Sakurai, M. Igai, S. Ohshima, H. Kuroda, S. Itoh, and K. Nakajima : “A Fully Run-time Auto-tuned Sparse Iterative Solver with OpenATLib”, in *Proceedings of 4th International Conference on Intelligent and Advanced Systems* (2012).

[48] N. Sagawa, N. Komoda, and K. Naono : “Improvement in Performance of GMRES(*m*) Method by Applying a Genetic Algorithm to the Restart Process”, in *Proceedings of the 2014 International Conference on Computational Science and Computational Intelligence*, pp.466-471 (2014).

[49] N. Sagawa, N. Komoda, and K. Naono : “Performance Improvement of a Genetic Algorithm-applied GMRES(*m*) method with Optimization of Subspace Parameter *m*”, in *Proceedings of the 3rd Asian Conference on Information Systems*, pp.34-40 (2014).

[50] N. Sagawa, N. Komoda, and K. Naono : “Proposal of a Genetic Algorithm-applied GMRES(*m*) method with Automatic Subspace Parameter Optimization”, *IEEJ Transactions on Electronics, Information and Systems*, Vol. 135, No.6 (2015). (to appear)

[51] S.M. Morris and W.E. Sciesser : “SALEM-A Programming System for the Simulation of Systems Described by Partial Differential Equations”, in *Proceedings of Full Joint Computer Conference 33*, pp.353-357 (1968).

[52] A.F. Cardenas and W.J. Karplus : “PDEL—a Language for Partial Differential Equations”, *Communications of the ACM*, Vol. 13, No.3, pp. 184-191 (1970).

[53] W. Schonauer and E. Schneph : “Software Considerations for Black Box Solver FIDISOL for Partial Differential Equations”, *ACM Transactions on Mathematical Software*, Vol.13, No.4, pp.333-350 (1987).

[54] G.O. Cook : “*ALPAL: A Tool for the Development for Large-Scale Simulation Codes*”, UCID-21482, Lawrence Livermore National Laboratory (1988).

[55] M. Ueda and T. Kawai : “DISTRAN - P.D.E. Translator”, in *Proceedings of 36th Annual convention IPSJ Japan*, pp.1019-1020 (1988). (in Japanese)

[56] J.R. Rice : “Programming Effort Analysis of the ELLPACK Language”, *SIGNUM Newsletter*, Vol. 14, pp.109-111 (1979).

[57] J.R. Rice : “ELLPACK: An Evolving Problem Solving Environment”, in *Proceedings of IFIP WG 2.5 Working Conference on Problem Solving Environments for Scientific Computing*, pp.233-245 (1985).

[58] J.R. Rice and R.F. Boisvert : “*Solving Elliptic Problems using ELLPACK*”, Springer-Verlag (1985).

[59] B.G. Backstrom : “*Simple Fields of Physics by Finite Element Analysis*”, GB Publishing (2005).

[60] F.J. Young : “*A Course in Computational Electrostatic Field Theory*”, Trafford Publishing (2012).

[61] K.A. Hawick and D.P. Playne : “Simulation Software Generation using a Domain-Specific Language for Partial Differential Field Equations”, in *Proceedings of 11th International Conference on Software Engineering Research and Practice* (2013)

[62] R.F. Boisvert and D.K. Kahaner : “DEQSOL and ELLPACK: Problem Solving Environments For Partial Differential Equations”, *Office of Naval Research Far East Information Bulletin*, Vol. 16, No. 1, pp.7-19 (1991).

[63] S. Attaway : "*MATLAB: A Practical Introduction to Programming and Problem Solving*", Butterworth-Heinemann (2013).

[64] G. Yagawa: "*Introduction to Finite Element Method in Fluid Dynamics and Heat Transfer*", Baifukan (1988). (in Japanese)

[65] H. Kawai, T Sawada, and T. Tanahashi : "A New FEM Algorithm Based on the MAC Method Suitable for the High Reynolds Flow Problems", in *Proceedings of Computational Mechanics '88*, pp.1509-1510 (1988).

[66] J.N. Reddy and D.K. Gartling : "*The finite element method in heat transfer and fluid dynamics*", CRC Press (1994).

[67] A.N. Brooks and T.J.R. Hughes : "Streamline Upwind/Petrov-Galerkin Formulations for Convection Dominated Flows with Particlar Emphasis on the Incompressible Navier‐Stokes Equasions", *Computer Methods in Applied Mechanics and Engineering*, Vol. 32, pp.199-259 (1982).

[68] N. Wansophark, A. Malatip, and P. Dechaumphai : "Streamline Upwind Finite Element Method for Conjugate Heat Transfer Problems", *Acta Mechanica Sinica*, Vol. 21, Springer-Verlag, pp.436-443 (2005).

[69] J. Baglama and L. Reichel: "Augmented GMRES-type Methods", *Numerical Linear Algebra with Applications 14*, pp. 337-350 (2007).

[70] A. Imakura, T. Sogabe, and S.L. Zhang: "A Look-back Technique of Restart for the GMRES(*m*) Method", *Applied Linear Algebra - in honor of Hans Schneider,* Novi-Sad (2010).

[71] A. Imakura, T. Sogabe, and S.L. Zhang: "A Look-back-Type Restart for the Restarted Krylov Subspace Methods to Solve Non-Hermitian Linear Systems", *Technical Report of Department of Computer Science, University of Tsukuba*, CS-TR-13-22 (2013).

[72] M.A. Gomes-Ruggiero, V.L. Rocha Lopes, and J.V. Toledo-Benavides : "A safeguard approach to detect stagnation of GMRES(*m*) with applications in Newton-Krylov methods", *Computational & Applied Mathematics*, Vol. 27, No.2, pp.175-199 (2008).

[73] A.H. Baker, E.R. Jessup, and Tz.V.Kolev : “A simple strategy for varying the restart parameter in GMRES(*m*)”, *Journal of Computational and Applied Mathematics 230*, pp.751-761 (2009).

[74] K. Naono, T. Katagiri, T. Sakurai, M. Igai, S. Ohshima, H. Kuroda, S. Itoh, and K. Nakajima : “A Fully Run-time Auto-tuned Sparse Iterative Solver with OpenATLib”, in Proceedings of *4th International Conference on Intelligent and Advanced Systems*, pp.143-148 (2012).