VISUALIZATION OF THE FOUR-DIMENSIONAL SPACE OF TWO BY TWO MATRICES WITH EMPHASIS ON SINGULAR AND ILL-CONDITIONED CASES

BY

LI-CHEN KO

Bachelor of Science in Pharmacy

Taipei Medical College

Taipei, Taiwan,

Republic of China

1987

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1992

Thesis
1992
K75v

VISUALIZATION OF THE FOUR-DIMENSIONAL SPACE OF TWO BY TWO MATRICES WITH EMPHASIS ON SINGULAR AND ILL-CONDITIONED CASES

Thesis Approved:

[illegible]

Thesis Adviser

J Chandler

[illegible]

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to my thesis advisor and the members of my committee who gave generously of their time and advice during the development and preparation of this thesis. Sincerely appreciated are the efforts of Dr. David Miller, my thesis advisor, and the other members of my committee, Drs. John Chandler and Benny Evans.

I would like to thanks my parents, my brothers, and my sister for their love and moral support. I also appreciate my friend Ting-Huan Hsiao for his suggestions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Computer graphics has become popular and common in diverse everyday contexts, since it is more easily recognized than text. In many pictorial software interfaces, icons, simple graphic symbols, are broadly used for communicating with users since they are easier to learn, need little memorization and result in fewer errors [Dam 84].

Computer graphics applications are classified into three classes: static data presentation, dynamic data presentation, and interactive graphics [Swezey 83]. Interactive computer graphics has gradually become an important area of computer science applications [Borufka 82]. Almost all interactive applications use graphics immensely in the user interface for operating the application-specific objective [Foley 90].

Computer graphics can represent large amount of scientific information in a single picture and improve understanding. Perceptible images let scientists develop their intuition and share it with collaborators by providing a way to communicate concepts that are difficult to describe verbally. Visualization renders the scientific data into some form which contains symbolic and geometrical characteristics. Frequently this is a representation of a multidimensional data set.

Graphic models generated by computers for physical, financial and economic systems are usually used as educational aids [Hearn 86]. Graphical educational aids not only provide versatile interactive environments for many topics but also provide more information about these topics. Visual effects that involve a lot of animation and interaction provide a learning environment that can help children and even adults to do their educational exercises whenever they need [Dam 84].

Interactive graphics that has tremendous user-computer interaction can not only enhance the user's ability to perceive information, and but also allow users to visualize real or imaginary objects, such as mathematical surfaces in 4-dimensional space. University of Massachusetts performed an investigation of students behavior when they are solving algebraic word problems [Coburn 85]. They found: the students get higher grades if a problem is represented in a computer programming job than if it is represented in a traditional format, when they try to solve the same algebraic problems at the first time [Coburn 85].

Eigenvalue problems is an important topic since it is involved in many scientific and engineering applications. An effective use of computer visualization can enhance the understanding and the displaying the geometric behavior of the vector field of the eigenvalues and eigenvectors.

This thesis is to develop an interactive educational visualization tool to demonstrate the four-dimensional space of two by two matrices and the eigenvectors of a selected matrix in this space, emphasizing the singular and ill-conditioned matrices and displaying them by shading. The general approach will be to slice the four-dimensional space into a series of two-dimensional images under a personal computer environment, and accept user input to select a target matrix for solving its eigenvalues and depicting the corresponding eigenvectors.

This thesis is arranged as follows: Chapter II, that is literature review, consists of visualization, eigenvalue problems and matrices, and computer aids in education. Chapter III contains the statement of the problem, and discusses the development of the interactive educational visualization tool. The user guide is discribed for the need of users when they execute this interactive tool and some examples and outputs are also shown. It also discusses the advantages and limitations of the tool, and the conclusions. Finally, chapter IV contains the summary, and the future work of this thesis.

CHAPTER II

LITERATURE REVIEW

2.1 Visualization

2.1.1 Introduction

Computer visualization not only provides friendly user interfaces to computer users, but also provides the ability to display virtually invisible structure of an object [Foley 90]. These characteristics make scientific visualization became an important topic in the late 1980s [Foley 90].

By providing the ability to represent or picture the enormous amount of data to be graphical images, scientific visualization can represent prodigious quantities of scientific data in a single picture and improve understanding, and it can render scientific data into a figure with symbolic and geometrical features [Foley 90]. This provides scientists and engineers an easy approach to understand and explain the enormous amount of data generated by the executions of programs [Foley 90]. The use of visualization in all branches of mathematics, such as graphs, diagrams and charts, would have a definite advantage in the study of mathematics.

Visualization can be implemented to display both static and dynamic images. This provides a way of communication; a series of images can generate more information than the whole one. Visualization also can represent some imagined multidimensional objects as two-dimensional or three-dimensional precise graphical representation. For example, a three-dimensional space can be sliced to be sequences of two-dimensional images along an axis [Wu 88].

The phrase scientific visualization has referred to the separate technologies for building images on computers. It consists of two-dimensional image processing, volume rendering, interactive computer graphics and high-quality rendering [Bishop 90]. For scientific and engineering visualization, now computer-produced animated movies and demonstrations of the time-varying behavior of real and simulated objects are gradually favored [Foley 90].

2.1.2 <u>Output Primitives</u>

By providing different output primitives of geometric components, computer graphic packages can be used to characterize the shape of pictures [Hearn 86]. Although different computer graphic packages support different output primitives, points and lines are the most rudimentary geometric components to define the shape of pictures [Hearn 86]. Except points and lines, polygons, curved figures, polyhedra, freeform surfaces, and character strings are also supported by some computer graphic packages [Foley 90 and

Hearn 86].

According to the different types of attributes that are provided by a certain computer graphic package, the shapes, widths, and colors of the output primitives may be displayed in versatile appearances [Foley 90]. The attributes of the output primitive classifications that were discussed in this section are listed in Table I. Lines can be displayed as dotted or dashed, fat or thin, and different colors. Areas could be filled with a certain color or with several colors defined in a pattern. Character strings might be shown reading from left to right, in a vertical line, or slanted diagonally across a computer screen. Individual characters could be displayed in different colors and sizes [Hearn 86].

TABLE I

ATTRIBUTES OF OUTPUT PRIMITIVES

Output Primitives	Attributes
Line	Type Color Width
Area	Interior style Pattern Fill color
Text	Font Color Size Orientation

2.1.3 Interactive Input Methods

An interactive graphics program system must have the ability to deal with the interactive input and allow users to input coordinate positions, select functions, or specify transformation parameters [Hearn 86]. There are several various types of hardware devices that can be applied to input data to a graphics program in an interactive system, such as keyboards, mouse, joysticks, buttons, dials, touch panels, light pens, two-dimensional, three-dimensional tablets, and voice devices [Hearn 86]. What kind of input devices should be chosen to input data to a graphics program depends on what kind of applications is applied and what kind of input data is needed.

Device-independence, which can increase portability of applications, is a primary goal in designing graphics packages [Foley 90]. Graphics programs usually use many different types of input data. The logical classification of input devices that can be divided into five types: locator, keyboard, valuator, choice, and pick [Foley 90]. The operations and corresponding physical devices of the logical classification of input devices are listed in Table II. The logical classification of input devices can make a graphics package independent of the type of hardware devices by providing a basis for designing input routines in the graphics package.

In order to facilitate users using interactive constructing pictures, there are numerous techniques that can be applied into graphics packages. The interactive picture-construction techniques consist of basic positioning methods, constraints, grids, gravity field, rubber-band methods, sketching, and dragging [Hearn 86]. Many design and painting packages apply the interactive picture-construction techniques to assist users to arrange objects, to draw figures, and to drag objectives etc.

TABLE II

THE OPERATIONS AND CORRESPONDING PHYSICAL DEVICES OF THE LOGICAL CLASSIFICATION OF INPUT DEVICES

Logical Input Devices	Operation	Physical Input Devices
Locator	Specifies a coordinate position (x, y).	Mouse, Joystick, TrackBall, Touch panel, Tablets
Keyboard	Specifies character string.	Keyboard
Valuator	Specifies scaler values.	Scaled dials
Choice	Specifies menu options and commands.	Buttons on mouse and TrackBall, Function keys on keyboard
Pick	Selects the desired entry.	Light Pen

2.1.4 <u>The Advantages of Interactive Graphics</u>

Interactive computer graphics is the most important method of making pictures. It can be used to create pictures of real objects and imaginary objects, such as mathematical surfaces in 4-dimensional space, and to build figures of data with no inherent geometry, such as survey results [Foley 90].

Static data presentations can be a good method for communicating information, but dynamic data presentations are usually better than static ones. The use of a combination of dynamic data presentations and interactive graphics is extraordinarily effective, as it allows users to master the animation by modifying the speed, the portion of the total picture shown, the percentage of enlargement or reduction displayed, and so on [Foley 90].

Interactive graphics enables broad user-computer interaction that increases the user's capability to understand data, and to envision real or fantastic objects. In order to conduct communication more efficient, interactive graphics produces more accurate results, higher-quality products, and makes less investigation and design costs.

2.1.5 Visualization Applications

Visualization software tools can be divided into two major types for computational scientists and engineers: graphics subroutine libraries, and animation applications. PHIGS+, GL, GKS, and SIGGRAPH CORE are examples of graphics libraries; MOVIE.BYU and products of Wavefront and Alias are used by scientists as animation applications [Upson 89].

Visualization applications can be found in almost every field of today's scientific studying, including mathematics, computer science, engineering, combustion science, medicine, meteorology, semiconductor devices, geosciences, biology, and chemistry. [Nielson 90]. Such as, CAD/CAM applications are broadly used in engineering [Rhodes 91].

In combustion science, M. B. Long et al. acquire experimental data from turbulent flows and flames and display them by an image where intensity or color is varying that is according to the measured quantity or the surface contour methods [Long 89].

H. Fuchs et al. discuss the techniques for displaying three-dimensional medical data, which are surface-based techniques, binary voxel rendering, and volume rendering, and suggest that volume rendering is useful for clinical applications [Fuchs 89]. The digital imaging modalities consisting of computerized tomography (CT), magnetic resonance (MR), positron emission tomography (PET) and single photon emission CT (SPECT) are applied in diagnostic medicine [Rhodes 91]. All these modalities are able to produce three-dimensional arrays of intensity values [Fuchs 89].

The paper of W. Hibbard et al. describes the functionalities of Man-computer Interactive Data Access System (McIDAS) and its applications in the earth science. Their visualizations are used for teaching at several institutions [Hibbard 89]. N. C. Kluksdahl et al. illustrate that visualization and animation of simulation data allow a better understanding of the quantum electron transport in semiconductors [Kluksdahl 89]. The paper of M. K. Seager et al. discusses tools for the graphical analysis of multitasking and microtasking applications which provide visual information of the design and debugging of parallel programs to a programmer [Seager 90].

In the paper of C. Upson et al., it describes a software system, named the application visualization system (AVS) which can combine software components, called modules, into executable flow networks or directed acyclic graphs to produce interactive scientific visualization applications quickly. AVS is designed for scientists and engineers [Upson 89].

Visualization tools provide an effective means to interpret voluminous data that bring to scientific discovery and understanding. Educators can utilize visualization by creating an image, animation sequences, or interactive simulation to support scientific learning in instructional processes, thus students can learn abstract scientific concepts rather than merely memorizing descriptions and formulas [Cunningham 90].

2.2 Eigenvalue Problems and Matrices

2.2.1 Introduction

For a square matrix A and $x = A^{-1}b$ system of linear equations where

$$A \in R^{nxn} \quad , b \in R^{n}$$

we have a solution iff

$$b \in R(A)$$

and it is unique iff

$$N(A) = \{0\}$$

(or the determinant of matrix A, denoted as |A| or det A, is not zero), where R(A) is the range of A defined by

$$R(A) = \{ y \in R^n \mid y = Ax \text{ for some } x \in R^n \}$$

and N(A) is the null space of A defined by

$$N(A) = \{ x \in R^n \mid Ax = 0 \}$$

If |A| is zero, we have either an infinite number of solutions or no solution, depending on b the vector of right-hand-sides [Golub 85 and Smith 86].

2.2.2 Eigenvalue Problems

Basically the eigenvalue problem is to solve the eigenvalues, denoted as lambda, of n homogeneous linear equations with n unknowns

$$Ax = \lambda x$$

that has a non-trivial solution; that is iff the matrix

$$(A - \lambda I)$$

is singular, since

$$Ax = \lambda x$$

could be written as

$$(A - \lambda I)\,x = 0$$

[Wilkinson 65]. For the equation, there will be only one solution x = 0, if lambda is arbitrary, or at least one non--trivial solution x corresponding to any eigenvalue [Wilkinson 65].

According to the definition of eigenvalue problem, there are two parts when we are working on the eigenvalue problem for an (n x n) matrix A. The first part is to find all scalars lambda such that

$$(A - \lambda I)$$

is singular. Then such scalars lambda are called the eigenvalues of the matrix A [Johnson 89]. The second part is to find all nonzero (n x 1) vectors x such that

$$(A - \lambda I)x = 0$$

for an given eigenvalue, lambda. Then such vectors are called the eigenvectors corresponding to the given eigenvalue [Johnson 89]. Obviously if x is a eigenvector of an (n x n) matrix, then kx is also a eigenvector of this (n x n) matrix.

Vectors are physical quantities, which have both magnitude and direction, and can be geometrically represented in two-dimensional space or three-dimensional space as directed line segments [Johnson 89]. The behavior of a vector field according to its position in space can be realized by geometric images. The eigenvalues and eigenvectors involve the geometric variation of the vector field. An example of calculating the eigenvalues and eigenvectors for a square matrix is shown in Figure 1.

2.2.3 Eigenvalue Applications

Some of the applications of eigenvalues include: solving systems of differential equations, solving optimization problems, diagonalizing linear transformations, and describing the evolution of "discrete-time systems" [Johnson 89].

Example: Find the eigenvalues and eigenvectors of a 2 x 2 matrix A,

$$A = \begin{vmatrix} 1 & -1 \\ 5 & 3 \end{vmatrix}$$

Solution: The problem involves computing

$$\det(A - \lambda I) = 0 \text{ , that is}$$

$$\begin{vmatrix} 1-\lambda & -1 \\ 5 & 3-\lambda \end{vmatrix} = (1-\lambda)(3-\lambda) - (-5)$$

$$= \lambda^2 - 4\lambda + 8 = [\lambda - (2+2i)][\lambda - (2-2i)], \text{ where } i = \sqrt{-1}$$

We get two eigenvalues: $\lambda = 2 + 2i$, $2 - 2i$

The eigenvectors of $\lambda = 2 + 2i$ *are found by solving*

$$[A - (2+2i)I]x = 0 \text{ , that is}$$

$$(-1-2i)x_1 - x_2 = 0 \text{ ...... } (1)$$

$$5x_1 + (1-2i)x_2 = 0 \text{ ...... } (2)$$

$$Eq. (1) \times (-1+2i) \text{ , get } 5x_1 + (1-2i)x_2 = 0 \text{ ...... } (3)$$

$$x = \begin{vmatrix} (1-2i)a \\ -5a \end{vmatrix} = a\begin{vmatrix} 1-2i \\ -5 \end{vmatrix} \text{ , } a \neq 0$$

The eigenvectors of $\lambda = 2 - 2i$ *are of the form*

$$x = \begin{vmatrix} (1+2i)b \\ -5b \end{vmatrix} = b\begin{vmatrix} 1+2i \\ -5 \end{vmatrix} \text{ , } b \neq 0$$

Figure 1. An example of computing the eigenvalues and eigenvectors of a 2 x 2 matrix.

The eigenvalue problems are applied to solve linear differential equations. Linear differential equations are related in classical engineering models, such as resistor/inductor/capacitor circuits in electrical engineering, dynamic models corresponding to Newton's equations in mechanical engineering, and linear reservoir models for river flow in civil engineering [Yakowitz 89].

2.2.4 Singular and ill-conditioned Matrices

For a matrix A, if its determinant is equal to zero, we call it a singular matrix. By the definition, the inverse of the matrix A, denoted as A^{-1}, does not exist. For a singular matrix, its determinant is equal to the product of its eigenvalues, so there should have at least one zero eigenvalue of a singular matrix [Wilkinson 65]. If the determinant of a matrix is not equal to zero, it is called nonsingular. Nonsingular matrices consist of two types: ill-conditioned matrices and well-conditioned matrices. The solution of a linear equation system under ill-conditioning is very sensitive to small changes in the coefficients [Golub 92 and Wilkinson 65].

An ill-conditioned matrix is a square matrix which is close to a singular matrix so that it has a small determinant compared to the magnitude of the entries [Smith 86]. The smallness of the determinant can not be used as a method to determine the conditioning of matrices, since a matrix that has a small determinant still can be well-conditioned [Golub 92]. For example,

$$\det(A) = \begin{vmatrix} 1 & 0 \\ 0 & 10^{-8} \end{vmatrix} = 10^{-8}$$

The determinant of matrix A is very small, but matrix A is perfectly well-conditioned. The reason is the lines defined by matrix A are perpendicular. They are

$$x_1 = 0$$

$$10^{-8}x_2 = 0$$

So this kind of matrices should be appropriately scaled before determining their conditioning.

Ill-conditioned matrices can be considered as an intermediate type between singular and nonsingular matrices. For approximate calculations in the computer, roundoff error blurs the boundary between singular and nonsingular matrices, since calculations are usually performed to a finite number of significant figures. The solution of linear equation system under ill-conditioning is very sensitive to small changes in the coefficients [Golub 92 and Wilkinson 65].

2.2.5 Condition Number

For an (n x n) matrix A, the condition number is defined as the norm of matrix A times the norm of the inverse of matrix A [Golub 85 and Golub 92], that is

$$cond(A) = \|A\| \|A^{-1}\|$$

One definition of the matrix norm is [Wilkinson 65 and Yakowitz 89]:

$$\|A\| = (\sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}|^2)^{1/2}$$

which is called the Frobenius norm. Another definition of the matrix norm is called the uniform matrix norm which is [Yakowitz 89]:

$$\|A\| = \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}|$$

It is classified as a subordinate matrix norm and denoted as [Wilkinson 65]:

$$\|A\|_\infty = \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}|$$

The other subordinate matrix norm is:

$$\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}|$$

The cond(A) is always greater than or equal to 1. If the condition number is very large compared to 1, the matrix A is called ill-conditioned [Smith 86]. The ill-conditioning is not precisely defined, because it counts on what type of the matrix norm one uses and what "large" one takes [Yakowitz 89].

For (n x n) matrices, to determine the conditioning of a matrix by computing condition number is helpful but laborious. It is explicitly quit difficult to compute the condition number of a matrix even though it is relatively easily to calculate the matrix norm, since it still needs to obtain the inverse of the given matrix [Smith 86]. In this study, since we only deal with a two by two case, we could still use condition number to be the major measure of ill-conditioning.

Examples of determining the conditioning of a matrix by using condition number are shown in Figure 2, Figure 3 and

Figure 4. The example in Figure 2 is a badly scaled matrix. If it is not scaled before computing its conditioning, we get its condition number is a large one that shows the matrix is not very well-conditioned. The matrix in Figure 2 is actually a well-conditioned matrix but badly scaled, so we have to scale it before computing its conditioning. Compare Figure 2 with Figure 3, we can see the difference of condition number between the matrix is not scaled and that is scaled.

2.3 Computer Aids in Education

Although computer-aided educational software may cover only a certain subject, the subjects covered by the current computer-aided educational software include biology, chemistry, engineering, geology, mathematics, and physics [Coburn 85 and Cunningham 90]. The computer educational applications can be divided into five major categories: computer-assisted instruction (CAI), computer learning environments, instructional/learning tools, computer managed instruction (CMI), and programming [Coburn 85].

2.3.1 Introduction

By providing well-designed software for educational purposes, computers become a powerful tool for teaching, learning and researching [Coburn 85 and Cunningham 90]. Visualization is the most important factor for computer aid educational software to make the achievement [Cunningham 90].

Example: Determine matrix A is ill-conditioned, singular, or well-conditioned by using the condition number.

$$A = \begin{vmatrix} 5 & 191 \\ 1 & 2 \end{vmatrix}$$

Solution: By the uniform matrix norm, finding the solution involves computing the following:

(1) matrix A

summation of the first row:	196
summation of the second row:	3
maximum of the two number:	196

(2) the inverse of A

$$A^{-1} = 1/(10-191) \begin{vmatrix} 2 & -191 \\ -1 & 5 \end{vmatrix}$$

$$= \begin{vmatrix} -2/181 & 191/181 \\ 1/181 & -5/181 \end{vmatrix}$$

summation of the first row:	193/181
summation of the second row:	6/181
maximum of the two number:	193/181

(3) Calculate cond(A)

$$cond(A) = \|A\| \|A^{-1}\|$$

$$= (196)(193/181)$$

$$= 208.994475$$

we get matrix A is not very well-conditioned

Figure 2. An example of using the condition number to determine the conditioning of a matrix

Example: Determine matrix A is ill-conditioned, singular, or well-conditioned by using the condition number.

$$A = \begin{vmatrix} 5 & 191 \\ 1 & 2 \end{vmatrix}$$

Solution: (A) By approximately scaled:

scale the first row by 191

$$\begin{vmatrix} 0.026 & 1 \\ 1 & 2 \end{vmatrix}$$

scale the second row by 2

$$\begin{vmatrix} 0.026 & 1 \\ 0.5 & 1 \end{vmatrix}$$

scale the first column by 0.5

$$\begin{vmatrix} 0.052 & 1 \\ 1 & 1 \end{vmatrix}$$

(B) By using the uniform matrix norm, involves computing the following:

(1) matrix $\begin{vmatrix} 0.052 & 1 \\ 1 & 1 \end{vmatrix}$

summation of the 1st row: 1.052
summation of the 2nd row: 2
maximum of the two number: 2

(2) the inverse of A

$$1/(0.052-1)\begin{vmatrix} 1 & -1 \\ -1 & 0.052 \end{vmatrix}$$

$$= \begin{vmatrix} -1000/948 & 1000/948 \\ 1000/948 & -52/948 \end{vmatrix}$$

summation of the 1st row: 0
summation of the 2nd row: 1052/948
maximum of the two number: 1052/948

(3) Calculate cond(A)

$$cond(A) = \|A\|\,\|A^{-1}\|$$

$$= (2)(1052/948)$$
$$= 2.219409$$

we get matrix A is well-conditioned

Figure 3. The example of Figure 2 with appropriately scaled before using the condition number to determine the conditioning of a matrix

Example: Determine matrix A is ill-conditioned, singular, or well-conditioned by using the condition number.

$$A = \begin{vmatrix} 4.1 & 1 \\ 5 & 2 \end{vmatrix}$$

Solution: By the uniform matrix norm, finding the solution involves computing the following:

(1) matrix A

summation of the first row is 5.1

summation of the second row is 7

maximum of the two number is 7

(2) the inverse of A

$$A^{-1} = 1/(8.2-5) \begin{vmatrix} 2 & -1 \\ -5 & 4.1 \end{vmatrix}$$

$$= \begin{vmatrix} 5/8 & -5/16 \\ -25/16 & 41/32 \end{vmatrix}$$

summation of the first row is 15/16

summation of the second row is 91/32

maximum of the two number is 91/32

(3) Calculate cond(A)

$$cond(A) = \|A\| \|A^{-1}\|$$

$$= (7)(91/32)$$

$$= 19.90625$$

We get matrix A is well-conditioned.

Figure 4. An example of using the condition number to determine the conditioning of a matrix

By applying the techniques of visualization, computer-aided educational software can simulate the real-world scientific phenomena and present the unseen scientific phenomena and concepts as symbolic pictures [Cunningham 90]. Without computer-aided educational software, teachers can only introduce or describe the unseen scientific phenomena and abstract concepts by words, and the students must imagine those phenomena and concepts by themselves [Coburn 85 and Cunningham 90]. By providing a friendly user interface and a clear educational goal, a well-designed computer-aided educational software can transcend learning difficulties and improve students' comprehension [Hentrel 85]. It also should be instructional, consistent, challenging, and gives students a chance to think and create imagination [Coburn 85 and Cunningham 90].

2.3.2 Computer-Assisted Instruction (CAI)

The products of computer-assisted instruction provide an environment to assist the users to learn new subjects or to enhance their knowledge that they have already had [Coburn 85]. The products of computer-assisted instruction can be divided into several major types, such as drill and practice, tutorial, simulation and demonstration.

By providing a series of questions, a drill and practice application provides a repetitive discipline environment for their users to help them be acquainted with the subject that is covered [Merrill 86]. Drill and practice applications are not responsible for giving the explanation of fundamental principles and solutions of the target subject, and they are designed for those users who have already learned the subject before. The difficulties of the questions provided by drill and practice applications usually consist of many different levels [Coburn 85], and users can gradually participate in the training of a drill and practice application. Currently the drill and practice applications cover many subjects, such as mathematics, chemistry, music, biology, and the vocabularies of different languages.

The design approach of a tutorial applications is to provide the users an interactive communication environment to learn a certain subject provided by the application [Coburn 85 and Cunningham 90]. A tutorial application not only provides a series of explanations, questions, and solutions of a certain subject, but also supports the feedback of the users' responses [Coburn 85, Cunningham 90, and Merrill 86]. Because tutorial applications are used to provide new materials of a certain subject, users are not asked to learn the fundamental principles or concepts about that subject before using a tutorial applications. A good tutorial application plays the roles of the teacher and textbook.

Simulation applications can simulate the real-world situations and scientific phenomena [Cunningham 90]. By providing visualization techniques, this sort of applications consists of an interactive user interface, dynamic or static graphics, and sound effect [Merrill 86]. Simulation applications allow the users to gain preparatory experience of the real-word situations by practicing with the provided simulative environments [Merrill 86]. The subjects covered by this sort of applications include engineering designs and technical training.

Demonstration applications can not only represent the features and relationships of unseen scientific phenomena and concepts by symbolic pictures, but also display the solving routines of a certain task by automatic illustrations [Cunningham 90]. The techniques of visualization and sound effect are also applied in developing demonstration applications [Merrill 86]. This sort of applications can increase the user's imagination and understanding of unseen scientific phenomena and concepts. Nowadays the demonstration applications cover many subjects, such as planetary motions, chemical reactions, genetic structures, and multi-dimensional space representations.

2.3.3 Computer Learning Environments

The products of computer learning environments only provide the users an environment of practicing cognitive and creative processes [Coburn 85 and Cunningham 90]. The products of computer learning environments do not have a specific instructional goal that is the major difference between the products of computer learning environments and those of computer-assisted instruction [Coburn 85]. The practices of problem solving, design, and decision making are continuous in such educational products.

2.3.4 Instructional/Learning Tools

The applications of this category emphasize how to assist users in solving a certain task [Coburn 85 and Merrill 86]. This sort of product are classified as tools, such as drawing devices, numerical analysis programs, data processors, word processors, sound synthesizers, and instrument monitoring devices [Coburn 85 and Merrill 86]]. Users can utilize these applications to accomplish their tasks more easily.

2.3.5 Computer-Managed Instruction (CMI)

The purpose of most computer-managed instruction tools is to assist educators in managing classroom instructional process [Hentrel 85 and Merrill 86]. The rudimentary CMI programs are basically record-keeping but the sophisticated CMI programs are not. According to the recorded grades and responses of a student, a sophisticated CMI program can diagnose the studying pattern of the student and prescribe it for the student.

2.3.6 Programming

By designing, coding, and debugging a program for accomplishing a certain specification, user can practice the problem solving skills and programming skills [Merrill 86]. The major purpose of this sort of applications is to improve the user's problem solving skills, but not to make the user to be a professional programmer [Merrill 86]. For this sort of applications, the applications play the role of the students and users play the role of the teachers.

CHAPTER III

EDUCATIONAL VISUALIZATION TOOL

3.1 Statement of the Problem

Given a two by two matrix, how do we display the four-dimensional space of nearby matrices and some of their eigenvalues and eigenvectors in a sequence of two-dimensional images by an interactive visualization program?

3.2 Introduction

Because two by two matrices have four entries, we can establish a mapping between the points of a four-dimensional space and the set of all two by two matrices. Then the set of singular matrices should correspond to some subset of the space. Since a four-dimensional space can be considered as the cartesian product of two two-dimensional spaces, we can image a set of slices of the four-dimensional space, each containing some curve or surface which correspond to the singular matrices. The ill-conditioned matrices presumably would correspond to points near the curve or surface, because they are close to singular matrices.

Evidently, if each point in a four-dimensional space corresponds to a two by two matrix, we can also associate the eigenvalues and eigenvectors of the matrix to the same point.

Since a graphical depiction of such vectors would occupy a number of screen pixels, obviously we would display such symbols only for suitably spaced samples in the four-dimensional space. However, combining such a display with the depiction of the curves or regions where matrices are singular or ill-conditioned would provide a visual sense of the relationships between these matrix properties which is probably impossible to get in any other way.

This thesis is to develop an interactive educational visualization tool that is designed to show the curves and points of singular and ill-conditioned matrices and eigenvectors of a matrix on a computer screen in order to provide a better understanding for a simple algebraic problem.

3.3 Design Approach

The interactive educational visualization tool is to display the behavior of a two by two matrix and its eigenvalues and eigenvectors by a series of two-dimensional images by slicing a four-dimensional space of points corresponding to matrices close to the given matrix. All the elements of the two by two matrix are real numbers. This tool emphasizes singular and ill-conditioned matrices by shading the points corresponding to such matrices. The interactive visualization tool is developed in a personal computer environment, using the standard C programming language and turbo C compiler.

The development of this interactive visualization program consists of two parts: user interface design, and matrix conditioning and eigenvalue problems. They will be discussed in Section 3.3.1 and Section 3.3.2.

Since the interactive visualization tool is designed for those user who has theoretical knowledge of singular and ill-conditioned matrices and understands the concept of eigenvalues and eigenvectors, there is no more training should be required for viewers to understand the demonstration of this interactive educational visualization tool.

The execution of the interactive visualization tool program consists of four steps: create user interface, interactively ask the user to input required information, examine the conditioning of two by two matrices and solve eigenvalues and eigenvectors of a selected matrix, and output the corresponding figures. In the first step, the user interface of interactive visualization tool will be shown on the user's screen and provide multi-level pull down menu. If the user inquires to view a specified two dimensional space by selecting a two by two matrix, the second step will be invoked. In the second step, the interactive visualization tool will interactively ask user to input required information of the two by two matrix. In the third step, the interactive visualization tool will call its functions to solve the eigenvalues and eigenvectors of the matrix given in the second step. In the fourth step, the interactive visualization tool will show the two-dimensional spaces where the selected matrix exists, depict its eigenvectors in the planes and show the curves and regions of singular and ill-conditioned matrices.

3.3.1 User Interface Design

This interactive educational visualization tool can accept user input from both the keyboard and the Microsoft compatible mouse. It also uses multi-level pull down menu as the user interface. The commands of the tool will be discussed in the Section 3.5.

The reason why the interactive educational visualization tool is designed to accept mouse input is that the mouse is a better input device for interactive graphics environments. The keyboard is not always an appropriate input device for interactive graphics programs, since it can not randomly select locations on the screen like the mouse does [Weiskamp 89]. Although the keyboard can support an emulated mouse cursor shown on the screen, it still does not have high speed that can move cursor quickly and provide useful positioning features [Weiskamp 89]. The visualization tool can access the mouse driver by calling int86() supported by Turbo C to execute the PC software interrupt 33h. If a mouse dose not exists, this visualization tool can emulate a mouse cursor shown on left top screen and moved using the keyboard.

This interactive educational visualization tool can display a series of two-dimensional planes by slicing the four-dimensional space formed by the matrices which are close to a given matrix, and show the curve of singular matrices and the regions of ill-conditioned matrices and depicting the eigenvectors of a selected matrix when users select 'Demo' function. The tool can also accept user input from the keyboard to select a target matrix and show the same kind of information as mentioning at the above on two two-dimensional planes where the selected matrix exists, when users select 'Option' function. It also provides on screen help that will be shown on screen when users select 'Help' function.

3.3.2 Matrix Conditioning and Eigenvalue Problems

The methods of the solving eigenvalues and eigenvectors has been discussed in Chapter II. The interactive educational visualization tool applies the methods discussed in Section 2.2.2 to solve eigenvalues and the corresponding eigenvectors. It determines the conditioning of a matrix by calculating its condition number. The definition of the condition number is described in Section 2.2.5. The interactive educational visualization tool applies the definition of the uniform matrix norm to calculate the condition number. It also performs the checking if a matrix is badly scaled, if it finds a badly scaled matrix then it will scale the matrix first before calculating its condition number.

3.4 User Guide

Users can type "matrix" to invoke the interactive visualization tool. After successfully invoking the interactive visualization tool, the first level of user interface of the interactive visualization tool will be shown on the user's screen. The first level of the user interface screen is shown in Figure 4.

Users can click any button on the mouse to invoke a desired command by moving the mouse cursor to the suitable box of the command, if a mouse exists. If there is no mouse available, users can use the emulated mouse cursor to click an appropriate box. Users can press a legal command key from the keyboard to request the tool executing. The tool favors any keyboard action, even if a mouse is available.

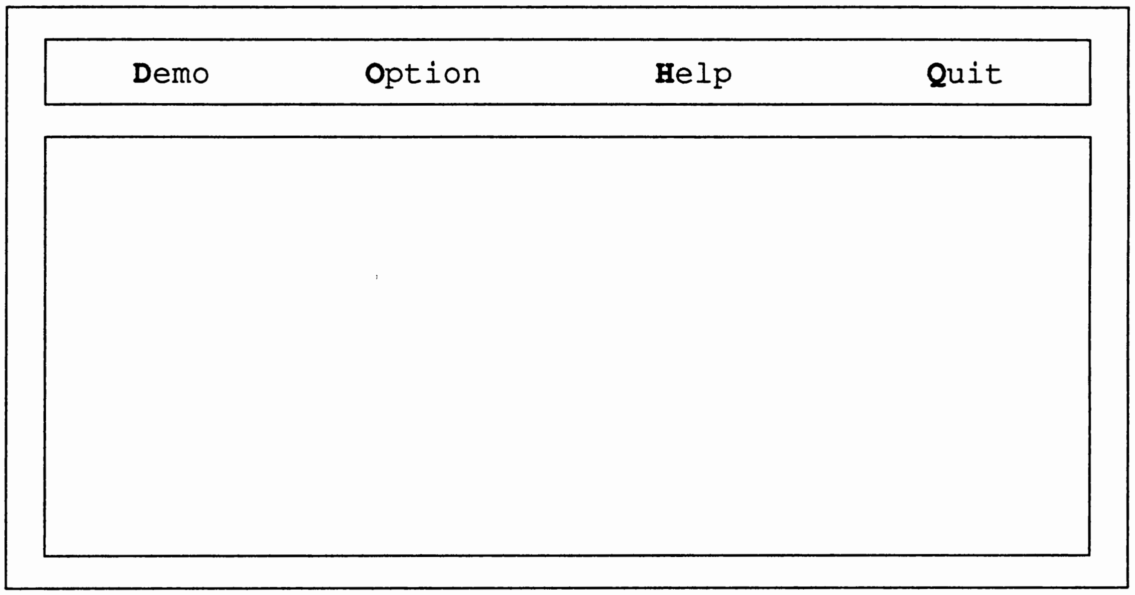


Figure 5. The screen of the first level of user interface of the interactive visualization tool

The keys of the keyboard that support the emulated mouse cursor are the following:

'Page Up' and 'Page Down': Both of them emulate the mouse button clicked.

'Home': Move the emulated mouse cursor to right top corner.

'End': Move the emulated mouse cursor to right bottom corner.

'Insert': Move the emulated mouse cursor to left top corner.

'Delete': Move the emulated mouse cursor to left bottom corner.

'right' of arrow key: Move the emulated mouse cursor right.

'left' of arrow key: Move the emulated mouse cursor left.

'up' of arrow key: Move the emulated mouse cursor up.

'down' of arrow key: Move the emulated mouse cursor down.

The reserved command keys of the keyboard are discussed as the following. Users can either type the upper or the lower case of d, o, h, or q to invoke desired command.

The legal commands of the first level of the user interface are **D**, **O**, **H**, and **Q**. There is no pull down menu for the items "Demo" and "Quit". The item "Help" contains one pull down menu. The item "Option" consists of two levels of pull down menu.

The legal commands of the first level of the user interface are introduced as following:

D or **d**: This command will invoke the demonstration function of the interactive visualization tool to show how the interactive visualization tool works. The demonstration function display a series of two-dimensional planes by slicing a four-dimensional space, and it needs a data file that should be put in drive B and named 'matrix.dat' to read selected matrices for displaying figures. This command does not contain another pull down menu.

O or **o**: This command will show a pull down menu that consists of three items, "**S**etup Variables", "**E**nlarge", and "**R**educe".

H or **h**: This command will show a pull down menu that consists of three items, "Help About Demo", "Help About Option", and "Help About Quit".

Q or **q**: This command will quit from the current function.

If the item "Option" has been chosen, the items of the pull down menu will be shown on the user's screen (see Figure 5). The legal commands of this pull down menu are upper or lower cases **S**, **E**, and **R**.

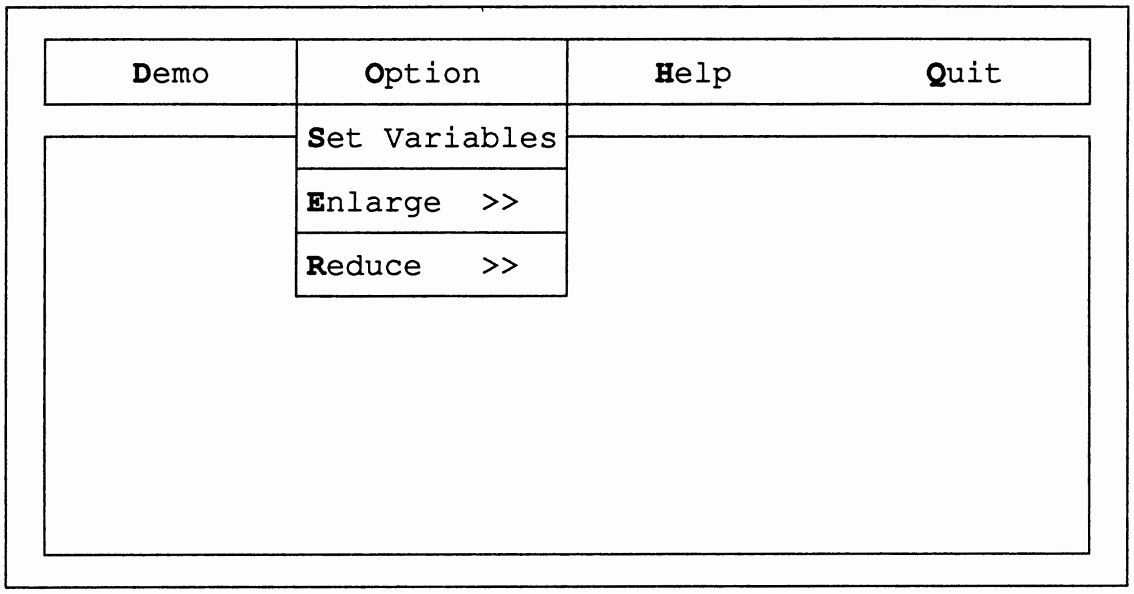


Figure 6. The screen of user interface of the item "Option"

The legal commands of the item "Option" are introduced as following:

S or **s**: This command will interactively ask user to input required information for performing the functions of the interactive visualization tool.

E or **e**: This command will show a pull down menu that consists of two items, "Enlarge Figure Only" and "Enlarge Figure / Old Figure".

R or **r**: This command will show a pull down menu that consists of two items, "Reduce Figure Only" and "Reduce Figure / Old Figure".

If the item "Enlarge" has been chosen, the items of the pull down menu will be shown on the user's screen (see Figure 6). The legal commands of this pull down menu are upper or lower cases **O** and **F**.

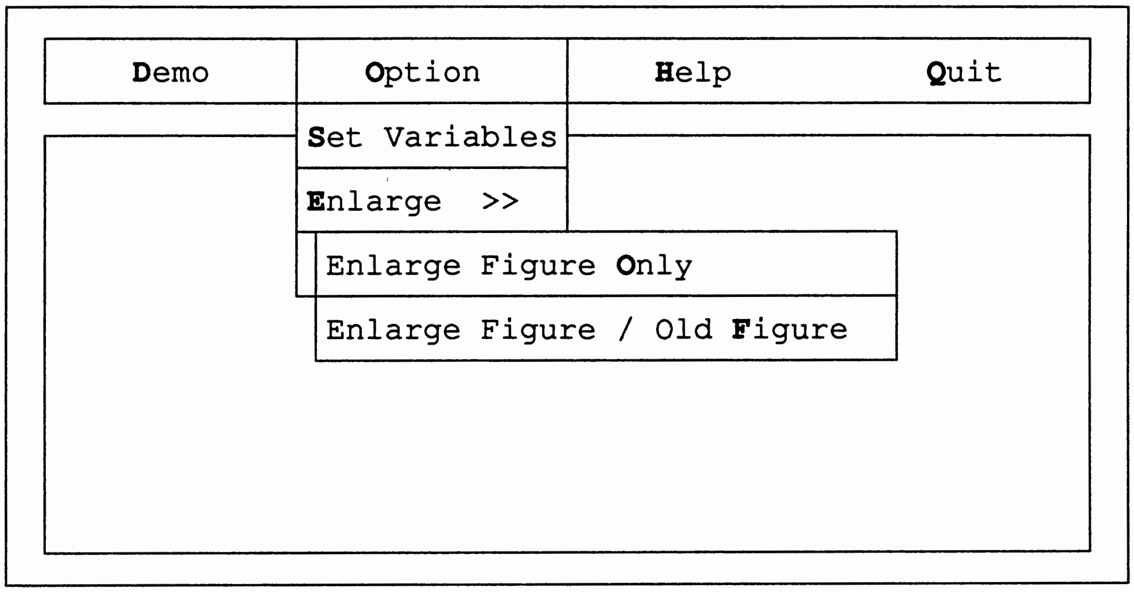


Figure 7. The screen of user interface of the item "Enlarge"

The legal commands of the item "Enlarge" are introduced as following:

O or **o**: This command will reduce the scales of the axes and relatively enlarge the scope of the figure. The original figure will be erased.

F or **f**: This command will reduce the scales of the axes and relatively enlarge the scope of the figure. The original figure will not be erased.

If the item "Reduce" has been chosen, the items of the pull down menu will be shown on the user's screen (see Figure 7). The legal commands of this pull down menu are upper or lower cases **O** and **F**.

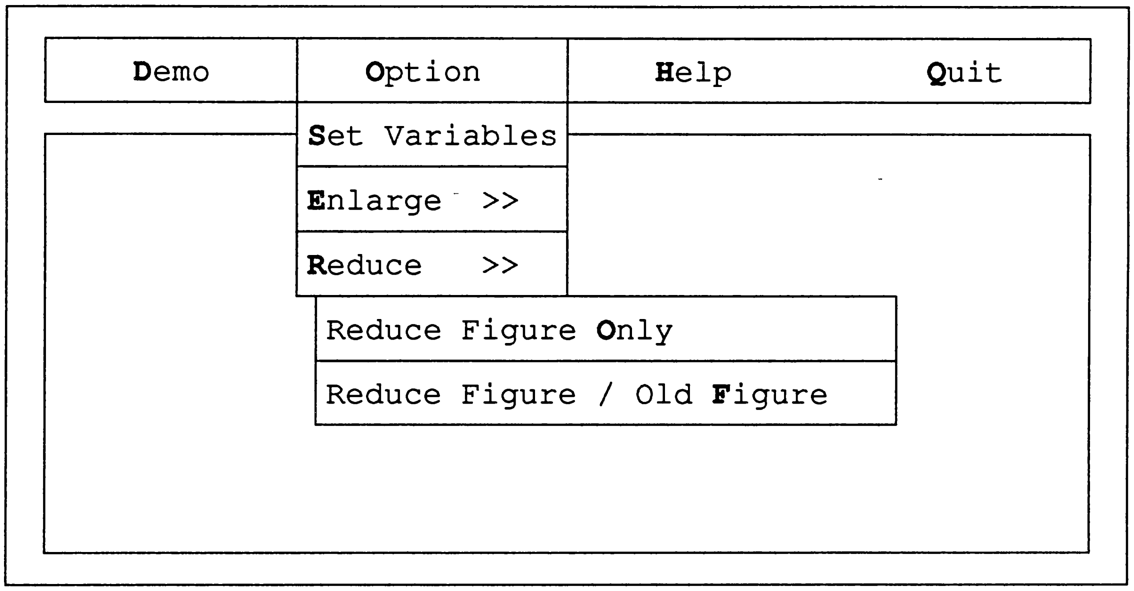


Figure 8. The screen of user interface of the item "Reduce"

The legal commands of the item "Reduce" are introduced as following:

O or **o**: This command will enlarge the scales of the axes and relatively reduce the scope of the figure. The original figure will be erased.

F or **f**: This command will enlarge the scales of the axes and relatively reduce the scope of the figure. The original figure will not be erased.

If the item "Help" has been chosen, the items of the pull down menu will be shown on the user's screen (see Figure 8). The legal commands of this pull down menu are upper or lower cases **D**, **O** and **Q**.

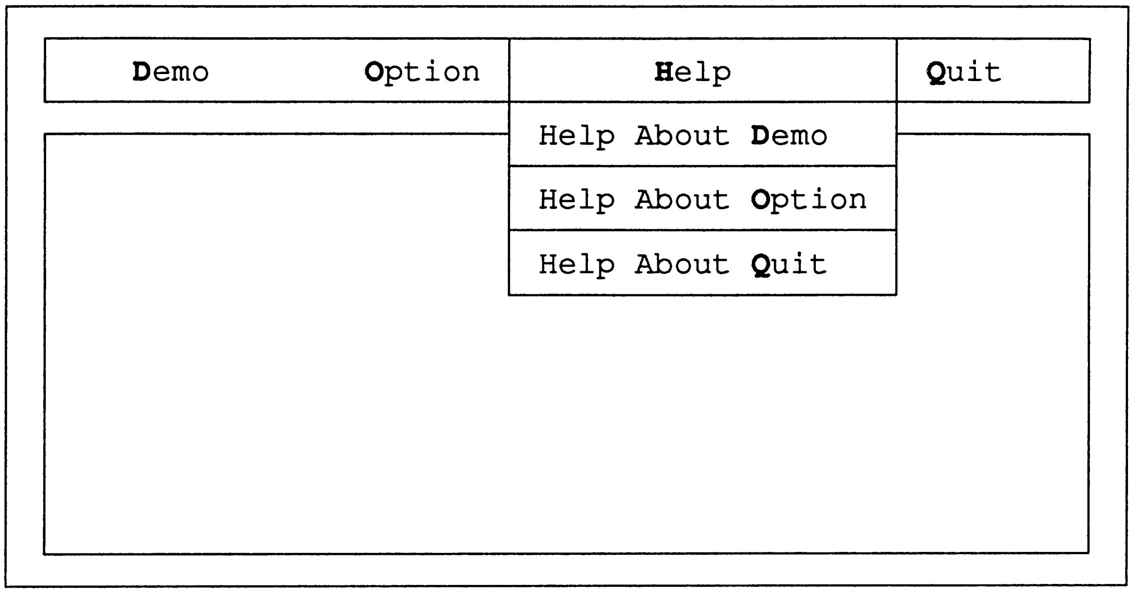


Figure 9. The screen of user interface of the item "Help"

The legal commands of the item "Help" are introduced as following:

D or **d**: This command will output the explanation of the item "Demo" to the user's screen.

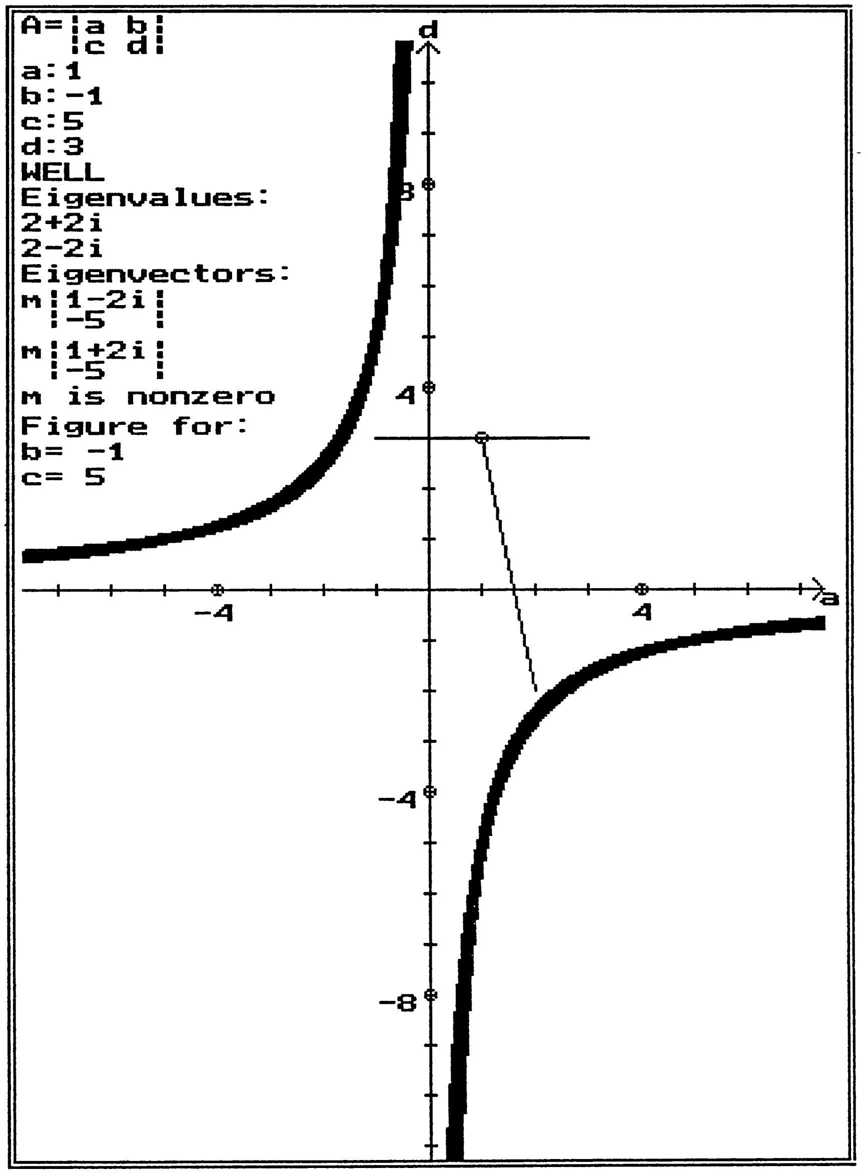
O or **o**: This command will output the explanation of the item "Option" to the user's screen.

Q or **q**: This command will output the explanation of the item "Quit" to the user's screen.

3.5 Examples and Outputs

After successfully invoking the interactive visualization tool and getting the first level of user interface shown on the screen (see Figure 4), users can either choose to execute "**D**emo", "**O**ption", "**H**elp" or "**Q**uit" function whose command keys were listed in Section 3.4. Here we only emphasize "**D**emo" and "**O**ption" function.

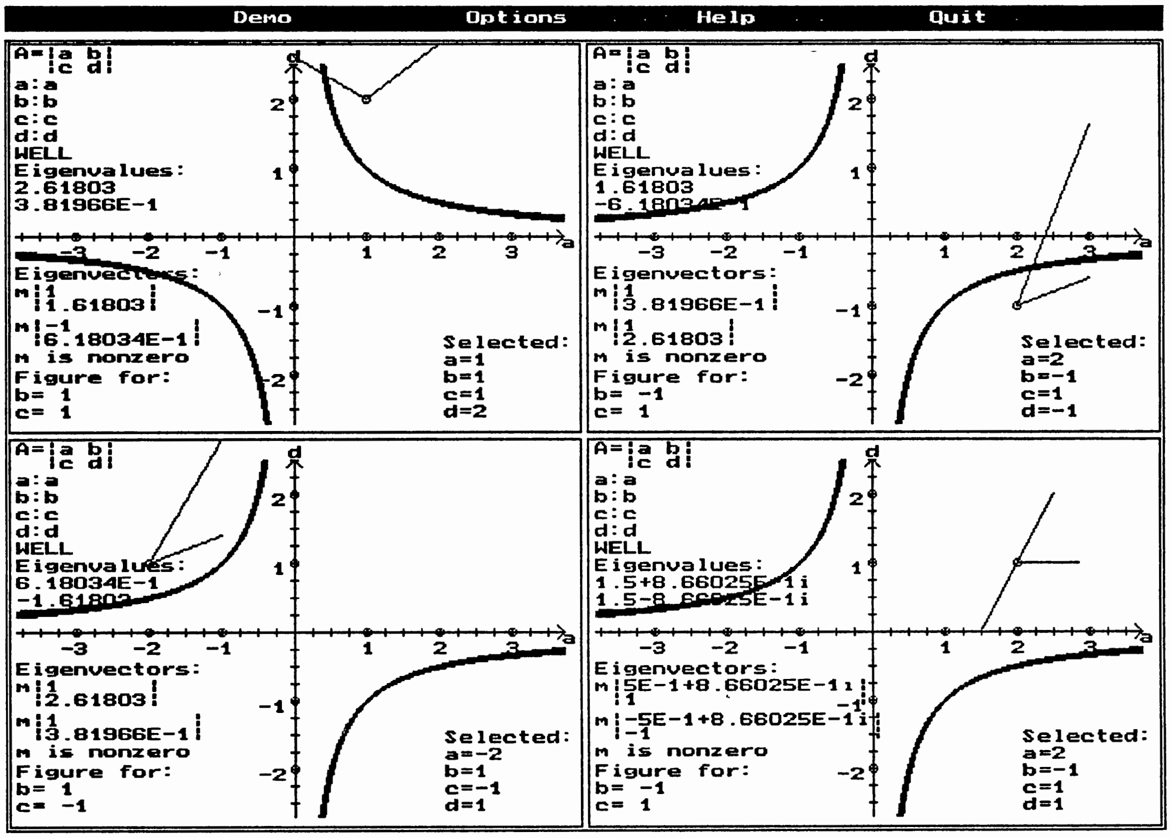
One of the output screen of a selected matrix is shown in Figure 10. Each of those two hyperbolas consists of the points that represent the singular or ill-conditioned matrices, where the points of the singular matrices are displayed in WHITE color and the area of the ill-conditioned matrices are shown in LIGHTRED color on the computer screen. The directed line segments of the point (1, 3) of a-d axes of Figure 10 represent the corresponding eigenvectors of that point. The real parts of the eigenvector corresponds to the eigenvalue 2+2i is displayed in LIGHTMAGENTA color, and that of the eigenvector corresponds to eigenvalue 2-2i is shown in LIGHTGREEN color on the computer screen (see Figure 10). In this example, these two eigenvectors have complex parts which are not the same value (one is -2i, the other is +2i), so they are shown in LIGHTGRAY and BROWN colors as two different lines on computer screen. If the complex parts of these two eigenvectors are the same, they will be overlapped at the same position and shown in BROWN color on the computer screen.



```
A=|a b|
  |c d|
a:1
b:-1
c:5
d:3
WELL
Eigenvalues:
2+2i
2-2i
Eigenvectors:
m|1-2i|
 |-5  |
m|1+2i|
 |-5  |
m is nonzero
Figure for:
b= -1
c= 5
```

Figure 10. One of the output screen of a selected matrix

One of the output screen of "**D**emo" function is shown in Figure 11. It consists of four figure windows representing some sliced planes of the four-dimensional space of two by two matrices. The screen of "**S**etup Variables" of "**O**ption" function is shown in Figure 12. The output screen of a selected matrix with two windows is shown in Figure 13, and its corresponding reduced figures (reduced by the scales of a and d axes being divided by two) after invoking "Reduce Figure **O**nly" of "**R**educe" item of "**O**ption" function is shown in Figure 14.



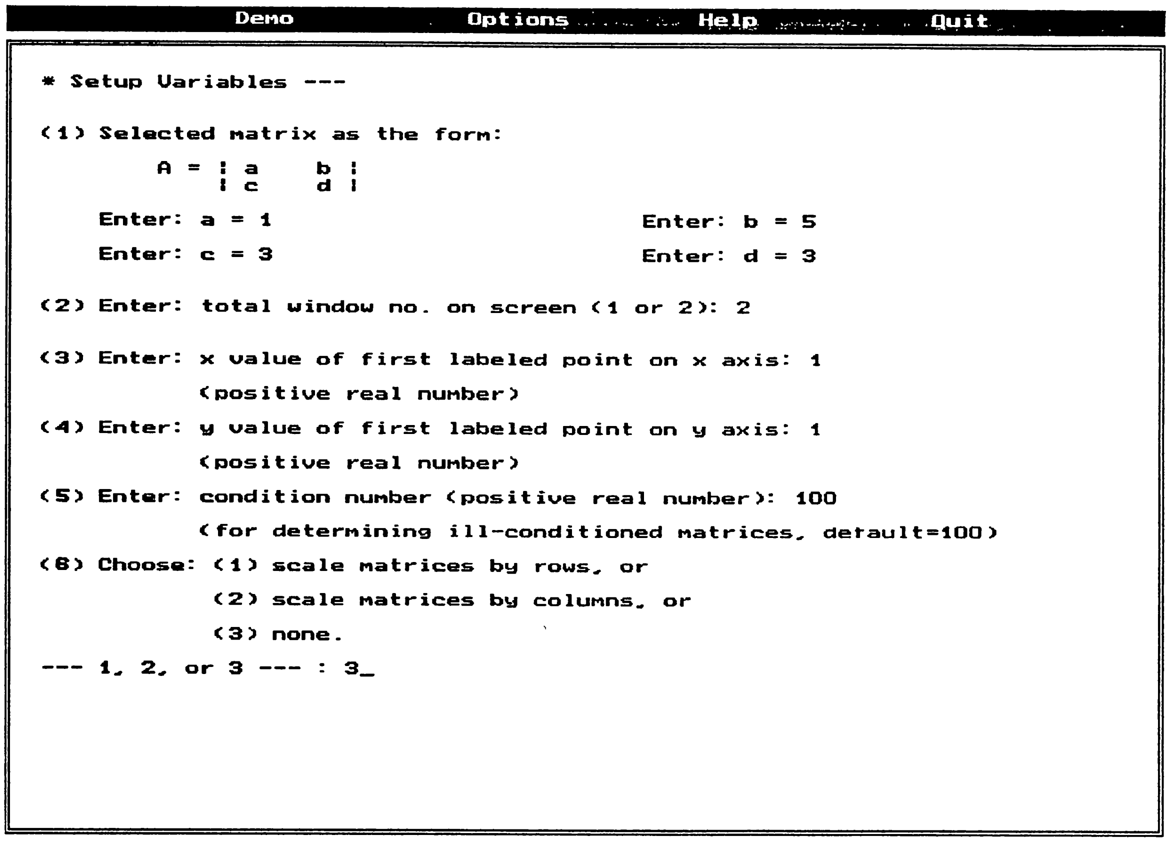
```
A=|a b|
  |c d|
a:a
b:b
c:c
d:d
WELL
Eigenvalues:
2.61803
3.81966E-1
Eigenvectors:
m|1        |
 |1.61803  |
m|-1          |
 |6.18034E-1  |
m is nonzero
Figure for:
b= 1
c= 1
                Selected:
                a=1
                b=1
                c=1
                d=2
```

```
A=|a b|
  |c d|
a:a
b:b
c:c
d:d
WELL
Eigenvalues:
1.61803
-6.18034E-1
Eigenvectors:
m|1           |
 |3.81966E-1  |
m|1        |
 |2.61803  |
m is nonzero
Figure for:
b= -1
c= 1
                Selected:
                a=2
                b=-1
                c=1
                d=-1
```

```
A=|a b|
  |c d|
a:a
b:b
c:c
d:d
WELL
Eigenvalues:
6.18034E-1
-1.61803
Eigenvectors:
m|1        |
 |2.61803  |
m|1           |
 |3.81966E-1  |
m is nonzero
Figure for:
b= 1
c= -1
                Selected:
                a=-2
                b=1
                c=-1
                d=1
```

```
A=|a b|
  |c d|
a:a
b:b
c:c
d:d
WELL
Eigenvalues:
1.5+8.66025E-1i
1.5-8.66025E-1i
Eigenvectors:
m|5E-1+8.66025E-1i |
 |1                |
m|-5E-1+8.66025E-1i|
 |-1               |
m is nonzero
Figure for:
b= -1
c= 1
                Selected:
                a=2
                b=-1
                c=1
                d=1
```

Figure 11. One of the output screen of "Demo" function

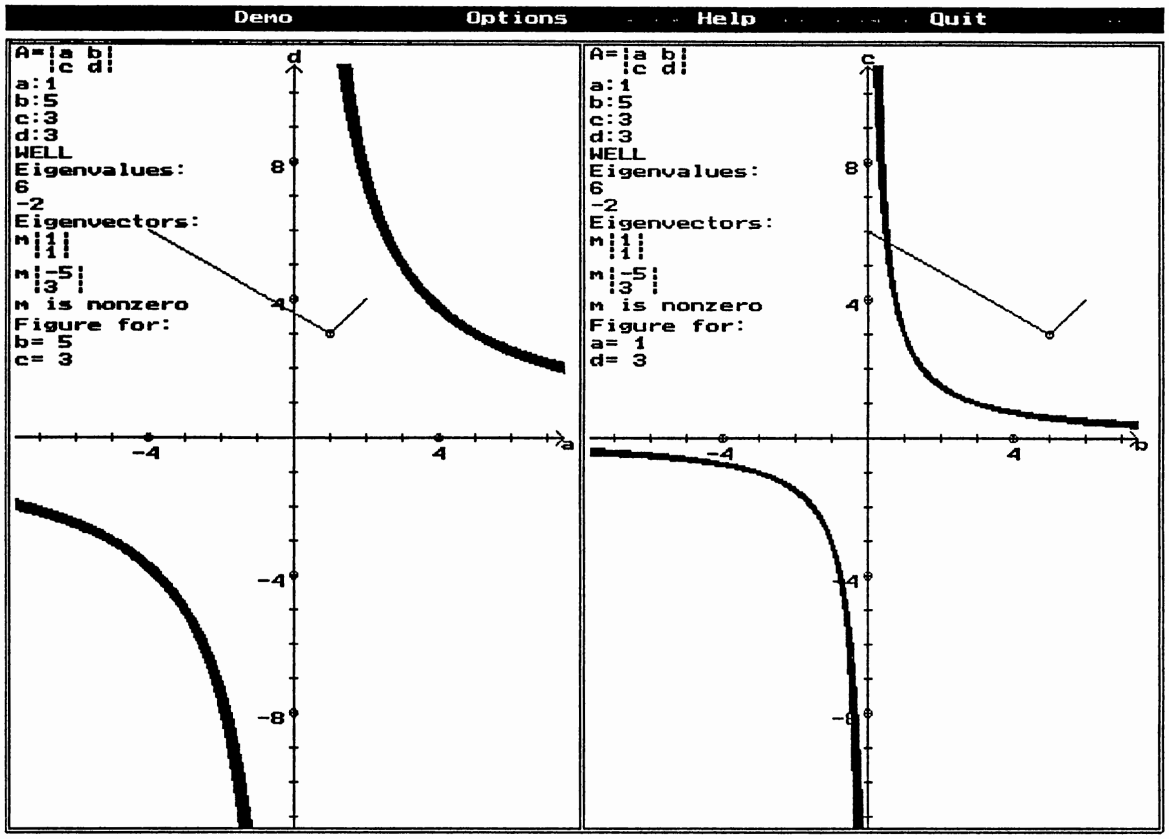


```
* Setup Variables ---

(1) Selected matrix as the form:
        A = | a    b |
            | c    d |
    Enter: a = 1                         Enter: b = 5
    Enter: c = 3                         Enter: d = 3

(2) Enter: total window no. on screen (1 or 2): 2

(3) Enter: x value of first labeled point on x axis: 1
           (positive real number)
(4) Enter: y value of first labeled point on y axis: 1
           (positive real number)
(5) Enter: condition number (positive real number): 100
           (for determining ill-conditioned matrices, default=100)
(6) Choose: (1) scale matrices by rows, or
            (2) scale matrices by columns, or
            (3) none.
--- 1, 2, or 3 --- : 3_
```

Figure 12. The screen of "Setup Variables" of "Option" function

3.6 Advantages and Limitations

The interactive visualization tool is designed to allow human errors by showing error messages on computer screen and providing opportunities for users to correct errors. Since it can access the Microsoft compatible mouse, it provides a much better interactive graphics program environment for users. Even if a mouse is not available, this visualization tool can emulate a mouse cursor moved using the keyboard. Users can also use a specific command to request a specific function. It is also easy to be used and should be portable. It not only supports users to select a matrix with real components and displays the corresponding figures, but also provides users to select a general matrix, such as its components are a, b, c, and d.

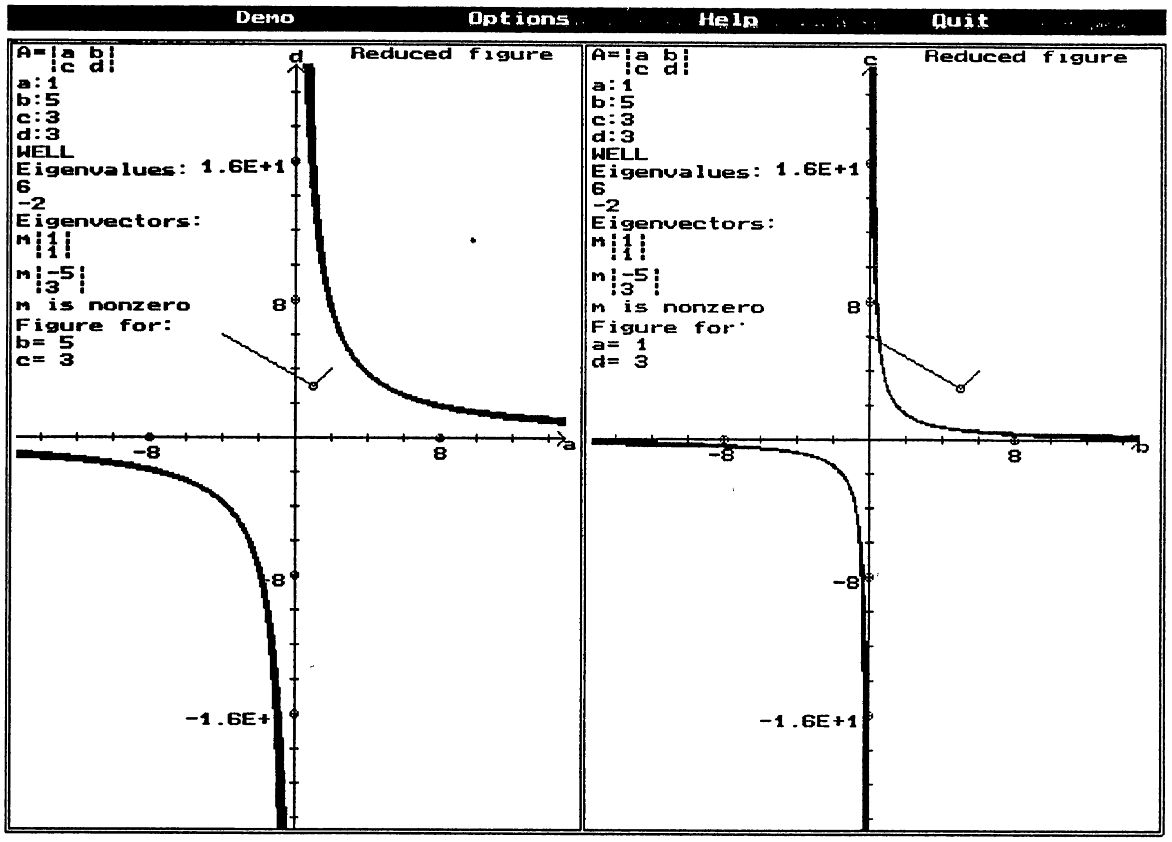


```
A=|a b|
  |c d|
a:1
b:5
c:3
d:3
WELL
Eigenvalues:
6
-2
Eigenvectors:
m|1|
 |1|
m|-5|
 |3 |
m is nonzero
Figure for:
b= 5
c= 3
```

```
A=|a b|
  |c d|
a:1
b:5
c:3
d:3
WELL
Eigenvalues:
6
-2
Eigenvectors:
m|1|
 |1|
m|-5|
 |3 |
m is nonzero
Figure for:
a= 1
d= 3
```

Figure 13. The output screen of a selected matrix with two windows

The interactive educational visualization tool can not handle a matrix with complex components. Since it only show figures in two-dimensional space, the complex part of a eigenvector is also displayed to be a color line the same as the real part of the eigenvector is displayed.



```
A=|a b|
  |c d|
a:1
b:5
c:3
d:3
WELL
Eigenvalues: 1.6E+1
6
-2
Eigenvectors:
m|1|
 |1|
m|-5|
 |3 |
m is nonzero
Figure for:
b= 5
c= 3
```

```
A=|a b|
  |c d|
a:1
b:5
c:3
d:3
WELL
Eigenvalues: 1.6E+1
6
-2
Eigenvectors:
m|1|
 |1|
m|-5|
 |3 |
m is nonzero
Figure for:
a= 1
d= 3
```

Figure 14. The reduced figures of the same example of Figure 13

3.7 Conclusions

The performance criteria of an educational visualization software is an important factor when it is evaluated [Hentrel 85]. There are several performance criteria which are: if the educational visualization program have clear, friendly instructions for user to understand easily, if the program is designed by a good guidance strategy, if the program is dependable, if the program is interactively executed, if the program is executed with an continuously displayed educational window [Hentrel 85].

The educational visualization program is developed for assisting students to get better understanding a simple example of linear algebra. It is developed under a personal computer environment for displaying the four-dimensional space of two by two matrices, emphasizing singular and ill-conditioned matrices, by slicing it into a series of two-dimensional images. Eigenvectors and eigenvalues of matrices corresponding to selected points in the space are also displayed graphically. It is a friendly program and easy to understand, and is designed as an interactive tool. It also has an instructional window displayed continuously when it is executed. The educational visualization program is consistent and dependable.

CHAPTER IV

SUMMARY AND FUTURE WORK

4.1 Summary

In Chapter I, a brief overview of advantages of computer graphics and interactive visualization programs were introduced. The general classifications of graphics software were discussed. The visualization applied in education filed and the important of eigenvalues problems for scientists and engineers were outlined.

In Chapter II, output primitives of visualization and the interactive input methods, such as locator, keyboard, valuator, choice, and pick were presented. The advantages of interactive graphics and visualization applications in the fields of computer science, combustion science, medicine, semiconductor devices, geosciences, education were discussed. Eigenvalue problems and eigenvalue applications were outlined. The definitions of singular and ill-conditioned matrices and the major measures of ill-conditioning of a matrix were discussed. The introduction of computer aids applied in education and the classifications of computer educational applications were discussed.

In Chapter III, the statement of the problem and the details of design approach of the interactive educational visualization program were presented. The user guide of the interactive educational visualization program were introduced, and examples and outputs were shown. The advantages and limitations of this tool was discussed and the conclusions of the interactive educational visualization tool were also discussed.

4.2 Future Work

Interactive visualization tools can increase the user's ability to understanding information, especially the huge data, and to envision physical or abstract objects, such as mathematical surfaces in 4-dimensional space.

If we can add some facilities in the educational visualization program to improve the flexibility of interaction, the educational visualization program will become more helpful for the users to get more information about the abstract concepts of the geometric properties of matrices and eigenvalues and eigenvectors of a selected matrix by changing the size of the region displayed, the spacing of the slices, or the viewpoint from which the space is seen.

REFERENCES

[Bishop 90]
G. Bishop, M. Monger, and P. Ramsey, "A Visualization Programming Environment for Multicomputers," IEEE Computer Graphics and Applications, vol. 10, July 1990, pp. 50-58.

[Borufka 82]
H. G. Borufka, and H. W. Kuhlmann, "Dialogue Cells: A Method for Defining Interactions," IEEE Computer Graphics and Applications, vol. 2, July 1982, pp. 25-33.

[Coburn 85]
P. Coburn, P. Kelman, N. Roberts et al., Practical Guide to Computers in Education, 2nd Edition, Addison-Wesley Publishing Company, Reading, Mass., 1985.

[Cunningham 90]
S. Cunningham, J. R. Brown, and M. McGrath, "Visualization in Science and Engineering Education," Visualization in Scientific Computing, IEEE Computer Society Press, Los Alamitos, California, 1990, pp. 48-57.

[Dam 84]
A. van Dam, "Computer Software for Graphics," Scientific American, vol. 251, no. 3, September 1984, pp. 146-159.

[Foley 90]
J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, Computer Graphics: Principles and Practice, Addison-Wesley Publishing Company, Reading, Mass., 1990.

[Fuchs 89]
H. Fuchs, M. Levoy, and S. M. Pizer, "Interactive Visualization of 3D Medical Data," Computer, vol. 22, no. 8, August 1989, pp. 46-51.

[Golub 85]
G. H. Golub, and C. F. van Loan, Matrix Computations, The John Hopkins University Press, Baltimore, Maryland, 1985.

[Golub 92]
G. H. Golub, and J. M. Ortega, Scientific Computing and Differential Equations, The Academic Press, Inc., San Diego, California, 1992.

[Hearn 86]

D. Hearn, and M. P. Baker, Computer Graphics, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[Hentrel 85]
B. K. Hentrel, and L. Harper,Computers in Education: A Guide for Educators, Ann Arbor: University of Michigan Press, 1985.

[Hibbard 89]
W. Hibbard, and D. Santek, "Visualizing Large Meteorological Data Sets," Computer, vol. 22, no. 8, August 1989, pp. 53-57.

[Johnson 89]
L. W. Johnson, R. D. Riess, and J. T. Arnold, Introduction to Linear Algebra, Addison-Wesley Publishing Company, Reading, Mass., 1989.

[Kluksdahl 89]
N. C. Kluksdahl, A. M. Kriman, and D. K. Ferry, "The Role of Visualization in the Simulation of Quantum Electronic Transport in Semiconductors," Computer, vol. 22, no. 8, August 1989, pp. 60-66.

[Long 89]
M. B. Long, K. Lyons, and J. K. Lam, "Acquisition and Representation of 2D and 3D Data from Turbulent Flows and Flames," Computer, vol. 22, no. 8, August 1989, pp. 39-45.

[Merrill 86]
P. F. Merrill, M. N. Tolman, L. Christensen, K. Hammons, B. R. Vincent, and P. L. Reynolds, Computers in Education, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[Nielson 90]
G. M. Nielson, B. Shriver, and L. J. Rosenblum, Eds., Visualization in Scientific Computing, IEEE Computer Society Press, Los Alamitos, California, 1990.

[Rhodes 91]
M. L. Rhodes, "Computer Graphics in Medicine: The Past Decade," IEEE Computer Graphics and Applications, vol. 11, January 1991, pp. 52-54.

[Seager 90]
M. K. Seager, N. E. Werner, M. E. Zosel, and R. E. Strout, II, "Graphical Analysis of Multi- and Microtasking Execution on Cray Multiprocessors," Visualization in Scientific Computing, IEEE Computer Society Press, Los Alamitos, California, 1990, pp. 160-173.

[Smith 86]
W. A. Smith, Elementary Numerical Analysis, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[Swezey 83]
R. W. Swezey, and E. G. Davis, "A Case Study of Human Factors Guidelines in Computer Graphics," IEEE Computer Graphics and Applications, vol. 3, November 1983, pp. 21-30.

[Upson 89]
C. Upson, T. Faulhaber, D. Kamins et al., "The Application Visualization System: A Computational Environment for Scientific Visualization," IEEE Computer Graphics and Applications, vol. 9, July 1989, pp. 30-42.

[Weiskamp 89]
K. Weiskamp, L. Heiny, and N. Shammas, Power Graphics Using Turbo C, John Wiley & Sons, Inc., New York, New York.

[Wilkinson 65]
J. H. Wilkinson, The Algebraic Eigenvalue Problem, Oxford University Press, Amen House, London, 1965.

[Wu 88]
K. Wu, and L. Hesselink, "Computer Display of Reconstructed 3-D Scalar data," Applied Optics, vol. 27, no. 2, January 1988, pp. 395-404.

[Yakowitz 89]
S. Yakowitz, and F. Szidarovszky, An Introduction to Numerical Computations, Macmillan Publishing Company, New York, New York, 1989.

APPENDIXES

APPENDIX A

A DATA FILE FOR THE EDUCATIONAL VISUALIZATION PROGRAM

The interactive educational visualization program needs a data file for executing "Demo" function. The data file is in the following format:

a b c d   a b c d   a b c d   a b c d ....

where: a, b, c, d are the components of a two by two matrix A

$$A = \begin{vmatrix} a & b \\ c & d \end{vmatrix}$$

```
/***********************************************************/
/*  FILE NAME: matrix.dat                                  */
/*  PURPOSE: A data file for the interactive educational   */
/*           program, named "matrix.c", is needed when the */
/*           visualization program executes the "Demo"     */
/*           function.                                     */
/***********************************************************/

1  1  1  2    -2  1 -1  1    2 -1  1 -1     2 -1  1  1
1  1  1  1     1 -1 -1  1    1 -1 -1 -1    -1 -1 -1 -1
2  1  1  2     1  2  2  1   -2 1  1 -2    -1  2  2 -1
```

APPENDIX B

THE EDUCATIONAL VISUALIZATION PROGRAM SOURCE CODE

```
/**********************************************************/
/*  FILE NAME: matrix.c                                   */
/*  PURPOSE: An interactive program supports a user input  */
/*           query to show matrices in a 2-D space with    */
/*           shading the singular and ill-conditioned      */
/*           matrices, and to display eigenvalues and the  */
/*           corresponding eigenvectors of a selected      */
/*           matrix.  This package can access mouse driver.*/
/**********************************************************/

#include <alloc.h>
#include <conio.h>
#include <dos.h>
#include <float.h>
#include <graphics.h>
#include <math.h>
#include <process.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

#ifdef_TINY_
#error GRAPHICS will not run in the tiny model.
#endif

#define LEFT 0          /* left button used */
#define RIGHT 1         /* right button used */
#define CENTER 2        /* either button used */
#define RESET 0         /* reset mouse */
#define DISPLAY 1       /* display mouse cursor */
#define REMOVE 2        /* hide mouse cursor */
#define STATUS 3        /* get mouse status */
#define COORD 4         /* setup mouse coordinates */
#define HITKEY 5        /* check mouse button pressed */
#define FREEKEY 6       /* check mouse button released */
#define WIDTH 8         /* emulated mouse cursor width */
#define HEIGHT 8        /* emulated mouse cursor height */
#define TRUE 1
#define FALSE 0

int GraphDriver;
int GraphMode;
int ErrorCode=0;
int Ill=0;          /* a flag for an ill matrix found */
int Well=1;         /* a flag for a well matrix found */
/* total digits got when convert a double to character type */
int NDigit=16;
int Initial=0;      /* for OptionProcess() using */
int Flag=0;         /* a flag using in ctod() */
int MaxX,MaxY,HighX,HighY,LowX,LowY;/*the boundary of window*/
int HX1,HX2,HX3,HX4,LX1,LX2,LX3,LX4;/*the boundary of window*/
int HY1,HY2,HY3,HY4,LY1,LY2,LY3,LY4;/*the boundary of window*/
       /* the parameters for drawing figures */
double OrgXLabel,OrgYLabel,FirstXLabel,FirstYLabel;
int OrgX,OrgY,XScale,YScale;
int WindowNo;                           /* total window number */
int ScaleMatrix;                          /* do scale a matrix or
not */
double ConditionNo;                     /* the condition number */
char OrgM[4][20];
      /* store the components of a selected matrix */
char SelectM[4][20];
char SelectUM1[4][20],SelectUM2[4][20];
char SelectUM3[4][20],SelectUM4[4][20];
int EVindex=0;      /* an index of EgVectors array */
int EVF=0;          /* an index of EgVectorF array */
int Efactor=1;      /* a factor for Enlargement() */
int Rfactor=1;      /* a factor for Reduction() */
     /* store the components of eigenvectors */
char VElement[2][20],EgVectors[4][20];
char EgValues[2][20];     /* store eigenvalues */
  /* store double type of components of a selected matrix */
double SelectMF[4];
double SelectMSF[4];      /* store a scaled matrix */
double Equ[4][2];      /* for computing eigenvectors using*/
    /* store the double value of components of eigenvectors */
double EgVectorF[4][2];
void *images[1];    /* memory space to store created images */
     /* store process option box images */
void *voidimage[2],*opimage[2];
struct viewporttype viewp;      /* store old viewport type */
     /* a flag which is TRUE when a mouse exists, otherwise */
int GotMouse;   /* program select the emulated mouse cursor */
int ECx,ECy;     /* x, y positions of emulated mouse cursor */
char *ECursor;    /* store the image of emulated cursor */
char *ECimage;   /* store the background of emulated cursor */
int ECursorShown;/* a flag to indicate emulated cursor shown*/
    /* flags which are true if simulated left or right */
int LHit,RHit;                               /* button pressed */
int LHitNo;      /* count the number of left button pressed */
int RHitNo;     /* count the number of right button pressed */
int LFreeNo;    /* count the number of left button released */
int RFreeNo;   /* count the number of right button released */
FILE *fip;  /* an input file to read data for DemoProcess() */

/**********************************************************/
```

```
/*  FUNCTION NAME: void WindowSetup()                          */
/*  PURPOSE: According to selected window number and the      */
/*           needs of the other functions to setup the        */
/*           boundary of each window shown.                    */
/***********************************************************/
void WindowSetup(windowno,opt)
int windowno,opt;
{
  LowX=0;
  HighX=MaxX;
  LowY=textheight("A")+12;
  HighY=MaxY;
  OrgXLabel=0.0;   /* set x value of the origin */
  OrgYLabel=0.0;   /* set y value of the origin */
  if(windowno==4)  /* set boundary of each window for */
  {                /* total window number 4 */
    LX1=LowX+2;                /* for the first window */
    LY1=LowY+2;
    HX1=(LowX+HighX)/2-2;
    HY1=(LowY+HighY)/2-2;
    LX2=LowX+2;                /* for the second window */
    LY2=(LowY+HighY)/2+2;
    HX2=(LowX+HighX)/2-2;
    HY2=HighY-2;
    LX3=(LowX+HighX)/2+2;      /* for the third window */
    LY3=LowY+2;
    HX3=HighX-2;
    HY3=(LowY+HighY)/2-2;
    LX4=(LowX+HighX)/2+2;      /* for the fourth window */
    LY4=(LowY+HighY)/2+2;
    HX4=HighX-2;
    HY4=HighY-2;
    OrgX=157;        /* x value of a pixel of the origin */
    OrgY=112;        /* y value of a pixel of the origin */
    if(opt==1)       /* for DemoProcess() */
    {
      FirstXLabel=0.25;
      FirstYLabel=0.25;
      XScale=10;
      YScale=10;
    }
  }
  else if(windowno<3)   /* set boundary of each window for */
  {                     /* total window number 1 or 2 */
    LX1=LowX+2;
    LY1=LowY+2;
    XScale=20;
    YScale=20;
    if(windowno==1)         /* for one window */
    {
      HX1=HighX-2;
      HY1=HighY-2;
      OrgX=318;    /* x value of a pixel of the origin */
      OrgY=228;    /* y value of a pixel of the origin */
    }
    else if(windowno==2)   /* for two windows */
    {
      HX1=(HighX-LowX-4)/2;
      HY1=HighY-2;
      LX2=(HighX-LowX-4)/2+2;
      LY2=LowY+2;
      HX2=HighX-2;
      HY2=HighY-2;
      OrgX=157;    /* x value of a pixel of the origin */
      OrgY=228;    /* y value of a pixel of the origin */
    }
  }
  strcpy(OrgM[0],"a");    /* set the content of OrgM[] to */
  strcpy(OrgM[1],"b");    /* be a, b, c and d shown as */
  strcpy(OrgM[2],"c");    /* axes labels */
  strcpy(OrgM[3],"d");
}

/**********************************************************/
/*  FUNCTION NAME: void Initialization()                  */
/*  PURPOSE: Initialize graphic mode, if find error output */
/*           error messages and exit this program.  Setup  */
/*           the boundary of a window shown on screen.     */
/*  LOGIC : Call initgraph() a function supported by turbo */
/*          c graphics package to initialize graphic mode. */
/*          Call graphresult(), getmaxx(), and getmaxy()   */
/*          to detect error and get max x and max y.       */
/**********************************************************/
void Initialization()
{                        /* initialize graphic mode */
  GraphDriver=DETECT;
  initgraph(&GraphDriver,&GraphMode,"");
  ErrorCode=graphresult();
  if(ErrorCode!=grOk) /* find error, output error messages */
  {                   /* then exit the program */
    printf("Graphics System Error:%s\n",
           grapherrormsg(ErrorCode));
    exit(1);
  }
  MaxX=getmaxx();     /* get maximum x */
  MaxY=getmaxy();     /* get maximum y */
  WindowSetup(1,1);   /* call WindowSetup() */
}

/**********************************************************/
/*  FUNCTION NAME: void DrawBar()                         */
/*  PURPOSE: To draw a bar for user menu.                 */
/*  LOGIC: Call bar() a function supported by turbo c     */
/*          graphics package to draw a bar.               */
/**********************************************************/
void DrawBar(lefttx,leftty,rightbx,rightby,color)
int lefttx,leftty,rightbx,rightby,color;
{
  setcolor(WHITE);
  setfillstyle(SOLID_FILL,color);
  bar(lefttx,leftty,rightbx,rightby);  /* call bar() */
}

/***********************************************************/
/*  FUNCTION NAME: void WindowBox()                        */
/*  PURPOSE: To draw a window box for an active window.    */
/*  LOGIC: Call line() a function supported by turbo c     */
/*         graphics package to draw a window box.          */
/***********************************************************/
void WindowBox(startx,starty,endx,endy,color)
int startx,starty,endx,endy,color;
{
  setcolor(color);
  line(startx,starty,startx,endy);
  line(startx,starty,endx,starty);
  line(endx,starty,endx,endy);
  line(endx,endy,startx,endy);
}

/***********************************************************/
/*  FUNCTION NAME: void Menu()                             */
/*  PURPOSE: To draw a user menu bar for this package.     */
/***********************************************************/
void Menu(color1,color2,color3,color4)
int color1,color2,color3,color4;
{                       /* set view port to be full screen */
  setviewport(0,0,MaxX,MaxY,1);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  settextjustify(LEFT_TEXT,TOP_TEXT);
                        /* draw a user menu bar */
  DrawBar(0,0,MaxX,textheight("A")+6,LIGHTGRAY);
  DrawBar(MaxX/5-8,0,2*MaxX/5-4*textwidth("D"),
       6+textheight("D"),color1);
  setcolor(RED);
  outtextxy(MaxX/5,3,"D");
  setcolor(BLACK);
  outtextxy(MaxX/5+textwidth("D"),3,"emo");
  DrawBar(2*MaxX/5-8,0,3*MaxX/5-4*textwidth("D"),
       6+textheight("D"),color2);
  setcolor(RED);
  outtextxy(2*MaxX/5,3,"O");
  setcolor(BLACK);
  outtextxy(2*MaxX/5+textwidth("O"),3,"ptions");
  DrawBar(3*MaxX/5-8,0,4*MaxX/5-4*textwidth("H"),
       6+textheight("H"),color3);
  setcolor(RED);
  outtextxy(3*MaxX/5,3,"H");
  setcolor(BLACK);
  outtextxy(3*MaxX/5+textwidth("H"),3,"elp");
  DrawBar(4*MaxX/5-8,0,MaxX-4*textwidth("Q"),
       6+textheight("Q"),color4);
  setcolor(RED);
  outtextxy(4*MaxX/5,3,"Q");
  setcolor(BLACK);
  outtextxy(4*MaxX/5+textwidth("Q"),3,"uit");
                      /* draw the major window box */
  WindowBox(LowX,LowY,HighX,HighY,WHITE);
}

/*********************************************************/
/*  FUNCTION NAME: void CheckMouse()                     */
/*  PURPOSE: To support communication between mouse driver */
/*           and this educational aid program.           */
/*  LOGIC: Microsoft mouse support predefined mouse      */
/*         functions and those parameters are sent to    */
/*         mouse driver by AX, BX, CX, and DX registers. */
/*         Call int86() to communicate with mouse driver. */
/*********************************************************/
void CheckMouse(ax,bx,cx,dx)
int *ax,*bx,*cx,*dx;
{
  union REGS InRegister,OutRegister;

  InRegister.x.ax=*ax;
  InRegister.x.bx=*bx;
  InRegister.x.cx=*cx;
  InRegister.x.dx=*dx;
  int86(0x33,&InRegister,&OutRegister);
  *ax=OutRegister.x.ax;
  *bx=OutRegister.x.bx;
  *cx=OutRegister.x.cx;
  *dx=OutRegister.x.dx;
}

/*********************************************************/
/*  FUNCTION NAME: void OutputEmulatedCursor()           */
/*  PURPOSE: To show or erase the emulated mouse cursor. */
/*  LOGIC: The viewport settings have to change to the full*/
/*         screen while the emulated mouse cursor is shown */
/*         or erased, so it is necessary to save old view- */
/*         port setting and old x, y position.           */
/*********************************************************/
void OutputEmulatedCursor()
{
  struct viewporttype view;
  int oldx,oldy;

  oldx=getx();                    /* save current position */
  oldy=gety();
  getviewsettings(&view);            /* save view settings */
                        /* set viewport to be full screen */
  setviewport(0,0,MaxX,MaxY,1);
  if(ECursorShown)
  {                              /* remove emulated cursor */
    putimage(ECx,ECy,ECimage,COPY_PUT);
    ECursorShown=FALSE;
  }
  else /* since emulated mouse cursor is not shown, output */
  {    /* the emulated mouse cursor */
    getimage(ECx,ECy,ECx+WIDTH,ECy+HEIGHT,ECimage);
                                /* display emulated cursor */
    putimage(ECx,ECy,ECursor,COPY_PUT);
    ECursorShown=TRUE;
    /* reset viewport settings and its current position */
    setviewport(view.left,view.top,view.right,view.bottom,1);
    moveto(oldx,oldy);
  }
}

/***********************************************************/
/*  FUNCTION NAME: void RemoveMouseCursor()                */
/*  PURPOSE: To remove mouse cursor from the screen.  It is*/
/*           necessary to erase mouse cursor before drawing*/
/*           anything on the screen.                       */
/*  LOGIC: Call CheckMouse() to erase mouse cursor if mouse*/
/*         exist, otherwise call OutputEmulatedCursor().   */
/***********************************************************/
void RemoveMouseCursor()
{
  int ax,bx,cx,dx;

  if(GotMouse)   /* mouse exists */
  {              /* erase mouse cursor */
    ax=REMOVE;
    CheckMouse(&ax,&bx,&cx,&dx);
  }
  else    /* mouse doesn't exist */
    OutputEmulatedCursor();
}

/***********************************************************/
/*  FUNCTION NAME: void DisplayMouseCursor()               */
/*  PURPOSE: To display mouse cursor on the screen.        */
/*  LOGIC: Call CheckMouse() to display mouse cursor if    */
/*         mouse exist, otherwise call                     */
/*         OutputEmulatedCursor().                         */
/***********************************************************/
void DisplayMouseCursor()
{
  int ax,bx,cx,dx;

  if(GotMouse)     /* mouse exists */
  {               /* display mouse cursor */
    ax=DISPLAY;
    CheckMouse(&ax,&bx,&cx,&dx);
  }
  else           /* mouse doesn't exist */
    OutputEmulatedCursor();
}
```

```
/**********************************************************/
/*  FUNCTION NAME: void InitEmulatedCursor()              */
/*  PURPOSE: To create and draw the emulated mouse cursor */
/*           image on the screen.                         */
/*  LOGIC: Allocate memory spaces for storing the emulated */
/*         mouse cursor image and its background image.   */
/**********************************************************/
void InitEmulatedCursor()
{
       /* allocate memories for the emulated mouse cursor */
  ECursor=malloc(imagesize(0,0,WIDTH,HEIGHT));
  ECimage=malloc(imagesize(0,0,WIDTH,HEIGHT));
       /* if memory is not enough, exit the program */
  if((ECursor==NULL)||(ECimage==NULL))
  {
    closegraph();
    printf("Not enought memory for executing the program.\n");
    exit(1);
  }
  getimage(0,0,WIDTH,HEIGHT,ECimage);
      /* draw mouse cursor image */
  DrawBar(0,0,WIDTH,HEIGHT,WHITE);
      /* save the image of the emulated mouse cursor */
  getimage(0,0,WIDTH,HEIGHT,ECursor);
  putimage(0,0,ECimage,COPY_PUT); /* erase mouse image */
}

/**********************************************************/
/*  FUNCTION NAME: int InitMouse()                        */
/*  PURPOSE: To initialize mouse and show the mouse cursor */
/*           at the left top position on the screen.      */
/*  LOGIC: Call this function right after initialize      */
/*         graphic mode. Call CheckMouse() to reset mouse, */
/*         if it is successful then set the mouse cursor at*/
/*         (0,0). If there are a mouse exists, the function*/
/*         returns 1, otherwise returns 0.                */
/**********************************************************/
int InitMouse()
{
  int ax,bx,cx,dx,x,y;

  GotMouse=TRUE;
  ax=RESET;          /* reset mouse */
       /* if mouse reset is successful then ax is 1 */
  CheckMouse(&ax,&bx,&cx,&dx);

  if(ax)    /* mouse reset successful */
  {         /* display mouse cursor at (0,0), and return 1 */
    x=0;
    y=0;
    if(GotMouse)          /* if mouse exists */
    {
      ax=COORD;           /* get mouse coordinates */
      CheckMouse(&ax,&bx,&x,&y);
    }
    else                   /* if mouse doesn't exist */
    {
      RemoveMouseCursor();/* remove current mouse cursor */
      ECx=x;             /* update mouse cursor's location */
      ECy=y;
                   /* display mouse cursor in new location */
      DisplayMouseCursor();
    }
    DisplayMouseCursor();
    return(1);
  }
  else   /* no mouse found */
  {
    GotMouse=FALSE;
    ECx=0;
    ECy=0;
    LHit=FALSE;   /* set button pressed to FALSE */
    RHit=FALSE;
    LHitNo=0; /* set button press and release counters to 0 */
    RHitNo=0;
    LFreeNo=0;
    RFreeNo=0;
                /* create and draw the emulated mouse cursor */
    InitEmulatedCursor();
    ECursorShown=FALSE;
    DisplayMouseCursor();  /* Display emulated mouse cursor */
    return(0);       /* return 0 indicated no mouse exists */
  }
}

/************************************************************/
/*  FUNCTION NAME: void GetMousePosition()                  */
/*  PURPOSE: To get the current mouse cursor position.      */
/*  LOGIC: Call CheckMouse() if mouse exists, otherwise     */
/*         the current mouse position is get from ECx and   */
/*         ECy (the x and y position of emulated mouse      */
/*         cursor).                                         */
/************************************************************/
void GetMousePosition(x,y)
int *x,*y;
{
  int ax,bx;

  if(GotMouse)      /* if mouse exists */
  {
    ax=STATUS;     /* get mouse status */
    CheckMouse(&ax,&bx,x,y);
  }
  else              /* if mouse doesn't exist */
  {
    *x=ECx;
    *y=ECy;
  }
```

```
}

/**********************************************************/
/*  FUNCTION NAME: int KeyBoardHit()                       */
/*  PURPOSE: To get a key from keyboard hit to move the    */
/*           mouse cursor.                                 */
/*  LOGIC: If mouse doesnot exist, the function update the */
/*         status of the emulated mouse cursor and the     */
/*         button pressed flags. If a certain emulated     */
/*         button key is pressed, the function returns -1. */
/*         The function returns 0 if an arrow key is       */
/*         pressed. Otherwise, it returns the pressed key  */
/*         value.                                          */
/**********************************************************/
int KeyBoardHit(key)
int key;
{
  int ch1,ch2,input;

  if(kbhit())  /* if a key of the keyboard is pressed */
  {
    ch1=getch();
    if(ch1==0) /* Handle arrow keys that have two character */
    {      /* and the first one is zero. Combine these two */
      ch2=getch(); /* characters into one character and set */
      ch2=ch2<<8;          /* the gotten value to be input */
      input=ch2|ch1;
    }
    else    /* handle other keys except arrow keys */
      input=ch1;
    RemoveMouseCursor();  /* erase mouse cursor */
    if(input==0x4900) /* 'Page Up' key, emulate right mouse */
    {                 /* key */
      DisplayMouseCursor();
      if(RHit)
        RHit=FALSE;
      else
        RHit=TRUE;
      if((key==RIGHT)||(key==CENTER))
        return(-1);  /* the emulated mouse key is pressed */
      if(RHit)
        RHitNo++;
      else
        RFreeNo--;
      return(0);  /* Wrong emulated mouse key is pressed */
    }
    else if(input==0x5100) /* 'Page Down' key, emulate left */
    {                       /* mouse key */
      DisplayMouseCursor();
      if(LHit)
        LHit=FALSE;
      else
        LHit=TRUE;
      if((key==LEFT)||(key==CENTER))
    return(-1); /* the emulated mouse key is pressed */
  if(LHit)
    LHitNo++;
  else
    LFreeNo++;
  return(0); /* Wrong emulated mouse key is pressed */
}
else if(input==0x4D00) /* 'right' of arrow key, move */
{                      /* the mouse cursor right */
  ECx+=1;          /* increase x */
  if(ECx>MaxX)
    ECx=MaxX;
  /* display emulated mouse cursor at new position */
  DisplayMouseCursor();
}
else if(input==0x4B00) /* 'left' of arrow key, move */
{                      /* the cursor left */
  ECx-=1;          /* decrease x */
  if(ECx<0)
    ECx=0;
  /* display emulated mouse cursor at new position */
  DisplayMouseCursor();
}
else if(input==0x4800) /* 'up' of arrow key, move the */
{                      /* cursor up */
  ECy-=1;          /* decrease y */
  if(ECy<0)
    ECy=0;
  /* display emulated mouse cursor at new position */
  DisplayMouseCursor();
}
else if(input==0x5000) /* 'down' of arrow key, move */
{                     /* cursor down */
  ECy+=1;          /* increase y */
  if(ECy>MaxY)
    ECy=MaxY;
  /* display emulated mouse cursor at new position */
  DisplayMouseCursor();
}
else if(input==0x4700) /* HOME key, move the cursor to */
{                      /* right top corner */
  ECx+=1;       /* increase x */
  if(ECx>MaxX)
    ECx=MaxX;
  ECy-=1;       /* decrease y */
  if(ECy<0)
    ECy=0;
  /* display emulated mouse cursor at new position */
  DisplayMouseCursor();
}
else if(input==0x4F00) /* 'End' key, move cursor to */
{                      /* right bottom corner */
  ECx+=1;         /* increase x */
  if(ECx>MaxX)
```

```
        ECx=MaxX;
      ECy+=1;          /* increase y */
      if(ECy>MaxY)
        ECy=MaxY;
      /* display emulated mouse cursor at new position */
      DisplayMouseCursor();
    }
    else if(input==0x5200)/* 'Insert' key, move the cursor */
    {                      /* to left top corner */
      ECx-=1;          /* decrease x */
      if(ECx<0)
        ECx=0;
      ECy-=1;          /* decrease y */
      if(ECy<0)
        ECy=0;
      /* display emulated mouse cursor at new position */
      DisplayMouseCursor();
    }
    else if(input==0x5300)/* 'Delete' key, move the cursor */
    {                      /* to left bottom corner */
      ECx-=1;
      if(ECx<0)        /* decrease x */
        ECx=0;
      ECy+=1;           /* increase y */
      if(ECy>MaxY)
        ECy=MaxY;
      /* display emulated mouse cursor at new position */
      DisplayMouseCursor();
    }
    else /* display emulated mouse cursor at the same */
    {    /* position */
      DisplayMouseCursor();
      return(input);
    }
  }
  return(0);     /* return 0 */
}

/*********************************************************/
/*  FUNCTION NAME: int IfMouseInWindow()                 */
/*  PURPOSE: To check if the mouse cursor is in a specified*/
/*           window box.                                 */
/*  LOGIC: The function returns TRUE if the mouse cursor is*/
/*         in the specified window box; otherwise returns  */
/*         FALSE.                                        */
/*********************************************************/
int IfMouseInWindow(x,y,left,top,right,bottom)
int x,y,left,top,right,bottom;
{
  if((x>=left)&&(x<=right)&&(y>=top)&&(y<=bottom))
    return(1);
  else
    return(0);
}
```

```
/************************************************************/
/*  FUNCTION NAME: int CheckMouseKey()                      */
/*  PURPOSE: To check the mouse status.                     */
/*  LOGIC: The function returns TRUE if a specific key      */
/*         actually do the specific action; otherwise       */
/*         returns FALSE.                                   */
/************************************************************/
int CheckMouseKey(action,key)
int action,key;
{
  int ax,bx,cx,dx;

  ax=action;
  if((key==LEFT)||(key==CENTER))
  {
    bx=LEFT;
    CheckMouse(&ax,&bx,&cx,&dx);
    if(bx)  /* return TRUE since the action performed */
      return(TRUE);
  }
  if((key==RIGHT)||(key==CENTER))
  {
    ax=action;
    bx=RIGHT;
    CheckMouse(&ax,&bx,&cx,&dx);
    if(bx)  /* return TRUE since the action performed */
      return(TRUE);
  }
  return(FALSE);
}

/************************************************************/
/*  FUNCTION NAME: int FreeKey()                            */
/*  PURPOSE: To check if the mouse key is released.         */
/*  LOGIC: The function returns TRUE if a specific key      */
/*         has been released; otherwise return FALSE.       */
/************************************************************/
int FreeKey(key)
int key;
{
  if(GotMouse)                      /* if mouse exists */
    return(CheckMouseKey(FREEKEY,key));
  else
  {
    if(((key==LEFT)||(key==CENTER))
      &&(LFreeNo))
    {
      LFreeNo--;
      return(TRUE);
    }
    else if(((key==RIGHT)||(key==CENTER))
         &&(RFreeNo))
    {
      RFreeNo--;
      return(TRUE);
    }
    else if(KeyBoardHit(key)<0)
    {
      if((key==LEFT)||(key==CENTER))
        return(TRUE);
      else if((key==RIGHT)||(key==CENTER))
        return(TRUE);
    }
    return(FALSE);
  }
}

/************************************************************/
/*  FUNCTION NAME: int HitKey()                            */
/*  PURPOSE: To check if the mouse key is pressed.         */
/*  LOGIC: The function returns TRUE if a specific key     */
/*         has been pressed; otherwise return FALSE.       */
/************************************************************/
int HitKey(key)
int key;
{
  if(GotMouse)     /* if mouse exists */
    return(CheckMouseKey(HITKEY,key));
  else
  {
    if(((key==LEFT)||(key==CENTER))
      &&(LHitNo))
    {
      LHitNo--;
      return(TRUE);
    }
    else if(((key==RIGHT)||(key==CENTER))
            &&(RHitNo))
    {
      RHitNo--;
      return(TRUE);
    }
    else if(KeyBoardHit(key)<0)
    {
      if((key==LEFT)||(key==CENTER))
        return(TRUE);
      else if((key==RIGHT)||(key==CENTER))
        return(TRUE);
    }
    return(FALSE);
  }
}

/************************************************************/
/*  FUNCTION NAME: int GetInput()                          */
/*  PURPOSE: To get a key from mouse buttons or keyboard.  */
/*  LOGIC: The function returns a key if a key of keyboard */
/*         is pressed, or it returns -1 if the key is      */
/*         got from mouse, otherwise it returns 0.         */
/*********************************************************/
int GetInput(key)
int key;
{
  int input;

  if(!GotMouse)            /* if mouse does not exist */
  {
    input=KeyBoardHit(key);
    return(input);
  }
  if(kbhit())  /* if a key of the keyboard is pressed */
    return(getch());
  else
  {
    if(HitKey(key))
    {
      while(!FreeKey(key))
        ;
      return(-1);
    }
    else if(FreeKey(key))
      return(-1);
    return(0);
  }
}

/*********************************************************/
/*  FUNCTION NAME: int KeepWaitingInput()                */
/*  PURPOSE: To get a key from mouse buttons or keyboard. */
/*  LOGIC: The function keeps to call GetInput() until a  */
/*         key is pressed.                               */
/*********************************************************/
int KeepWaitingInput(key)
int key;
{
  int input;

  do
  {
    input=GetInput(key);
  }while(input==0);
  return(input);
}

/*********************************************************/
/*  FUNCTION NAME: void DoubletoChar()                   */
/*  PURPOSE: To convert a double integer into a character */
/*           type.                                       */
/*  LOGIC: Output of this function is as shorter as      */
/*         possible as the format:                       */
/*            [-][m]m.[d][d][d][d][d][d]E[+/-][x]x       */
/*         example: 10, 1.23, 1.234567, 10.0689,         */
/*                  1.000006E+2                              */
/*         If it is 1.0000057, then output 1.000006.         */
/************************************************************/
void DoubletoChar(value,string)
double value;
char string[];
{
  char *s,temps[20];
  int decpt,sign,flag=0,add=0,i,j,k,index,length;

  if(value==0)
  {
    string[0]='0';
    string[1]='\0';
    return;
  }
  for(i=0;i<20;i++)
    temps[i]=' ';
  s=ecvt(value,NDigit,&decpt,&sign);
  strcpy(temps,s);
  j=0;
  if(sign!=0)
    string[j++]='-';
  if(decpt==1)
    index=decpt;
  else if((decpt>1)||(decpt<=0))
    index=1;
  /* handle a double value is less than 100.00 or the first */
  /* part of output format of a double value which is */
  /* greater or equal to 100.00 */
  for(i=0;i<strlen(temps);i++)
  {
    if(i==index)
      string[j++]='.';
    if(isdigit(temps[i]))
      string[j++]=temps[i];
  }
  string[j]='\0';
  for(i=strlen(string)-1;i>=0;i--)
  {
    if((string[i]=='0')||(string[i]=='.'))
      string[i]='\0';
    else if(string[i]!='0')
      i=-1;
  }
       /* control the first output part to be total 8 digit */
       /* (include the decimal point) */
  length=strlen(string);
  if(((sign==0)&&(length>8))||((sign!=0)&&(length>9)))
  {
    if(sign==0)
      j=8;
    else if(sign!=0)
      j=9;
  if(string[j]>4)
  {
    string[j]='\0';
    j--;
    if(string[j]=='9')
    {
      while((string[j]=='9')&&(j>1))
      {
        string[j]='0';
        j--;
        if(((j==1)&&(sign==0))||((j==2)&&(sign!=0)))
        {
          if(sign==0)
            k=0;
          else if(sign!=0)
            k=1;
          if(string[k]!='9')
            string[k]+=1;
          else
          {
            string[k]='1';
            add=1;     /* add one more digit in */
          }
          j=1; /* exit while loop */
        }
        else if(string[j]!='9')
        {
          string[j]+=1;
          j=1; /* exit while loop */
        }
      }
    }
    else
      string[j]='\0';
  }
  else
    string[j]='\0';
}
for(i=strlen(string)-1;i>=0;i--)
{
  if((string[i]=='0')||(string[i]=='.'))
    string[i]='\0';
  else if(string[i]!='0')
    i=-1;
}
/* handle the second part of output format, following by */
/*  a 'E', of a double value which is greater or equal */
/* 100.00 */
if((decpt>1)||(decpt<=0))
{
  if(add==1)
    value=decpt;
  else
    value=decpt-1;
    if(value==0)
      return;
    j=strlen(string);
    string[j++]='E';
            /* if the exponent is 10, or 20 ... */
    if(fmod(value,10.0)==0)
      flag=3;
    s=ecvt(value,3,&decpt,&sign);
    strcpy(temps,s);
    if(sign!=0)
      string[j++]='-';
    else
      string[j++]='+';
    i=0;
    while(temps[i]!='0')
      string[j++]=temps[i++];
    if(flag==3)
      string[j++]=temps[i++];
    string[j]='\0';
  }
}

/**********************************************************/
/*  FUNCTION NAME: void ctod()                            */
/*  PURPOSE: To convert a character string into a double  */
/*           integer.                                     */
/*  LOGIC: Output of this function is as shorter as       */
/*         possible as the format:                        */
/*             [-][m]m.[d][d][d][d][d][d]E[+/-][x]x       */
/*         example: 10, 1.23, 1.234567, 10.0689,          */
/*                  1.000006E+2                           */
/*         If it is 1.0000057, then output 1.000006.      */
/**********************************************************/
void ctod(array,starti,endi,dvalue)
char array[];
int starti,endi;
double *dvalue;
{
  int i,TempI,sign=0;
  double value=0.0,TempValue=0.0;

  for(i=starti;i<endi;i++)
  {
    if(array[i]=='-')    /* an minus value */
      sign=1;
    else if((array[i]!=' ')&&(array[i]!='/')
    &&(array[i]!='.'))
      value=value*10+(array[i]-'0');/*convert to be integer*/
    else if(array[i] == '/')  /* got a fraction */
    {
      Flag=1;
      TempValue=value;
      value=0;
    }
    else if(array[i]=='.')  /* got a decimal point */
    {
      Flag=2;
      TempI=i;
      TempValue=value;
      value=0;
    }
    else
    {
      if(Flag==1)  /* indicate that got a fraction */
      {
        value=TempValue/value;
        Flag=0;
      }
      else if(Flag==2) /* indicate that got a decimal point */
      {
        value=TempValue+value/pow10(i-TempI-1);
        Flag=0;
      }
      value=0;
    }
  }
  if(Flag==1)  /* a fraction */
  {
    value=TempValue/value;
    Flag=0;
  }
  else if(Flag==2) /* a decimal point */
  {
    value=TempValue+value/pow10(i-TempI-1);
    Flag=0;
  }
  if(sign==1)/* indicate original string is an minus value */
    *dvalue=value*(-1);   /* store the final got double */
  else
    *dvalue=value;
}

/**********************************************************/
/*  FUNCTION NAME: void ChartoDouble()                    */
/*  PURPOSE: To transfer the character type of the content */
/*           of an array to be the corresponding integer  */
/*           values.                                      */
/*  LOGIC : Transfers an character to an integer.         */
/**********************************************************/
void ChartoDouble(array,DValue)
char array[];
double *DValue;
{
  int i,power=0,E;
  double temp1,temp2;

  for(i=0;i<strlen(array);i++)
  {
    if((array[i]=='e')||(array[i]=='E'))
    {                /* if got a scientific number */
      E=i;
      power=1;    /* there are power part */
    }
  }
  if(power==0) /* handle a number that is not a scientific */
  {            /* number */
    ctod(array,0,strlen(array),&temp1);
    *DValue=temp1;
  }
  else if(power==1)  /* handle a scientific number */
  {
    ctod(array,0,E,&temp1);
    ctod(array,E+1,strlen(array),&temp2);
    if(temp2<0)
    {
      temp2*=(-1);
      *DValue=temp1/pow(10.0,temp2);
    }
    else if(temp2>0)
      *DValue=temp1*pow(10.0,temp2);

    else if(temp2==0)
      *DValue=temp1;   /* store the final got value */
  }
}

/**********************************************************/
/*  FUNCTION NAME: void Simplified()                      */
/*  PURPOSE: To simplify three double integers that are   */
/*           divided by their common factors.             */
/*  LOGIC : Set these three double integers to be positive.*/
/*          Keep to check if these three double integers  */
/*          can be divided by their common factors their  */
/*          range is from 1 to 19.                        */
/**********************************************************/
void Simplified(a,b,c)
double *a,*b,*c;
{
  double j;
  int Asign=0,Bsign=0,Csign=0,i;

  if(*a<0)   /* if a is less than 0, set it to be positive */
  {
    (*a)*=(-1);
    Asign=1;
  }
  if(*b<0) /* if b is less than 0, set it to be positive */
  {
    (*b)*=(-1);
    Bsign=1;
  }
  if(*c==0)
```

```
{  /* keep to check if these three double integers can be */
    /* divided by their common factors their range is from */
   for(i=1;i<20;i++)                                /* 1 to 19 */
   {
     j=1.0;   /* reset j */
     j*=i;
         /* check if a and b can be divided by j */
     if((fmod(*a,j)==0)&&(fmod(*b,j)==0))
     {
       (*a)/=j;
       (*b)/=j;
     }
   }
   if((Asign==1)&&(Bsign==1))
     ;
   else if(Asign==1)
     (*a)*=(-1);      /* reset the sign of a */
   else if(Bsign==1)
     (*b)*=(-1);      /* reset the sign of b */
}
else if(*c!=0)
{
   if(*c<0) /* if c is less than 0, set it to be positive */
   {
     (*c)*=(-1);
     Csign=1;
   }
    /* keep to check if these three double integers can be */
    /* divided by their common factors their range is from */
   for(i=1;i<20;i++)                                /* 1 to 19 */
   {
     j=1.0;
     j*=i;
     if((fmod(*a,j)==0)&&(fmod(*b,j)==0)&&(fmod(*c,j)==0))
     {
       (*a)/=j;
       (*b)/=j;
       (*c)/=j;
     }
   }
   if((Asign==1)&&(Bsign==1)&&(Csign==1))
     ;
   else if((Asign==1)&&(Bsign==1))
   {
     (*a)*=(-1);
     (*b)*=(-1);
   }
   else if((Bsign==1)&&(Csign==1))
   {
     (*b)*=(-1);
     (*c)*=(-1);
   }
   else if((Asign==1)&&(Csign==1))
   {
        (*a)*=(-1);
        (*c)*=(-1);
      }
      else if(Asign==1)
        (*a)*=(-1);
      else if(Bsign==1)
        (*b)*=(-1);
      else if(Csign==1)
        (*c)*=(-1);                  /* reset the sign of c */
    }
}

/*******************************************************/
/*  FUNCTION NAME: void EgVector()                     */
/*  PURPOSE: To solve eigenvectors corresponding to a  */
/*           specified eigenvalue.                     */
/*  LOGIC : Separate to handle real part and imaginary part*/
/*          of a eigenvalue.  Solve eigenvectors by solving*/
/*          the equations:  (egv: eigenvalue)          */
/*                  (a-egv)x1+(b)x2=0                  */
/*                  (c)x1+(d-egv)x2=0                  */
/*          After got eigenvectors, call Simplified() to */
/*          simplify the components of the eigenvectors.  */
/*******************************************************/
void EgVector(egvalue,e,f,sign,index)
double egvalue,e,f;
int sign,index;
{
  char string[20];
  double temp=0,egvt11,egvt12;
  int i,j;

  if(sign==0)  /* handle real number */
  {                       /* apply equations */
    Equ[0][0]=SelectMF[0]-egvalue;
    Equ[1][0]=SelectMF[1];
    Equ[2][0]=SelectMF[2];
    Equ[3][0]=SelectMF[3]-egvalue;
    if((Equ[0][0]==0)&&(Equ[1][0]==0))
    {
      if((Equ[3][0]!=0)&&(Equ[2][0]!=0))
      {
        egvt11=Equ[3][0]*(-1);   /* got a real eigenvector */
        egvt12=Equ[2][0];  /* got another real eigenvector */
      }
      else if((Equ[3][0]!=0)&&(Equ[2][0]==0))
      {
        egvt11=1; /* got a real eigenvector */
        egvt12=0; /* got another real eigenvector */
      }
      else if((Equ[3][0]==0)&&(Equ[2][0]!=0))
      {
        egvt11=0; /* got a real eigenvector */
        egvt12=1; /* got another real eigenvector */
      }
      else
      {
        egvt11=1; /* got a real eigenvector */
        egvt12=1; /* got another real eigenvector */
      }
    }
    else if((Equ[0][0]!=0)&&(Equ[1][0]==0))
    {
      egvt11=0;
      egvt12=1;
    }
    else
    {
      egvt11=Equ[1][0]*(-1);
      egvt12=Equ[0][0];
    }
    Simplified(&egvt11,&egvt12,&temp);
    DoubletoChar(egvt11,string);
    EgVectorF[EVF][0]=egvt11; /* store the element */
    EgVectorF[EVF++][1]=0;     /* value of eigenvectors */
    strcpy(VElement[0],string);
    DoubletoChar(egvt12,string);
    EgVectorF[EVF][0]=egvt12; /* store the element */
    EgVectorF[EVF++][1]=0;     /* value of eigenvectors */
    strcpy(VElement[1],string);
    strcpy(EgVectors[EVindex++],VElement[0]);
    strcpy(EgVectors[EVindex++],VElement[1]);
  }
  else if(sign==1)  /* handle complex number */
  {
    Equ[0][0]=SelectMF[0]-e;
    Equ[1][0]=SelectMF[1];
    Equ[2][0]=SelectMF[2];
    Equ[3][0]=SelectMF[3]-e;
    if(index==1)    /* handle first got eigenvalue */
    {
      Equ[0][1]=f*(-1);
      Equ[3][1]=f*(-1);
    }
    else if(index==2)  /* handle another eigenvalue */
    {
      Equ[0][1]=f;
      Equ[3][1]=f;
    }
    Equ[1][1]=Equ[1][0]*Equ[0][1]*(-1);
    Equ[1][0]=Equ[1][0]*Equ[0][0];
    Equ[0][0]=Equ[0][0]*Equ[0][0]+Equ[0][1]*Equ[0][1];
    Equ[0][1]=Equ[0][0]*((-1)*Equ[0][1]+Equ[0][1]);
    if(Equ[0][1]==0)
    {
      Equ[0][0]*=(-1);
      Simplified(&Equ[0][0],&Equ[1][0],&Equ[1][1]);
      DoubletoChar(Equ[1][0],string);
      /* store the real value of components of a eigenvector */
       EgVectorF[EVF][0]=Equ[1][0];
       strcpy(VElement[0],string);
       if(Equ[1][1]>0)
       {
         DoubletoChar(Equ[1][1],string);
      /* store the real value of components of a eigenvector */
         EgVectorF[EVF++][1]=Equ[1][1];
         strcat(VElement[0],"+");
         strcat(VElement[0],string);
         strcat(VElement[0],"i");
       }
       else if(Equ[1][1]<0)
       {
         DoubletoChar(Equ[1][1],string);
     /* store the real value of components of a eigenvector */
         EgVectorF[EVF++][1]=Equ[1][1];
         strcat(VElement[0],string);
         strcat(VElement[0],"i");
       }
       else if(Equ[1][1]==0)
         EgVectorF[EVF++][1]=0;
       DoubletoChar(Equ[0][0],string);
      /* store the real value of components of a eigenvector */
       EgVectorF[EVF][0]=Equ[0][0];
       EgVectorF[EVF++][1]=0;
       strcpy(VElement[1],string);
          /* store the components of a eigenvector */
       strcpy(EgVectors[EVindex++],VElement[0]);
       strcpy(EgVectors[EVindex++],VElement[1]);
     }
   }
}

/*******************************************************/
/*  FUNCTION NAME: void EgValue()                      */
/*  PURPOSE: To solve eigenvalues corresponding to a   */
/*            selected matrix.                         */
/*  LOGIC : Separate to handle real part and imaginary part*/
/*          of a eigenvalue.  Solve eigenvalues by solving */
/*          the equations:  (egv: eigenvalue)          */
/*                  |(a-egv)    b| = 0                 */
/*                  | c   (d-egv)|                     */
/*          After got eigenvectors, call EgVector() to */
/*          get the corresponding eigenvectors.        */
/*******************************************************/
void EigenValue()
{
  int sign=0;
  double a,b,c,d,e,f,egv1,egv2;
  char string[50];

  c=SelectMF[0]*SelectMF[3]-SelectMF[1]*SelectMF[2];
  b=SelectMF[0]*(-1)+SelectMF[3]*(-1);
```

```
    a=1.0;
    d=pow(b,2.0)-4.0*a*c;
    if(d<0.0)  /* got a complex number */
    {
      sign=1;
      d=d*(-1);
    }
    e=b*(-1)/(2*a);
    f=sqrt(d)/(2*a);
    if(sign==0)    /* handle real number */
    {
      egv1=e+f;    /* get eigenvalues */
      egv2=e-f;
      DoubletoChar(egv1,EgValues[0]);
      DoubletoChar(egv2,EgValues[1]);
        /* to solve the corresponding eigenvectors */
      EgVector(egv1,e,f,sign,0);
      EgVector(egv2,e,f,sign,0);
    }
    else if(sign==1)   /* handle complex number */
    {
      if(e!=0)
      {
        DoubletoChar(e,EgValues[0]);
        strcat(EgValues[0],"+");
        DoubletoChar(e,EgValues[1]);
        strcat(EgValues[1],"-");
      }
      else
        strcpy(EgValues[1],"-");
      DoubletoChar(f,string);
      strcat(string,"i");
      if(e!=0)
        strcat(EgValues[0],string);
      else
        strcpy(EgValues[0],string);
      DoubletoChar(f,string);
      strcat(string,"i");
      strcat(EgValues[1],string);
         /* to solve the corresponding eigenvectors */
      EgVector(0.0,e,f,sign,1);
      EgVector(0.0,e,f,sign,2);
    }
    EVindex=0;  /* reset index flags that are used for */
    EVF=0;      /* EgVectors[] and EgVectorF[] */
  }

  /***********************************************************/
  /*  FUNCTION NAME: void DrawScale()                        */
  /*  PURPOSE: To draw the scale of x and y axes and output  */
  /*           the scale label.                              */
  /*  LOGIC : According to the position of the origin,       */
  /*          pixels numbers per scale and the original label*/
  /*          and the first scale label, we can calculate    */
```

```
/*          the screen position for each scale of x and y  */
/*          axes, and output scale labels per 4 scale      */
/*          mark on screen.                                */
/***********************************************************/
void DrawScale(scale,count0,count1,startp,endp,orgxy,flag)
int scale;
double count0,count1;
int startp,endp,orgxy,flag;
{
  double xcount, ycount;
  int label=0,section, subtotal, drawscale=scale;
  static char neg[]="-";
  char string[20],temps[20];

        /* calculate the total sections of the axes */
  if(endp>startp)
    subtotal=endp-startp;
  else
    subtotal=startp-endp;
  section=subtotal/scale;
  if(flag==1) /* draw scales and output labels of x axes */
  {
    xcount=count0;
    line(startp,orgxy-2,startp,orgxy+2);
    label++;
    xcount=xcount+(count1-count0);
    while(section>0)
    {  /* draw the scale of the positive part of x axes */
      if(endp>startp)
      {
        line(startp+drawscale,orgxy-2,startp+drawscale,
             orgxy+2);
        DoubletoChar(xcount,string);
      }
      /* draw the scale of the negative part of x axes */
      else if(endp<startp)
      {
        line(startp-drawscale,orgxy-2,startp-drawscale,
             orgxy+2);
        DoubletoChar(xcount,temps);
        strcpy(string,neg);
        strcat(string,temps);
      }
      if((label%4)==0)   /* output the label of x scale */
      {
        if(endp>startp)  /* positive part */
        {
          circle(startp+drawscale,orgxy,2);
          outtextxy(startp+drawscale,orgxy+6,string);
        }
        else if(endp<startp) /* negative part */
        {
          circle(startp-drawscale,orgxy,2);
          outtextxy(startp-drawscale-2,orgxy+6,string);
        }
      }
      label++;
      xcount=xcount+(count1-count0);
      section--;
      drawscale=drawscale+scale;
    }
  }
  else if(flag==2)
  {      /* draw scales and output labels of y axes */
    ycount=count0;
    line(orgxy-2,startp,orgxy+2,startp);
    label++;
    ycount=ycount+(count1-count0);
    while(section>0)
    { /* draw the scale of the positive part of y axes */
      if(endp<startp)
      {
        line(orgxy-2,startp-drawscale,orgxy+2,
             startp-drawscale);
        DoubletoChar(ycount,string);
      }
      /* draw the scale of the negative part of y axes */
      else if(endp>startp)
      {
        line(orgxy-2,startp+drawscale,orgxy+2,
             startp+drawscale);
        DoubletoChar(ycount,temps);
        strcpy(string,neg);
        strcat(string,temps);
      }
      if((label%4)==0)  /* output the label of y scale */
      {
        if(endp<startp) /* the positive part */
        {
          circle(orgxy,startp-drawscale,2);
          outtextxy(orgxy-4,startp-drawscale,string);
        }
        else if(endp>startp) /* the negative part */
        {
          circle(orgxy,startp+drawscale,2);
          outtextxy(orgxy-4,startp+drawscale,string);
        }
      }
      label++;
      ycount=ycount+(count1-count0);
      section--;
      drawscale=drawscale+scale;
    }
  }
}

/************************************************************/
/*  FUNCTION NAME: void DrawXAxis()                         */
/*  PURPOSE: To draw the axis line, the scales and the     */
/*           scale labels of x axis.                        */
/*  LOGIC : Draw x axis by calling line(), then call       */
/*          DrawScale() to mark scales and those labels of */
/*          x axis.                                         */
/**********************************************************/
void DrawXAxis(x1,y1,x2,y2,flag)
int x1,y1,x2,y2,flag;
{
  line(x1,y1,x2,y2);              /* draw x axis */
  line(x2-4,y2-4,x2,y2);
  line(x2-4,y2+4,x2,y2);
  if(flag==1)
    outtextxy(x2+2,y2,OrgM[0]);
  else if(flag==2)
    outtextxy(x2+2,y2,OrgM[1]);
  /* positive part */
  DrawScale(XScale,OrgXLabel,FirstXLabel,OrgX,x2,OrgY,1);
  /* negative part */
  DrawScale(XScale,OrgXLabel,FirstXLabel,OrgX,x1,OrgY,1);
}

/**********************************************************/
/*  FUNCTION NAME: void DrawYAxis()                         */
/*  PURPOSE: To draw the axis line, the scales and the     */
/*           scale labels of y axis.                        */
/*  LOGIC : Draw y axis by calling line(), then call       */
/*          DrawScale() to mark scales and those labels of */
/*          y axies.                                        */
/**********************************************************/
void DrawYAxis(x1,y1,x2,y2,flag)
int x1,y1,x2,y2,flag;
{
  line(x1,y1,x2,y2);              /* draw y axis */
  line(x1,y1,x1-4,y1+4);
  line(x1,y1,x1+4,y1+4);
  if(flag==1)
    outtextxy(x1,y1-8,OrgM[3]);
  if(flag==2)
    outtextxy(x1,y1-8,OrgM[2]);
  settextjustify(RIGHT_TEXT,TOP_TEXT);
  /* negative part */
  DrawScale(YScale,OrgYLabel,FirstYLabel,OrgY,y2,OrgX,2);
  /* positive part */
  DrawScale(YScale,OrgYLabel,FirstYLabel,OrgY,y1,OrgX,2);
}

/**********************************************************/
/*  FUNCTION NAME: void DrawAxes()                          */
/*  PURPOSE: To draw the axes line, the scales and the     */
/*           scale labels of x and y axes.                  */
/*  LOGIC : Draw x and y axes by calling DrawXAxis() and   */
/*          DrawYAxis().                                    */
/**********************************************************/
```

```
void DrawAxes(windowno,color,flag)
int windowno,color,flag;
{
  setcolor(color);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  settextjustify(CENTER_TEXT,TOP_TEXT);
  if(windowno==1)
  {
    setviewport(LX1,LY1,HX1,HY1,1);
    DrawXAxis(4,228,628,228,flag);        /* draw x axis */
    DrawYAxis(318,12,318,460,flag);       /* draw y axis */
  }
  else if(windowno==2)
  {
    DrawXAxis(4,228,306,228,flag);        /* draw x axis */
    DrawYAxis(157,12,157,460,flag);       /* draw y axis */
  }
  else if(windowno==4)
  {
    DrawXAxis(4,112,306,112,flag);        /* draw x axis */
    DrawYAxis(157,12,157,220,flag);       /* draw y axis */
  }
}

/**********************************************************/
/*  FUNCTION NAME: void WriteNote()                       */
/*  PURPOSE: To write the figure information and eigenvales*/
/*           and corresponding eigenvectors on            */
/*           corresponding figure.                        */
/**********************************************************/
void WriteNote(opt,figure,B,C,AA,BB,CC,DD,string)
int opt,figure;
char B[],C[],AA[],BB[],CC[],DD[],string[];
{
  int length,h,i,j;

  setcolor(WHITE);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  settextjustify(LEFT_TEXT,TOP_TEXT);
  outtextxy(4,2,"A=|a b|");    /* write matrix format */
  outtextxy(4+2*textwidth("A"),10,"|c d|");
  setcolor(LIGHTGREEN);
  outtextxy(4,20,"a:");       /* write content of matrix */
  outtextxy(4+2*textwidth("A"),20,AA);
  outtextxy(4,30,"b:");
  outtextxy(4+2*textwidth("A"),30,BB);
  outtextxy(4,40,"c:");
  outtextxy(4+2*textwidth("A"),40,CC);
  outtextxy(4,50,"d:");
  outtextxy(4+2*textwidth("A"),50,DD);
  setcolor(YELLOW);
  outtextxy(4,60,string);
  /* write eigenvalues of a selected matrix */
  setcolor(WHITE);
```

```
outtextxy(4,70,"Eigenvalues:");
setcolor(LIGHTMAGENTA);
outtextxy(4,80,EgValues[0]);
outtextxy(4,90,EgValues[1]);
if(figure<5)
  h=30;
else
  h=0;
setcolor(WHITE);
/* write eigenvectors of a selected matrix */
outtextxy(4,100+h,"Eigenvectors:");
setcolor(LIGHTMAGENTA);
if(strlen(EgVectors[0])>strlen(EgVectors[1]))
  length=strlen(EgVectors[0]);
else
  length=strlen(EgVectors[1]);
outtextxy(4,110+h,"m|");
outtextxy(4+2*textwidth("|"),110+h,EgVectors[0]);
outtextxy(4+(length+2)*textwidth("|"),110+h,"|");
outtextxy(4,118+h," |");
outtextxy(4+2*textwidth("|"),118+h,EgVectors[1]);
outtextxy(4+(length+2)*textwidth("|"),118+h,"|");
if(strlen(EgVectors[2])>strlen(EgVectors[3]))
  length=strlen(EgVectors[2]);
else
  length=strlen(EgVectors[3]);
outtextxy(4,130+h,"m|");
outtextxy(4+2*textwidth("|"),130+h,EgVectors[2]);
outtextxy(4+(length+2)*textwidth("|"),130+h,"|");
outtextxy(4,138+h," |");
outtextxy(4+2*textwidth("|"),138+h,EgVectors[3]);
outtextxy(4+(length+2)*textwidth("|"),138+h,"|");
outtextxy(4,148+h,"m is nonzero");
setcolor(WHITE);
/* mark the figure information */
outtextxy(4,160+h,"Figure for:");
setcolor(YELLOW);
outtextxy(4,170+h,B);
outtextxy(4,180+h,C);
/* for DemoProcess() and for Setupvariables() */
if(((opt==1)&&(figure<5))||((opt==4)&&(figure<9)))
{
  if(figure<5)
  {
    i=0;
    j=0;
  }
  else if(figure<7)
  {
    i=270;
    j=210;
  }
  else if(figure<9)
  {
      i=0;
      j=210;
    }
    outtextxy(240+i,170+j,"Selected:");
    setcolor(LIGHTGREEN);
    outtextxy(240+i,180+j,"a="); /* write content of matrix */
    outtextxy(240+2*textwidth("A")+i,180+j,SelectM[0]);
    outtextxy(240+i,190+j,"b=");
    outtextxy(240+2*textwidth("A")+i,190+j,SelectM[1]);
    outtextxy(240+i,200+j,"c=");
    outtextxy(240+2*textwidth("A")+i,200+j,SelectM[2]);
    outtextxy(240+i,210+j,"d=");
    outtextxy(240+2*textwidth("A")+i,210+j,SelectM[3]);
  }     /* for Enlargement() and Reduction() */
  else if((opt==2)&&((figure==1)||(figure==2)))
    outtextxy(220,2,"Old figure");
  else if((opt==3)&&((figure==1)||(figure==2)))
    outtextxy(220,2,"Old figure");
  else if((opt==2)&&((figure>=3)||(figure<=8)))
    outtextxy(180,2,"Enlarged figure");
  else if((opt==3)&&((figure>=3)||(figure<=8)))
    outtextxy(188,2,"Reduced figure");
}

/*********************************************************/
/*  FUNCTION NAME: void DrawSingular()                   */
/*  PURPOSE: To mark the pixels that represent singular  */
/*           matrice in a 2-D space that is a hyperbola. */
/*  LOGIC: Check from the minimum x pixel value to maximum */
/*         x pixel value to calculate the corresponding  */
/*         world value a, then b+c is divided by a to get */
/*         the world value d, then calculate to get the  */
/*         closely corresponding y pixel of d, call      */
/*         call putpixel() to mark it as WHITE color.    */
/*********************************************************/
void DrawSingular(b,c,fx,fy,orgx,orgy,xscale,yscale,oxlabel,
                  fxlabel,oylabel,fylabel)
double b,c;
int fx,fy,orgx,orgy,xscale,yscale;
double oxlabel,fxlabel,oylabel,fylabel;
{
  double a,a1,a2,d,d1,d2,roundoff1,roundoff2;
  int x,x1,x2,y,y1,y2,tempx,tempy;

  for(x=4;x<fx;x++)  /* check the whole x pixels */
  {
    tempx=x;
    a=(((double)(tempx-orgx))/xscale)*(fxlabel-oxlabel);
    if(a!=0.0)
    {
      d=(double)(b*c/a);/* get corresponding y1 pixel of d */
      y1=RealPtoScrP(a,d,orgx,orgy,yscale,oylabel,fylabel,2);
      d1=(((double)(orgy-y1))/yscale)*(fylabel-oylabel);
      y2=y1+1;
      d2=(((double)(orgy-y2))/yscale)*(fylabel-oylabel);
      if(b*c>a*d1)      /* set roundoff1 to be positive */
        roundoff1=b*c-a*d1;
      else
        roundoff1=a*d1-b*c;
      if(b*c>a*d2)       /* set roundoff2 to be positive */
        roundoff2=b*c-a*d2;
      else
        roundoff2=a*d2-b*c;
      if(roundoff1<roundoff2) /* choose a closer pixel to */
      {                      /* represent a singular matirx */
        if((y1>11)&&(y1<=fy))
          putpixel(x,y1,WHITE);
      }
      else
      {
        if((y2>11)&&(y2<=fy))
          putpixel(x,y2,WHITE);
      }
    }
  }
  for(y=12;y<fy;y++)  /* check the whole y pixels */
  {
    tempy=y;
    d=(((double)(orgy-tempy))/yscale)*(fylabel-oylabel);
    if(d!=0.0)
    {
      a=(double)(b*c/d);
      x1=RealPtoScrP(a,d,orgx,orgy,xscale,oxlabel,fxlabel,1);
      a1=(((double)(x1-orgx))/xscale)*(fxlabel-oxlabel);
      x2=x1+1;
      a2=(((double)(x2-orgx))/xscale)*(fxlabel-oxlabel);
      if(b*c>a1*d)         /* set roundoff1 to be positive */
        roundoff1=b*c-a1*d;
      else
        roundoff1=a1*d-b*c;
      if(b*c>a2*d)         /* set roundoff2 to be positive */
        roundoff2=b*c-a2*d;
      else
        roundoff2=a2*d-b*c;
      if(roundoff1<roundoff2)  /* choose a closer pixel to */
      {                       /* represent a singular matrix */
        if((x1>3)&&(x1<=fx))
          putpixel(x1,y,WHITE);
      }
      else
      {
        if((x2>3)&&(x2<=fx))
          putpixel(x2,y,WHITE);
      }
    }
  }
}
```

```
/*******************************************************/
/*  FUNCTION NAME: void MarkSelectPoint()              */
/*  PURPOSE: To mark the pixels that represent the selected*/
/*           matrix, that is set by user input, in a 2-D  */
/*           space.                                    */
/*  LOGIC: Calculate the corresponding pixel for a selected*/
/*         matrix and call putpixel() to mark it as WHITE  */
/*         color and call circle() to draw a LIGHTGREEN    */
/*         circle around that pixel.                   */
/*******************************************************/
void MarkSelectPoint(a,d,orgx,orgy,xscale,yscale,oxlabel,
                     fxlabel,oylabel,fylabel,x,y)
double a,d;
int orgx,orgy,xscale,yscale;
double oxlabel,fxlabel,oylabel,fylabel;
int *x,*y;
{
  *x=RealPtoScrP(a,d,orgx,orgy,xscale,oxlabel,fxlabel,1);
  *y=RealPtoScrP(a,d,orgx,orgy,yscale,oylabel,fylabel,2);
  putpixel(*x,*y,WHITE);
  setcolor(LIGHTGREEN);
  circle(*x,*y,2);
}

/*******************************************************/
/*  FUNCTION NAME: void CalcDrawEigenV()               */
/*  PURPOSE: To draw eigenvectors of a selected matrix on */
/*           screen.                                   */
/*  LOGIC: Calculate the corresponding position of x and y */
/*         value of a eigenvector and show it on screen by */
/*         parameter color.                            */
/*******************************************************/
void CalcDrawEigenV(x,y,color,xscale,yscale,oxlabel,
                    fxlabel,oylabel,fylabel,Rx,Ry)
int x,y,color,xscale,yscale;
double oxlabel,fxlabel,oylabel,fylabel,Rx,Ry;
{
  int x1,y1;

  x1=RealPtoScrP(Rx,Ry,x,y,xscale,oxlabel,fxlabel,1);
  y1=RealPtoScrP(Rx,Ry,x,y,yscale,oylabel,fylabel,2);
  setcolor(color);
  line(x,y,x1,y1); /* draw eigenvector */
}

/*******************************************************/
/*  FUNCTION NAME: void DrawEigenvectors()             */
/*  PURPOSE: To draw eigenvectors of a selected matrix on */
/*           screen.                                   */
/*  LOGIC: Calculate the corresponding position of x and y */
/*         value of a eigenvector and show it on screen by */
/*         parameter color by call CalcDrawEigenV(). The   */
/*         first eigenvalue is in LIGHTMAGENTA color, the  */
/*         second is in LIGHTGREEN color. The imaginary    */
/*        part: the first one is in LIGHTGRAY color, the  */
/*        second is in BROWN color. The x and y values of */
/*        a selected point, get from MarkSelectPoint().   */
/**********************************************************/
void DrawEigenvectors(x,y,xscale,yscale,oxlabel,
                      fxlabel,oylabel,fylabel)
int x,y,xscale,yscale;
double oxlabel,fxlabel,oylabel,fylabel;
{
  double Rx1,Ry1,Rx2,Ry2,Ix1,Iy1,Ix2,Iy2;

  /* draw eigenvectors */
  Rx1=EgVectorF[0][0];  /* vector 1 */
  Ry1=EgVectorF[1][0];
  Rx2=EgVectorF[2][0];  /* vector 2 */
  Ry2=EgVectorF[3][0];
  /* draw first eigenvector of real part */
  CalcDrawEigenV(x,y,LIGHTMAGENTA,xscale,yscale,oxlabel,
                 fxlabel,oylabel,fylabel,Rx1,Ry1);
  /* draw second eigenvector of real part */
  CalcDrawEigenV(x,y,LIGHTGREEN,xscale,yscale,oxlabel,
                 fxlabel,oylabel,fylabel,Rx2,Ry2);
  Ix1=EgVectorF[0][1];
  Iy1=EgVectorF[1][1];
  Ix2=EgVectorF[2][1];
  Iy2=EgVectorF[3][1];
  if((Ix1!=0)||(Iy1!=0))
    /* draw first eigenvector of complex part */
    CalcDrawEigenV(x,y,LIGHTGRAY,xscale,yscale,oxlabel,
                   fxlabel,oylabel,fylabel,Ix1,Iy1);
  if((Ix2!=0)||(Iy2!=0))
    /* draw second eigenvector of complex part */
    CalcDrawEigenV(x,y,BROWN,xscale,yscale,oxlabel,
                   fxlabel,oylabel,fylabel,Ix2,Iy2);
}

/**********************************************************/
/*  FUNCTION NAME: void Inverse()                         */
/*  PURPOSE: To get the inverse of a matrix.              */
/*  LOGIC: According to a known matrix, to calculate its  */
/*         inverse matrix.                                */
/**********************************************************/
void Inverse(ai,bi,ci,di,a,d,b,c)
double *ai,*bi,*ci,*di,a,d,b,c;
{
  *ai=d/(a*d-b*c);
  *bi=(b*(-1))/(a*d-b*c);
  *ci=(c*(-1))/(a*d-b*c);
  *di=a/(a*d-b*c);
}

/**********************************************************/
/*  FUNCTION NAME: void AbsoluteDouble()                  */
/*  PURPOSE: To get the absolute value of a double integer.*/
```

```
/**********************************************************/
void AbsoluteDouble(dint,abs)
double dint,*abs;
{
  *abs=dint;
  if(*abs<0)
    (*abs)*=(-1);
}

/**********************************************************/
/*  FUNCTION NAME: int Condition()                         */
/*  PURPOSE: To get the conditioning of a matrix.          */
/*  LOGIC: Get the conditioning of a matrix according to   */
/*         the condition number defined as:                */
/*               cond(A)=||A|| ||inverse of A||            */
/*         and the definition of ||A|| is according to     */
/*         uniform matrix norm.                            */
/*         Need to call Inverse() to get inverse of the    */
/*         matrix first. If the condition number of a      */
/*         matrix is greater than 500, we think it as ill- */
/*         conditioning, the function returns ILL,         */
/*         otherwise returns WELL.                         */
/**********************************************************/
int Condition(a,d,b,c)
double a,d,b,c;
{
  double ai,bi,ci,di,MaxA,MaxAi,MaxA1,MaxAi1,cond;
  double aa,bb,cc,dd;

       /* get inverse of a matrix */
  Inverse(&ai,&bi,&ci,&di,a,d,b,c);
  AbsoluteDouble(a,&aa);
  AbsoluteDouble(b,&bb);
  AbsoluteDouble(c,&cc);
  AbsoluteDouble(d,&dd);
  MaxA=max(aa+bb,cc+dd);  /* uniform matrix norm */

  AbsoluteDouble(ai,&aa);
  AbsoluteDouble(bi,&bb);
  AbsoluteDouble(ci,&cc);
  AbsoluteDouble(di,&dd);
  MaxAi=max(aa+bb,cc+dd); /* uniform matrix norm */

  cond=MaxA*MaxAi;  /* the definition of condition number */
  if(cond>ConditionNo)
    return(Ill);    /* ill-conditioning */
  else
    return(Well);   /* well-conditioning */
}

/**********************************************************/
/*  FUNCTION NAME: void RowScaled()                        */
/*  PURPOSE: To scale a selected matrix if the components  */
/*           of the matrix is much different.              */
/*  LOGIC: Scale each row whose component is divided by the*/
/*         larger component of the row.                     */
/***********************************************************/
void RowScaled(a,b,c,d)
double a,b,c,d;
{
  double aa,bb,cc,dd;

  AbsoluteDouble(a,&aa);
  AbsoluteDouble(b,&bb);
  AbsoluteDouble(c,&cc);
  AbsoluteDouble(d,&dd);
  if((aa!=0)&&(aa>bb))    /* scale the matrix */
  {
    if(((bb/aa)<0.027)||((bb/aa)==0)||(ScaleMatrix==1))
    {
      SelectMSF[0]=a/a;
      SelectMSF[1]=b/a;
      AbsoluteDouble(SelectMSF[0],&aa);
      AbsoluteDouble(SelectMSF[1],&bb);
    }
  }
  if((bb!=0)&&(bb>aa))
  {
    if(((aa/bb)<0.027)||((aa/bb)==0)||(ScaleMatrix==1))
    {
      SelectMSF[0]=a/b;
      SelectMSF[1]=b/b;
      AbsoluteDouble(SelectMSF[0],&aa);
      AbsoluteDouble(SelectMSF[1],&bb);
    }
  }
  if((cc!=0)&&(cc>dd))
  {
    if(((dd/cc)<0.027)||((dd/cc)==0)||(ScaleMatrix==1))
    {
      SelectMSF[2]=c/c;
      SelectMSF[3]=d/c;
      AbsoluteDouble(SelectMSF[2],&cc);
      AbsoluteDouble(SelectMSF[3],&dd);
    }
  }
  if((dd!=0)&&(dd>cc))
  {
    if(((cc/dd)<0.027)||((cc/dd)==0)||(ScaleMatrix==1))
    {
      SelectMSF[2]=c/d;
      SelectMSF[3]=d/d;
    }
  }
}

/***********************************************************/
/*  FUNCTION NAME: void ColumnScaled()                      */
/*  PURPOSE: To scale a selected matrix if the components  */
/*           of the matrix is much different.              */
/*  LOGIC: Scale each column whose component is divided by */
/*         the larger component of the column.             */
/**********************************************************/
void ColumnScaled(a,b,c,d)
double a,b,c,d;
{
  double aa,bb,cc,dd;

  AbsoluteDouble(a,&aa);
  AbsoluteDouble(b,&bb);
  AbsoluteDouble(c,&cc);
  AbsoluteDouble(d,&dd);
  if((aa!=0)&&(aa>cc))    /* scale the matrix */
  {
    if(((cc/aa)<0.027)||((cc/aa)==0)||(ScaleMatrix==2))
    {
      SelectMSF[0]=a/a;
      SelectMSF[2]=c/a;
      AbsoluteDouble(SelectMSF[0],&aa);
      AbsoluteDouble(SelectMSF[2],&cc);
    }
  }
  if((cc!=0)&&(cc>aa))
  {
    if(((aa/cc)<0.027)||((aa/cc)==0)||(ScaleMatrix==2))
    {
      SelectMSF[0]=a/c;
      SelectMSF[2]=c/c;
      AbsoluteDouble(SelectMSF[0],&aa);
      AbsoluteDouble(SelectMSF[2],&cc);
    }
  }
  if((bb!=0)&&(bb>dd))
  {
    if(((dd/bb)<0.027)||((dd/bb)==0)||(ScaleMatrix==2))
    {
      SelectMSF[1]=b/b;
      SelectMSF[3]=d/b;
      AbsoluteDouble(SelectMSF[1],&bb);
      AbsoluteDouble(SelectMSF[3],&dd);
    }
  }
  if((dd!=0)&&(dd>bb))
  {
    if(((bb/dd)<0.027)||((bb/dd)==0)||(ScaleMatrix==2))
    {
      SelectMSF[1]=b/d;
      SelectMSF[3]=d/d;
    }
  }
}
```

```
/*********************************************************/
/*  FUNCTION NAME: void ScaleSelectedM()                 */
/*  PURPOSE: To scale a selected matrix according to users'*/
/*           request.                                    */
/*********************************************************/
void ScaleSelectedM(a,b,c,d)
double a,b,c,d;
{
  SelectMSF[0]=a;
  SelectMSF[1]=b;
  SelectMSF[2]=c;
  SelectMSF[3]=d;
  if(ScaleMatrix==1)  /* scale rows of a matrix */
    RowScaled(a,b,c,d);
          /* scale rows and columns of a matrix */
  else if(ScaleMatrix==2)
  {
    RowScaled(a,b,c,d);
    ColumnScaled(SelectMSF[0],SelectMSF[1],
                 SelectMSF[2],SelectMSF[3]);
  }
}

/*********************************************************/
/*  FUNCTION NAME: void DrawIllYX()                      */
/*  PURPOSE: To mark the pixels that represent ill-      */
/*           conditioned matrice in a 2-D space that is an */
/*           area closer to a hyperbola that represents   */
/*           singer matrices.                             */
/*  LOGIC: Get the world value of the screen pixels which */
/*         reprent matrices, and call ScaleSelectedM() to */
/*         check if the matrix needed to be scaled and if */
/*         true then scale it, and call Condition() to    */
/*         determine the conditioning of a matrix, if get */
/*         it is ill-conditioning, call putpixel() to mark */
/*         that pixel in LIGHTRED color.                  */
/*********************************************************/
void DrawIllYX(b,c,starty,endy,startx,endx,flag,part)
double b,c;
int starty,endy,startx,endx,flag,part;
{
  double a,d,orgx,orgy,xscale,yscale,fxlabel,fylabel,cc;
  double oxlabel,oylabel,tempx,tempy;
  int x,y,StartIllX,FirstIll,WellCounter;

  orgx=OrgX;
  orgy=OrgY;
  xscale=XScale;
  yscale=YScale;
  fxlabel=FirstXLabel;
  fylabel=FirstYLabel;
  oxlabel=OrgXLabel;
  oylabel=OrgYLabel;
  if((flag==1)&&(part==1))   /* for the case of b*c<0 */
```

```
{                 /* draw a figure shown on 3rd quadrant */
  for(y=starty;y<=endy;y++)
  {
    tempy=y;
    StartIllX=-1;     /* reset counters */
    FirstIll=0;
    WellCounter=0;
    for(x=startx;x>=endx;x--)
    {
      tempx=x;
      a=(tempx-orgx)/xscale*(fxlabel-oxlabel);
      d=(orgy-tempy)/yscale*(fylabel-oylabel);
      ScaleSelectedM(a,b,c,d);
      if((SelectMSF[0]*SelectMSF[3]-
          SelectMSF[1]*SelectMSF[2])!=0)
      {
        if(Condition(SelectMSF[0],SelectMSF[3],
                     SelectMSF[1],SelectMSF[2])==Ill)
        {            /* find an ill-conditioned case */
          putpixel(x,y,LIGHTRED);
          if(FirstIll==0)
          {
            FirstIll=1;
            StartIllX=x;
          }
        }
        else
        {
          if((StartIllX!=-1)&&(FirstIll==1))
            WellCounter++;
          if(WellCounter>10)
          {
            x=endx;
            startx=StartIllX;
          }
        }
      }
    }
  }
}
else if((flag==1)&&(part==2))  /* for the case of b*c<0 */
{                 /* draw a figure shown on 2nd quadrant */
  for(y=starty;y>=endy;y--)
  {
    tempy=y;
    StartIllX=-1;    /* reset counters */
    FirstIll=0;
    WellCounter=0;
    for(x=startx;x<=endx;x++)
    {
      tempx=x;
      a=(tempx-orgx)/xscale*(fxlabel-oxlabel);
      d=(orgy-tempy)/yscale*(fylabel-oylabel);
      ScaleSelectedM(a,b,c,d);
```

```
          if((SelectMSF[0]*SelectMSF[3]-
              SelectMSF[1]*SelectMSF[2])!=0)
           {
             if(Condition(SelectMSF[0],SelectMSF[3],
                          SelectMSF[1],SelectMSF[2])==Ill)
             {          /* find an ill-conditioned case */
               putpixel(x,y,LIGHTRED);
               if(FirstIll==0)
               {
                 FirstIll=1;
                 StartIllX=x;
               }
             }
             else
             {
               if((StartIllX!=-1)&&(FirstIll==1))
                 WellCounter++;
               if(WellCounter>10)
               {
                 x=endx;
                 startx=StartIllX;
               }
             }
           }
         }
       }
     }
     else if((flag==2)&&(part==1))  /* for the case of b*c>0 */
     {                 /* draw a figure shown on 4th quadrant */
       for(y=starty;y>=endy;y--)
       {
         tempy=y;
         StartIllX=-1;      /* reset counters */
         FirstIll=0;
         WellCounter=0;
         for(x=startx;x>=endx;x--)
         {
           tempx=x;
           a=(tempx-orgx)/xscale*(fxlabel-oxlabel);
           d=(orgy-tempy)/yscale*(fylabel-oylabel);
           ScaleSelectedM(a,b,c,d);
           if((SelectMSF[0]*SelectMSF[3]-
             SelectMSF[1]*SelectMSF[2])!=0)
           {
             if(Condition(SelectMSF[0],SelectMSF[3],
                          SelectMSF[1],SelectMSF[2])==Ill)
             {            /* find an ill-conditioned case */
               putpixel(x,y,LIGHTRED);
               if(FirstIll==0)
               {
                 FirstIll=1;
                 StartIllX=x;
               }
             }
          else
          {
            if((StartIllX!=-1)&&(FirstIll==1))
              WellCounter++;
            if(WellCounter>10)
            {
              x=endx;
              startx=StartIllX;
            }
          }
        }
      }
    }
  }
  else if((flag==2)&&(part==2)) /* for the case of b*c>0 */
  {                /* draw a figure shown on 1st quadrant */
    for(y=starty;y<=endy;y++)
    {
      tempy=y;
      StartIllX=-1;   /* reset counters */
      FirstIll=0;
      WellCounter=0;
      for(x=startx;x<=endx;x++)
      {
        tempx=x;
        a=(tempx-orgx)/xscale*(fxlabel-oxlabel);
        d=(orgy-tempy)/yscale*(fylabel-oylabel);
        ScaleSelectedM(a,b,c,d);
        if((SelectMSF[0]*SelectMSF[3]-
            SelectMSF[1]*SelectMSF[2])!=0)
        {
          if(Condition(SelectMSF[0],SelectMSF[3],
                       SelectMSF[1],SelectMSF[2])==Ill)
          {                /* find an ill-conditioned case */
            putpixel(x,y,LIGHTRED);
            if(FirstIll==0)
            {
              FirstIll=1;
              StartIllX=x;
            }
          }
          else
          {
            if((StartIllX!=-1)&&(FirstIll==1))
              WellCounter++;
            if(WellCounter>10)
            {
              x=endx;
              startx=StartIllX;
            }
          }
        }
      }
    }
  }
  else if(flag==3)   /* for the case of b*c=0 */
  {
    for(y=starty;y<=endy;y++)
    {
      tempy=y;
      StartIllX=-1;     /* reset counters */
      FirstIll=0;
      WellCounter=0;
      for(x=startx;x<=endx;x++)
      {
        tempx=x;
        a=(tempx-orgx)/xscale*(fxlabel-oxlabel);
        d=(orgy-tempy)/yscale*(fylabel-oylabel);
        ScaleSelectedM(a,b,c,d);
        if((SelectMSF[0]*SelectMSF[3]-
            SelectMSF[1]*SelectMSF[2])!=0)
        {
          if(Condition(SelectMSF[0],SelectMSF[3],
                       SelectMSF[1],SelectMSF[2])==Ill)
          {            /* find an ill-conditioned case */
            putpixel(x,y,LIGHTRED);
            if(FirstIll==0)
            {
              FirstIll=1;
              StartIllX=x;
            }
          }
          else
          {
            if((StartIllX!=-1)&&(FirstIll==1))
              WellCounter++;
            if(WellCounter>10)
            {
              x=endx;
              startx=StartIllX;
            }
          }
        }
      }
    }
  }
}

/*******************************************************/
/*  FUNCTION NAME: void SetupFigureConstant()          */
/*  PURPOSE: To set some constants that will be needed in  */
/*           other functions.                          */
/*******************************************************/
void SetupFigureConstant(b,c,figure,fx,fy,startx1,endx1,
                  starty1,endy1,startx2,endx2,starty2,endy2)
double b,c;
int figure,*fx,*fy,*startx1,*endx1,*starty1,*endy1;
int *startx2,*endx2,*starty2,*endy2;
```

```
{
  int sign=0;

  OrgM[0][1]='\0';
  OrgM[1][1]='\0';
  OrgM[2][1]='\0';
  OrgM[3][1]='\0';
  if(b*c<0)
    sign=1;
  if(figure<5)  /* for wnidow number = 4 */
  {
    *fx=306;   /* the maximum x */
    *fy=220;   /* the maxmum y */
    *startx1=4;   /* for DrawIll() using */
    *endx1=157;
    *startx2=157;
    *endx2=306;
    if(sign==1)  /* for drawing figures on the 2nd and 3rd */
    {            /* quadrants */
      *starty1=12;
      *endy1=112;
      *starty2=112;
      *endy2=220;
    }
    else/* for drawing figures on the 1st and 4th quadrants */
    {
      *starty1=112;
      *endy1=220;
      *starty2=12;
      *endy2=112;
    }
  }
  else if((figure==5)||(figure==6))/* for window number = 1 */
  {
    *fx=628;    /* the maximum x */
    *fy=460;    /* the maximum y */
    *startx1=4;
    *endx1=318;
    *startx2=318;
    *endx2=628;
    if(sign==1)/* for drawing figures on the 2nd and 3rd */
    {          /* quadrants */
      *starty1=12;
      *endy1=228;
      *starty2=228;
      *endy2=460;
    }
    else/* for drawing figures on the 1st and 4th quadrants */
    {
      *starty1=228;
      *endy1=460;
      *starty2=12;
      *endy2=228;
    }
  }
  else if((figure==7)||(figure==8))/* for window number = 2 */
  {
    *fx=306;
    *fy=460;
    *startx1=4;
    *endx1=157;
    *startx2=157;
    *endx2=306;
    if(sign==1)/* for drawing figures on the 2nd and 3rd */
    {           /* quadrants */
      *starty1=12;
      *endy1=228;
      *starty2=228;
      *endy2=460;
    }
    else/* for drawing figures on the 1st and 4th quadrants */
    {
      *starty1=228;
      *endy1=460;
      *starty2=12;
      *endy2=228;
    }
  }
}

/**********************************************************/
/*  FUNCTION NAME: void CheckSelectedMCond()              */
/*  PURPOSE: To check the conditioning of a selected      */
/*           matrix.                                      */
/*  LOGIC: Call Condition() to get the conditioning of a  */
/*         matrix, and store its conditioning type as a   */
/*         string for WriteNote() using.                  */
/**********************************************************/
void CheckSelectedMCond(a,b,c,d,string)
double a,b,c,d;
char string[];
{
  if((a*d-b*c)==0)          /* a singular matrix */
    strcpy(string,"SINGULAR");
  else if((a*d-b*c)!=0)
  {
    ScaleSelectedM(a,b,c,d);
    if(Condition(SelectMSF[0],SelectMSF[3],
                 SelectMSF[1],SelectMSF[2])==Ill)
      strcpy(string,"ILL"); /* a ill-conditioned matrix */
    else
      strcpy(string,"WELL"); /* a well-conditioned matrix */
  }
}

/**********************************************************/
/*  FUNCTION NAME: void DrawFigure()                      */
/*  PURPOSE: To draw figures in windows on screen,        */
/*           including draw axes, write related          */
/*           information, draw singular matrices, draw    */
/*           eigenvectors, draw ill-conditioned matrices.  */
/*  LOGIC: Control the figures drawn type by opt parameter.*/
/*         For demo, setup variables function, opt=1.     */
/*         For enlarge and old figures function, opt=2.   */
/*         For reduce and old figures function, opt=3.    */
/*         Others, opt=4.                                 */
/*********************************************************/
void DrawFigure(opt,figure,b,c,a,d)
int opt,figure;
double b,c,a,d;
{
  char temp[4][20],string[10];
  int fx,fy,startx1,endx1,starty1,endy1,startx2,endx2;
  int starty2,endy2,x,y;

  EigenValue();
  SetupFigureConstant(b,c,figure,&fx,&fy,&startx1,&endx1,
       &starty1,&endy1,&startx2,&endx2,&starty2,&endy2);
  strcpy(temp[1],OrgM[1]);
  strcpy(temp[2],OrgM[2]);
  if(((opt==1)&&((figure<=5)||(figure==7)))||
     (((opt==2)||(opt==3))&&((figure==1)||(figure==3)||
     (figure==5)||(figure==7)))||((opt==4)&&(figure<9)))
  {
    strcat(OrgM[1],"= ");
    strcat(OrgM[1],SelectM[1]);
    strcat(OrgM[2],"= ");
    strcat(OrgM[2],SelectM[2]);
  }
  if(((opt==1)&&((figure==6)||(figure==8)))||
     (((opt==2)||(opt==3))&&((figure==2)||(figure==4)||
     (figure==6)||(figure==8))))
  {
    strcat(OrgM[0],"= ");
    strcat(OrgM[0],SelectM[0]);
    strcat(OrgM[3],"= ");
    strcat(OrgM[3],SelectM[3]);
  }
  CheckSelectedMCond(a,b,c,d,string);
  switch(figure)
  {
    case 1: setviewport(LX1,LY1,HX1,HY1,1);
            if((opt==1)||(opt==4))
              WriteNote(opt,1,OrgM[1],OrgM[2],OrgM[0],
                        temp[1],temp[2],OrgM[3],string);
            else if((opt==2)||(opt==3))
              WriteNote(opt,1,OrgM[1],OrgM[2],SelectM[0],
                   SelectM[1],SelectM[2],SelectM[3],string);
            DrawAxes(4,LIGHTCYAN,1);
            break;
    case 2: setviewport(LX2,LY2,HX2,HY2,1);
            /* for DemoProcess() and SetupVariables() */
```

```
         if((opt==1)||(opt==4))
         {
           WriteNote(opt,2,OrgM[1],OrgM[2],OrgM[0],
                     temp[1],temp[2],OrgM[3],string);
           DrawAxes(4,LIGHTCYAN,1);
         }
         else if((opt==2)||(opt==3))
         {
           WriteNote(opt,2,OrgM[0],OrgM[3],SelectM[0],
                 SelectM[1],SelectM[2],SelectM[3],string);
           DrawAxes(4,LIGHTCYAN,2);
         }
         break;
case 3:  setviewport(LX3,LY3,HX3,HY3,1);
         if((opt==1)||(opt==4))
           WriteNote(opt,3,OrgM[1],OrgM[2],OrgM[0],
                     temp[1],temp[2],OrgM[3],string);
         else if((opt==2)||(opt==3))
           WriteNote(opt,3,OrgM[1],OrgM[2],SelectM[0],
                 SelectM[1],SelectM[2],SelectM[3],string);
         DrawAxes(4,LIGHTCYAN,1);
         break;
case 4:  setviewport(LX4,LY4,HX4,HY4,1);
         if((opt==1)||(opt==4))
         {
           WriteNote(opt,4,OrgM[1],OrgM[2],OrgM[0],
                     temp[1],temp[2],OrgM[3],string);
           DrawAxes(4,LIGHTCYAN,1);
         }
         else if((opt==2)||(opt==3))
         {
           WriteNote(opt,4,OrgM[0],OrgM[3],SelectM[0],
                 SelectM[1],SelectM[2],SelectM[3],string);
           DrawAxes(4,LIGHTCYAN,2);
         }
         break;
case 5:  setviewport(LX1,LY1,HX1,HY1,1);
         if(opt==4)
           WriteNote(opt,5,OrgM[1],OrgM[2],OrgM[0],
                     temp[1],temp[2],OrgM[3],string);
         else
           WriteNote(opt,5,OrgM[1],OrgM[2],SelectM[0],
                 SelectM[1],SelectM[2],SelectM[3],string);
         DrawAxes(1,LIGHTCYAN,1);
         break;
case 6:  setviewport(LX1,LY1,HX1,HY1,1);
         if(opt==4)
         {
           WriteNote(opt,6,OrgM[1],OrgM[2],OrgM[0],
                     temp[1],temp[2],OrgM[3],string);
           DrawAxes(1,LIGHTCYAN,1);
         }
         else
         {
              WriteNote(opt,6,OrgM[0],OrgM[3],SelectM[0],
                   SelectM[1],SelectM[2],SelectM[3],string);
              DrawAxes(1,LIGHTCYAN,2);
            }
            break;
    case 7: setviewport(LX1,LY1,HX1,HY1,1);
            if(opt==4)
              WriteNote(opt,7,OrgM[1],OrgM[2],OrgM[0],
                        temp[1],temp[2],OrgM[3],string);
            else
              WriteNote(opt,7,OrgM[1],OrgM[2],SelectM[0],
                   SelectM[1],SelectM[2],SelectM[3],string);
            DrawAxes(2,LIGHTCYAN,1);
            break;
    case 8: setviewport(LX2,LY2,HX2,HY2,1);
            if(opt==4)
            {
              WriteNote(opt,8,OrgM[1],OrgM[2],OrgM[0],
                        temp[1],temp[2],OrgM[3],string);
              DrawAxes(2,LIGHTCYAN,1);
            }
            else
            {
              WriteNote(opt,8,OrgM[0],OrgM[3],SelectM[0],
                   SelectM[1],SelectM[2],SelectM[3],string);
              DrawAxes(2,LIGHTCYAN,2);
            }
            break;
    default: break;
  }
  DrawSingular(b,c,fx,fy,OrgX,OrgY,XScale,YScale,OrgXLabel,
               FirstXLabel,OrgYLabel,FirstYLabel);
  MarkSelectPoint(a,d,OrgX,OrgY,XScale,YScale,OrgXLabel,
  FirstXLabel,OrgYLabel,FirstYLabel,&x,&y);
  DrawEigenvectors(x,y,XScale,YScale,OrgXLabel,
                   FirstXLabel,OrgYLabel,FirstYLabel);
  if(b*c==0)
  {
    if((b==0)&&(c==0))   /* well-conditioned matrices */
      ;
    else
      DrawIllYX(b,c,4,fx,12,fy,3,0);
  }
  else if(b*c<0)
  {            /* draw a figure shown on the 3rd quadrant */
    DrawIllYX(b,c,starty1,endy1,endx1,startx1,1,1);
       /* draw a figure shown on the 3rd quadrant */
    DrawIllYX(b,c,endy2,starty2,startx2,endx2,1,2);
  }
  else if(b*c>0)
  {           /* draw a figure shown on the 4th quadrant */
    DrawIllYX(b,c,endy1,starty1,endx1,startx1,2,1);
      /* draw a figure shown on the 1st quadrant */
    DrawIllYX(b,c,starty2,endy2,startx2,endx2,2,2);
  }
}

/**********************************************************/
/*  FUNCTION NAME: int RealPtoScrP()                      */
/*  PURPOSE: To convert a world value to be physical screen*/
/*           pixel  value for drawing figures.            */
/*  LOGIC : According to the position of the origin,      */
/*          pixels numbers per scale and the original label*/
/*          and the first scale label, we can calculate   */
/*          the screen position for a world value.        */
/**********************************************************/
int RealPtoScrP(a,d,orgxp,orgyp,scale,count0,count1,flag)
double  a,d;
int orgxp,orgyp,scale;
double count0,count1;
int flag;
{
  double section,scrp;
  int ScrPt;

  if(flag==1)  /* calculate for x coordinate */
  {
    if(a>count0)
    {
      section=(a-count0)/(count1-count0);
      scrp=orgxp+(section*scale);
      if(scrp>639)
        scrp=639;
    }
    else
    {
      section=(count0-a)/(count1-count0);
      scrp=orgxp-(section*scale);
      if(scrp<0)
        scrp=0;
    }
  }
  else if(flag==2)  /* calculate for y coordinate */
  {
    if(d>count0)
    {
      section=(d-count0)/(count1-count0);
      scrp=orgyp-(section*scale);
      if(scrp<0)
        scrp=0;
    }
    else
    {
      section=(count0-d)/(count1-count0);
      scrp=orgyp+(section*scale);
      if(scrp>479)
        scrp=479;
    }
  }
  ScrPt=scrp;
  /* return the physical location on the screen */
  return(ScrPt);
}

/**********************************************************/
/*  FUNCTION NAME: void ClearUserScreen()                 */
/*  PURPOSE: To clear an active window on screen.         */
/**********************************************************/
void ClearUserScreen()
{
  setviewport(0,0,MaxX,MaxY,1);
  /* draw background color of major window box */
  DrawBar(LowX,LowY-6,HighX,HighY,BLUE);
  WindowBox(LowX,LowY,HighX,HighY,WHITE);
  Menu(LIGHTGRAY,LIGHTGRAY,LIGHTGRAY,LIGHTGRAY);
}

/**********************************************************/
/*  FUNCTION NAME: void GetUserOption()                   */
/*  PURPOSE: To get a key form mouse or keyboard by user  */
/*           input.                                       */
/**********************************************************/
void GetUserOption(ch,x,y,opt)
char *ch;
int  *x,*y,opt;
{
  setviewport(0,0,MaxX,MaxY,1);
  settextjustify(LEFT_TEXT,TOP_TEXT);
  DisplayMouseCursor();
  if(opt==1)
  {
    RemoveMouseCursor();
    Menu(LIGHTGRAY,LIGHTGRAY,LIGHTGRAY,LIGHTGRAY);
    DisplayMouseCursor();
  }
  *ch=KeepWaitingInput(CENTER);
  GetMousePosition(&*x,&*y);
  RemoveMouseCursor();
}

/**********************************************************/
/*  FUNCTION NAME: void ExplainText()                     */
/*  PURPOSE: To explain what this package is doing. For   */
/*          DemoProcess() using.                          */
/**********************************************************/
void ExplainText()
{
  setviewport(LowX+2,LowY+2,HighX-2,HighY-2,1);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
  settextjustify(LEFT_TEXT,TOP_TEXT);
  setcolor(WHITE);
  outtextxy(18,18,"Demonstrate a two by two matrix A in");
  outtextxy(18,40,"a 2-D space ...");
  setcolor(LIGHTGREEN);
  outtextxy(18,62,"A = | a    b |");
  outtextxy(18,80,"    | c    d |");
  setcolor(WHITE);
  outtextxy(18,102,"For a ");
  setcolor(LIGHTRED);
  outtextxy(18+6*textwidth("A"),102,"singular matrix A,");
  setcolor(YELLOW);
  outtextxy(18,124,"det A = ad - bc = 0");
  setcolor(WHITE);
  outtextxy(18,146,"such that   ");
  setcolor(LIGHTRED);
  outtextxy(18+12*textwidth("A"),146,"ad = bc");
  setcolor(WHITE);
  outtextxy(18,168,"If we set b=1 and c=2, then ad=2 which");
  outtextxy(18,190,"is a hyperbola in a 2-D space.  We");
  outtextxy(18,212,"draw the singular points in WHITE");
  outtextxy(18,234,"color, mark the ill-conditioned points");
  outtextxy(18,256,"in ");
  setcolor(LIGHTRED);
  outtextxy(18+3*textwidth("A"),256,"RED ");
  setcolor(WHITE);
  outtextxy(18+7*textwidth("A"),256,
            "color, and draw the real parts");
  outtextxy(18,278,"of eigenvectors of a selected point in");
  setcolor(LIGHTMAGENTA);
  outtextxy(18,300,"LIGHTMAGENTA ");
  setcolor(WHITE);
  outtextxy(18+13*textwidth("A"),300,"and ");
  setcolor(LIGHTGREEN);
  outtextxy(18+17*textwidth("A"),300,"LIGHTGREEN ");
  setcolor(WHITE);
  outtextxy(18+28*textwidth("A"),300,"color, the");
  outtextxy(18,322,"complex parts in ");
  setcolor(LIGHTGRAY);
  outtextxy(18+17*textwidth("A"),322,"LIGHTGRAY ");
  setcolor(WHITE);
  outtextxy(18+27*textwidth("A"),322,"and ");
  setcolor(BROWN);
  outtextxy(18+31*textwidth("A"),322,"BROWN");
  setcolor(WHITE);
  outtextxy(18,344,"color.");
  setcolor(YELLOW);
  outtextxy(18,400,"Press any key to continue ...");
  outtextxy(18,425,"Or press 'q' or 'Q' to exit.");
}

/***********************************************************/
/*  FUNCTION NAME: void DemoProcess()                      */
/*  PURPOSE: To demonstrate a two by two matrix in a 2-D   */
/*           space by showing its eigenvectos and other    */
/*           closer matrices that are singular, ill-       */
/*           conditioned, or well-conditioned.             */
/*  LOGIC : Need a input file matrix.dat to readin data.   */
/*          There are 4 figures shown on screen each time. */
/***********************************************************/
void DemoProcess()
{
  double temp;
  char ch;
  int i,j,start=1,x,y;

  WindowSetup(4,1);
  ClearUserScreen();
  WindowBox(LowX+2,LowY+2,HighX-2,HighY-2,WHITE);
  setviewport(LowX+2,LowY+2,HighX-2,HighY-2,1);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
  settextjustify(LEFT_TEXT,TOP_TEXT);
  setcolor(WHITE);
  outtextxy(18,18,"Please put input file - matrix.dat -");
  outtextxy(18,48,"in drive B.");
  outtextxy(18,78,"When ready, press any key ......");
  outtextxy(18,108,"Or press 'q' or 'Q' to exit.");
  GetUserOption(&ch,&x,&y,0);
  if((ch=='q')||(ch=='Q')||(IfMouseInWindow(x,y,
      4*MaxX/5,0,MaxX-4*textwidth("Q"),6+textheight("Q"))))
  {
    ClearUserScreen();
    DisplayMouseCursor();
    return;
  }
  ClearUserScreen();
  if(!(fip=fopen("b:matrix.dat","r"))) /* open input file */
  {
    setviewport(LowX+2,LowY+2,HighX-2,HighY-2,1);
    settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
    settextjustify(LEFT_TEXT,TOP_TEXT);
    setcolor(RED);
    outtextxy(18,18,"Cannot open input file.");
    delay(2000);
    ClearUserScreen();
    DisplayMouseCursor();
    return;
  }
  while(!feof(fip))  /* readin data until end of file */
  {
    WindowBox(LowX+2,LowY+2,HighX-2,HighY-2,WHITE);
    if(start==1)
    {
      ExplainText();
      GetUserOption(&ch,&x,&y,0);
      if((ch=='q')||(ch=='Q')||(IfMouseInWindow(x,y,
      4*MaxX/5,0,MaxX-4*textwidth("Q"),6+textheight("Q"))))
      {
        ClearUserScreen();
        DisplayMouseCursor();
        fclose(fip);
    return;
  }
  ClearUserScreen();
  start=2;
}
for(i=1;i<5;i++)
{
  if(!feof(fip))
  {
    fscanf(fip,"%s%s%s%s",SelectM[0],SelectM[1],
           SelectM[2],SelectM[3]);
    for(j=0;j<4;j++)
      ChartoDouble(SelectM[j],&SelectMF[j]);
    setviewport(0,0,MaxX,MaxY,1);
    switch(i)
    {
      case 1: WindowBox(LX1,LY1,HX1,HY1,WHITE);
              DrawFigure(1,1,SelectMF[1],SelectMF[2],
                         SelectMF[0],SelectMF[3]);
              break;
      case 2: WindowBox(LX2,LY2,HX2,HY2,WHITE);
              DrawFigure(1,2,SelectMF[1],SelectMF[2],
                         SelectMF[0],SelectMF[3]);
              break;
      case 3: WindowBox(LX3,LY3,HX3,HY3,WHITE);
              DrawFigure(1,3,SelectMF[1],SelectMF[2],
                         SelectMF[0],SelectMF[3]);
              break;
      case 4: WindowBox(LX4,LY4,HX4,HY4,WHITE);
              DrawFigure(1,4,SelectMF[1],SelectMF[2],
                         SelectMF[0],SelectMF[3]);
              break;
      default: break;
    }
  }
  else
 i=5;
}
setviewport(0,0,MaxX,MaxY,1);
settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
settextjustify(LEFT_TEXT,TOP_TEXT);
setcolor(BROWN);
outtextxy(118,238,"press any key to continue or");
outtextxy(118,258,"press 'q' or 'Q' to exit.");
GetUserOption(&ch,&x,&y,0);
if((ch=='q')||(ch=='Q')||(IfMouseInWindow(x,y,
  4*MaxX/5,0,MaxX-4*textwidth("Q"),6+textheight("Q"))))
{
  ClearUserScreen();
  DisplayMouseCursor();
  fclose(fip);
  return;
}
ClearUserScreen();
```

```
  }
  fclose(fip);
  DisplayMouseCursor();
}

/************************************************************/
/*  FUNCTION NAME: void GetUserInput()                      */
/*  PURPOSE: To get a selected matrix and relative          */
/*           variables by user input from keyboard.         */
/************************************************************/
void GetUserInput(string,x,y,count)
char string[];
int x,y,count;
{
  int i=1,j,bkflag=0;
  char tempch[1];

  setcolor(YELLOW);
  tempch[1]='\0';
  j=0;
  outtextxy(x+(count+j)*textwidth("A"),y,"_");
  while(i)
  {
    string[j]=getch();
    tempch[0]=string[j];
    if(tempch[0]=='\b')
    {
      putimage(x+(count+j)*textwidth("A"),y,images[0],
               COPY_PUT);
      if(j>=1)
      {
        putimage(x+(count+j-1)*textwidth("A"),y,images[0],
                 COPY_PUT);
        outtextxy(x+(count+j-1)*textwidth("A"),y,"_");
        j--;
      }
      else
        outtextxy(x+(count+j)*textwidth("A"),y,"_");
      bkflag=1;
    }
    else if(tempch[0]!='\r')
    {
      putimage(x+(count+j)*textwidth("A"),y,images[0],
       COPY_PUT);
      outtextxy(x+(count+j)*textwidth("A"),y,tempch);
      outtextxy(x+(count+j+1)*textwidth("A"),y,"_");
      bkflag=0;
      j++;
    }
    else
    {
      if(bkflag==1)
        bkflag=0;
      i=0;
    }
  }
  string[j]='\0';
  if(bkflag==0)
    putimage(x+(count+j)*textwidth("A"),y,images[0],
             COPY_PUT);
  else if(bkflag==1)
    putimage(x+(count+j-1)*textwidth("A"),y,images[0],
             COPY_PUT);
  setcolor(WHITE);
}

/************************************************************/
/*  FUNCTION NAME: int CheckSelectMatrix()                  */
/*  PURPOSE: To avoid illegal user input in selecting a     */
/*           matrix.                                        */
/*  LOGIC: If illegal user input exist, return 0. If user   */
/*         choose an unknown matrix, return 2. Otherwise    */
/*         return 1.                                        */
/************************************************************/
int CheckSelectMatrix()
{
  int i,j,dot,divided,unknown=0;
  char c;

  for(i=0;i<4;i++)
  {
    dot=0;
    divided=0;
    for(j=0;j<strlen(SelectM[i]);j++)
    {
      c=SelectM[i][j];
      if(((c>='a')&&(c<='z'))||((c>='A')&&(c<='Z')))
      {         /* an alphabetic */
       if((c=='e')||(c=='E'))
       {
         if((SelectM[i][j+1]=='+')||(SelectM[i][j+1]=='-')
           ||((SelectM[i][j+1]>='0')&&(SelectM[i][j+1]<='9')))
           ;
         else if((j==0)&&(SelectM[i][1]=='\0'))
           unknown++;
         else
           return(0);
       }
       else if((j==0)&&(SelectM[i][1]=='\0'))
         unknown++;
       else if((j!=0)||(SelectM[i][1]!='\0'))
         return(0);
      }
      else if(c=='.')
      {
        if(dot==0)
          dot=1;
        else      /* two more dot exist in a string */
```

```
          return(0);
        }
        else if(c=='/')
        {
          if(divided==0)
            divided=1;
          else       /* two more fraction exist in a string */
            return(0);
        }
        else if((c>='0')&&(c<='9'))
          ;
        else if((c=='-')&&(SelectM[i][j+1]!='\0'))
          ;
        else if((c=='+')&&(SelectM[i][j+1]!='\0'))
          ;
        else
          return(0);
      }
    }
    if(unknown==4) /* user selects an unknown matrix */
      return(2);
    return(1);
  }

  /**********************************************************/
  /*  FUNCTION NAME: int CheckSelectMatrix()              */
  /*  PURPOSE: To avoid illegal user input in setting x, y   */
  /*           labels.                                     */
  /*  LOGIC: If illegal user input exist, return 0. Otherwise*/
  /*         return 1.                                     */
  /**********************************************************/
  int CheckSelectXYLabel(string)
  char string[];
  {
    int j,dot,divided;
    char c;

    dot=0;
    divided=0;
    for(j=0;j<strlen(string);j++)
    {
      c=string[j];
      if((c>='0')&&(c<='9'))
        ;
      else if(c=='.')
      {
        if(dot==0)
          dot=1;
        else
          return(0);
      }
      else if(c=='/')
      {
        if(divided==0)
        divided=1;
      else
        return(0);
    }
    else if((c=='e')||(c=='E'))
    {
      if((string[j+1]=='+')||(string[j+1]=='-')
        ||((string[j+1]>='0')&&(string[j+1]<='9')))
        ;
      else
        return(0);
    }
    else if((c=='-')&&(string[j+1]!='\0'))
      ;
    else if((c=='+')&&(string[j+1]!='\0'))
      ;
    else
      return(0);
  }
  return(1);
}

/*******************************************************/
/*  FUNCTION NAME: void ErrorMessage()                 */
/*  PURPOSE: To output error message when got an illegal   */
/*           user input.                                */
/*******************************************************/
void ErrorMessage()
{
  settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
  setcolor(LIGHTRED);
  outtextxy(18,388,"ERROR! --- Illegal User Input.");
  delay(400);
  ClearUserScreen();
}

/*******************************************************/
/*  FUNCTION NAME: int SelectMultiMatrix()              */
/*  PURPOSE: To provide user to select more than one     */
/*          selected matrix, set total window number, and */
/*          set the first x, y label at positive part of  */
/*          the axes, and get those input strings.        */
/*  LOGIC: If succeed, the function returns 1, otherwise   */
/*         return 0.                                      */
/*******************************************************/
int SelectMultiMatrix()
{
  char string[50],ch;
  int  i,x,y,j=0,k;

  outtextxy(18,138,
  "(2) Enter: total window no. on screen (1, 2 or 4): ");
  GetUserInput(string,18,138,52);
  WindowNo=atoi(string);
```

```
if((WindowNo!=1)&&(WindowNo!=2)&&(WindowNo!=4))
{
  ErrorMessage();
  return(0);
}
outtextxy(18,158,"(3) Selected real matrices as the form:");
setcolor(LIGHTGREEN);
outtextxy(18,168,"        A = | a    b |");
outtextxy(18,178,"            | c    d |");
setcolor(WHITE);
for(i=0;i<WindowNo;i++)
{
  outtextxy(18,198+j,"    Enter: a = ");
  GetUserInput(string,18,198+j,15);
  strcpy(SelectM[0],string);
  outtextxy(318,198+j,"    Enter: b = ");
  GetUserInput(string,318,198+j,15);
  strcpy(SelectM[1],string);
  outtextxy(18,218+j,"    Enter: c = ");
  GetUserInput(string,18,218+j,15);
  strcpy(SelectM[2],string);
  outtextxy(318,218+j,"    Enter: d = ");
  GetUserInput(string,318,218+j,15);
  strcpy(SelectM[3],string);
  outtextxy(18,228+j,
     "------------------------------------
      ----------------------------");
  if(CheckSelectMatrix()!=1)
  {
    ErrorMessage();
    return(0);
  }
  if(i==0)
    for(k=0;k<4;k++)
      strcpy(SelectUM1[k],SelectM[k]);
  else if(i==1)
    for(k=0;k<4;k++)
      strcpy(SelectUM2[k],SelectM[k]);
  else if(i==2)
    for(k=0;k<4;k++)
      strcpy(SelectUM3[k],SelectM[k]);
  else if(i==3)
    for(k=0;k<4;k++)
      strcpy(SelectUM4[k],SelectM[k]);
  j+=40;
}
j-=40;
outtextxy(18,248+j,
"(4) Enter: x value of first labeled point on x axis: ");
outtextxy(18+11*textwidth("S"),258+j,
          "(positive real number)");
GetUserInput(string,18,248+j,53);
ChartoDouble(string,&FirstXLabel);
if(FirstXLabel<0)
```

```
    FirstXLabel*=(-1);
  if(CheckSelectXYLabel(string)!=1)
  {
    ErrorMessage();
    return(0);
  }
  outtextxy(18,278+j,
  "(5) Enter: y value of first labeled point on y axis: ");
  outtextxy(18+11*textwidth("S"),288+j,
             "(positive real number)");
  GetUserInput(string,18,278+j,53);
  ChartoDouble(string,&FirstYLabel);
  if(FirstYLabel<0)
    FirstYLabel*=(-1);
  if(CheckSelectXYLabel(string)!=1)
  {
    ErrorMessage();
    return(0);
  }
  outtextxy(18,308+j,
  "(6) Enter: condition number (positive real number): ");
  GetUserInput(string,18,308+j,52);
  ChartoDouble(string,&ConditionNo);
  if(ConditionNo<0)
  {
    ErrorMessage();
    return(0);
  }
  ClearUserScreen();
  return(1);
}

/************************************************************/
/*  FUNCTION NAME: int SelectedMatrix()                     */
/*  PURPOSE: To provide user to select a selected matrix,   */
/*           set total window number, and set the first x,  */
/*           y label at positive part of the axes, and get  */
/*           those input strings.                           */
/*  LOGIC: If succeed, the function returns 1, otherwise    */
/*         return 0.                                        */
/************************************************************/
int SelectedMatrix()
{
  char string[50],ch;
  int  i,x,y,select,multi;

  WindowBox(LowX+2,LowY+2,HighX-2,HighY-2,WHITE);
  setviewport(LowX+2,LowY+2,HighX-2,HighY-2,1);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  settextjustify(LEFT_TEXT,TOP_TEXT);
  setcolor(LIGHTCYAN);
  outtextxy(18,18,"* Setup Variables ---");
  setcolor(WHITE);
  outtextxy(18,48,"(1) Selected matrix as the form:");
```

```
setcolor(LIGHTGREEN);
outtextxy(18,68,"        A = | a    b |");
outtextxy(18,78,"            | c    d |");
setcolor(WHITE);
outtextxy(18,98,"    Enter: a = ");
GetUserInput(string,18,98,15);
strcpy(SelectM[0],string);
outtextxy(318,98,"    Enter: b = ");
GetUserInput(string,318,98,15);
strcpy(SelectM[1],string);
outtextxy(18,118,"    Enter: c = ");
GetUserInput(string,18,118,15);
strcpy(SelectM[2],string);
outtextxy(318,118,"    Enter: d = ");
GetUserInput(string,318,118,15);
strcpy(SelectM[3],string);
select=CheckSelectMatrix();
if((select!=1)&&(select!=2))
{
  ErrorMessage();
  return(0);
}
else if(select==1)
{
  outtextxy(18,148,
  "(2) Enter: total window no. on screen (1 or 2): ");
  GetUserInput(string,18,148,48);
  WindowNo=atoi(string);
  if((WindowNo!=1)&&(WindowNo!=2))
  {
    ErrorMessage();
    return(0);
  }
  outtextxy(18,178,
  "(3) Enter: x value of first labeled point on x axis: ");
  outtextxy(18+11*textwidth("S"),198,
            "(positive real number)");
  GetUserInput(string,18,178,53);
  ChartoDouble(string,&FirstXLabel);
  if(FirstXLabel<0)
    FirstXLabel*=(-1);
  if(CheckSelectXYLabel(string)!=1)
  {
    ErrorMessage();
    return(0);
  }
  outtextxy(18,218,
  "(4) Enter: y value of first labeled point on y axis: ");
  outtextxy(18+11*textwidth("S"),238,
            "(positive real number)");
  GetUserInput(string,18,218,53);
  ChartoDouble(string,&FirstYLabel);
  if(FirstYLabel<0)
    FirstYLabel*=(-1);
    if(CheckSelectXYLabel(string)!=1)
    {
      ErrorMessage();
      return(0);
    }
    outtextxy(18,258,
    "(5) Enter: condition number (positive real number): ");
    outtextxy(18+11*textwidth("S"),278,
   "(for determining ill-conditioned matrices, default=100)");
    GetUserInput(string,18,258,52);
    ChartoDouble(string,&ConditionNo);
    if(ConditionNo<1)
    {
      ErrorMessage();
      return(0);
    }
    else if(string[0]=='\0')
      ConditionNo=100.000000;
    outtextxy(18,298,
    "(6) Choose: (1) scale matrices by rows, or");
    outtextxy(18,318,
    "            (2) scale matrices by columns, or");
    outtextxy(18,338,"              (3) none.");
    outtextxy(18,358,"--- 1, 2, or 3 --- : ");
    GetUserInput(string,18,358,21);
    ScaleMatrix=atoi(string);
    /* if users press 'Enter' instead of pressing 1, 2, */
    /* or 3, then ScaleMatrix=0 means users didnot want */
    /* to scale the matrix */
    if((ScaleMatrix!=1)&&(ScaleMatrix!=2)
       &&(ScaleMatrix!=3)&&(ScaleMatrix!=0))
    {
      ErrorMessage();
      return(0);
    }
    ClearUserScreen();
    return(1);
  }
  else if(select==2)
  {
    multi=SelectMultiMatrix();
    if(multi!=1)
      return(0);
    else
      return(2);
  }
  return(0);
}

/**********************************************************/
/*  FUNCTION NAME: void OptionDrawFigure()               */
/*  PURPOSE: To draw figures after call OptionProcess() and*/
/*           SetupVariable().                             */
/*  LOGIC: Control the figures drawn type by opt parameter.*/
/*         For setup variables function, opt=1.          */
/*         For enlarge figure only function, opt=2.      */
/*         For reduce figure only function, opt=3.       */
/*         Others, opt=4.                                */
/*******************************************************/
void OptionDrawFigure(opt)
int opt;
{
  int x,y,j;
  char ch;
  double a,b,c,d;
  void *svimage[1];

  a=SelectMF[0];
  b=SelectMF[1];
  c=SelectMF[2];
  d=SelectMF[3];
  WindowSetup(WindowNo,1);
  if(WindowNo==1)
  {
    WindowBox(LX1,LY1,HX1,HY1,WHITE);
    DrawFigure(opt,5,b,c,a,d);
    if(opt!=4)
    {
      setviewport(0,0,MaxX,MaxY,1);
      svimage[0]=malloc(imagesize(18,418,639,458));
      getimage(18,418,639,458,svimage[0]);
      settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
      settextjustify(LEFT_TEXT,TOP_TEXT);
      setcolor(YELLOW);
      outtextxy(18,418,"Press any key to continue ...");
      outtextxy(18,438,"or Press 'Q' or 'q' to abort.");
      GetUserOption(&ch,&x,&y,0);
      if((ch!='q')&&(ch!='Q')&&(!IfMouseInWindow(x,y,
      4*MaxX/5,0,MaxX-4*textwidth("Q"),6+textheight("Q"))))
      {
        ClearUserScreen();
        free(svimage[0]);
        DrawFigure(opt,6,a,d,b,c);
      }
      else
      {
        putimage(18,418,svimage[0],COPY_PUT);
        free(svimage[0]);
      }
    }
  }
  else if(WindowNo==2)
  {
    WindowBox(LX1,LY1,HX1,HY1,WHITE);
    WindowBox(LX2,LY2,HX2,HY2,WHITE);
    DrawFigure(opt,7,b,c,a,d);
    if(opt!=4)
      DrawFigure(opt,8,a,d,b,c);
```

```
        else if(opt==4)
        {
          for(j=0;j<4;j++)
            strcpy(SelectM[j],SelectUM2[j]);
          for(j=0;j<4;j++)
            ChartoDouble(SelectM[j],&SelectMF[j]);
          DrawFigure(opt,8,SelectMF[1],SelectMF[2],
                     SelectMF[0],SelectMF[3]);
        }
      }
      else if(WindowNo==4)
      {
        WindowBox(LX1,LY1,HX1,HY1,WHITE);
        WindowBox(LX2,LY2,HX2,HY2,WHITE);
        WindowBox(LX3,LY3,HX3,HY3,WHITE);
        WindowBox(LX4,LY4,HX4,HY4,WHITE);
        DrawFigure(opt,1,b,c,a,d);
        for(j=0;j<4;j++)
          strcpy(SelectM[j],SelectUM2[j]);
        for(j=0;j<4;j++)
          ChartoDouble(SelectM[j],&SelectMF[j]);
        DrawFigure(opt,2,SelectMF[1],SelectMF[2],
                   SelectMF[0],SelectMF[3]);
        for(j=0;j<4;j++)
          strcpy(SelectM[j],SelectUM3[j]);
        for(j=0;j<4;j++)
          ChartoDouble(SelectM[j],&SelectMF[j]);
        DrawFigure(opt,3,SelectMF[1],SelectMF[2],
                   SelectMF[0],SelectMF[3]);
        for(j=0;j<4;j++)
          strcpy(SelectM[j],SelectUM4[j]);
        for(j=0;j<4;j++)
          ChartoDouble(SelectM[j],&SelectMF[j]);
        DrawFigure(opt,4,SelectMF[1],SelectMF[2],
                   SelectMF[0],SelectMF[3]);
      }
    }

/**********************************************************/
/*  FUNCTION NAME: void SetupVariable()                   */
/*  PURPOSE: To get the settings for selected matrices,   */
/*           window number, x and y labels from user input */
/*           by keyboard.                                 */
/*  LOGIC: If SelectedMatrix() return 1, draw figures by  */
/*         call OptionDrawFigure(1). If SelectedMatrix()  */
/*         return 2, means user choose an unknown matrix, */
/*         draw figure by call OptionDrawFigure(4).       */
/**********************************************************/
void SetupVariable()
{
  char ch;
  int j,select;
  double a,b,c,d,temp;
  setviewport(0,0,MaxX,MaxY,1);
  select=SelectedMatrix();
  if(select==0)
    return;
  else if(select==1)
  {
    for(j=0;j<4;j++)
      ChartoDouble(SelectM[j],&SelectMF[j]);
    OptionDrawFigure(1);
    Efactor=1;
    Rfactor=1;
    Initial=1;
  }
  else if(select==2)
  {
    for(j=0;j<4;j++)
      strcpy(SelectM[j],SelectUM1[j]);
    for(j=0;j<4;j++)
      ChartoDouble(SelectM[j],&SelectMF[j]);
    OptionDrawFigure(4);
    Initial=0;
  }
}

/************************************************************/
/*  FUNCTION NAME: void EnlargeFigureOnly()                 */
/*  PURPOSE: To draw enlarged figures for current figures. */
/*  LOGIC: Set FirstXLabel, FirstYLabel to be divided by a */
/*         factor, then call OptionDrawFigure(2). Reset    */
/*         FirstXLabel, FirstYLabel to be old value.       */
/************************************************************/
void EnlargeFigureOnly()
{
  double fxlabel,fylabel;

  fxlabel=FirstXLabel; /* store original FirstXLabel */
  fylabel=FirstYLabel; /* store original FirstYLabel */
  FirstXLabel/=Efactor;
  FirstYLabel/=Efactor;
  OptionDrawFigure(2);
  FirstXLabel=fxlabel; /* reset original FirstXLabel back */
  FirstYLabel=fylabel; /* reset original FirstYLabel back */
}

/************************************************************/
/*  FUNCTION NAME: void EnlargeReduceOldFigure()            */
/*  PURPOSE: To draw enlarged figures or reduced figures   */
/*           for current figures.                          */
/*  LOGIC: Control the figures drawn type by opt parameter.*/
/*         For enlarge figures, opt=1.                     */
/*         For reduce figures, opt=2.                      */
/************************************************************/
void EnlargeReduceOldFigure(opt)
int opt;
```

```
{
  double a,b,c,d,fxlabel,fylabel;

  a=SelectMF[0];
  b=SelectMF[1];
  c=SelectMF[2];
  d=SelectMF[3];
  WindowSetup(4,2);
  WindowBox(LX1,LY1,HX1,HY1,WHITE);
  WindowBox(LX2,LY2,HX2,HY2,WHITE);
  WindowBox(LX3,LY3,HX3,HY3,WHITE);
  WindowBox(LX4,LY4,HX4,HY4,WHITE);
  DrawFigure(opt+1,1,b,c,a,d);  /* draw old figure */
  DrawFigure(opt+1,2,a,d,b,c);
  fxlabel=FirstXLabel; /* store original FirstXLabel */
  fylabel=FirstYLabel; /* store original FirstYLabel */
  if(opt==1)  /* draw enlarge figure */
  {
    FirstXLabel/=Efactor;
    FirstYLabel/=Efactor;
  }
  else if(opt==2) /* draw reduce figure */
  {
    FirstXLabel*=Rfactor;
    FirstYLabel*=Rfactor;
  }
  DrawFigure(opt+1,3,b,c,a,d);
  DrawFigure(opt+1,4,a,d,b,c);
  FirstXLabel=fxlabel; /* reset original FirstXLabel back */
  FirstYLabel=fylabel; /* reset original FirstYLabel back */
}

/**********************************************************/
/*  FUNCTION NAME: void EnlargeReduceMenu()               */
/*  PURPOSE: To draw a menu box for Enlargement() and     */
/*           Reduction().                                 */
/**********************************************************/
void EnlargeReduceMenu(string,xsize,ysize,color1,color2)
char string[];
int xsize,ysize,color1,color2;
{
  int i=strlen(string);

  DrawBar(0,0,xsize,ysize,LIGHTGRAY);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  settextjustify(LEFT_TEXT,TOP_TEXT);
  WindowBox(5,4,250,4*textheight("O")+22,BLACK);
  /* set hightlighting of "Enlarge Only" box */
  DrawBar(6,8,249,26,color1);
  setcolor(BLACK);
  outtextxy(16,14,string);
  outtextxy(16+i*textwidth("E"),14," Figure");
  setcolor(RED);
  outtextxy(16+(i+7)*textwidth("E"),14," O");
```

```
    setcolor(BLACK);
    outtextxy(16+(i+9)*textwidth("E"),14,"nly");
    DrawBar(6,30,249,44+textheight("E"),color2);
    setcolor(BLACK);
    outtextxy(16,30+textheight("E"),string);
    outtextxy(16+i*textwidth("E"),30+textheight("E"),
              " Figure");
    outtextxy(16+(i+7)*textwidth("E"),30+textheight("E"),
              " / Old");
    setcolor(RED);
    outtextxy(16+(i+13)*textwidth("E"),30+textheight("E"),
              " F");
    setcolor(BLACK);
    outtextxy(16+(i+15)*textwidth("E"),30+textheight("E"),
              "igure");
  }

  /************************************************************/
  /*  FUNCTION NAME: void OptionMenu()                        */
  /*  PURPOSE: To draw a menu box for OptionProcess().        */
  /************************************************************/
  void OptionMenu(xsize,ysize,color1,color2,color3)
  int xsize,ysize,color1,color2,color3;
  {
    DrawBar(0,0,xsize,ysize,LIGHTGRAY);
    settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
    settextjustify(LEFT_TEXT,TOP_TEXT);
    WindowBox(5,4,165,6*textheight("O")+30,BLACK);
    /* set hightlighting of "Setup Variables" box */
    DrawBar(6,8,164,26,color1);
    if(Initial>=0)
      setcolor(RED);
    outtextxy(16,14,"S");
    if(Initial>=0)
      setcolor(BLACK);
    outtextxy(16+textwidth("O"),14,"etup Variables");
    DrawBar(6,30,164,44+textheight("O"),color2);
    DrawBar(6,48+textheight("O"),164,51+3*textheight("O"),
         color3);
    if(Initial>=1)
      setcolor(RED);
    else
      setcolor(DARKGRAY);
    outtextxy(16,30+textheight("O"),"E");
    outtextxy(16,46+2*textheight("O"),"R");
    if(Initial>=1)
      setcolor(BLACK);
    else
      setcolor(DARKGRAY);
    outtextxy(16+textwidth("O"),30+textheight("O"),
              "nlarge      >>");
    outtextxy(16+textwidth("O"),46+2*textheight("O"),
              "educe       >>");
  }
```

```
/*********************************************************/
/*  FUNCTION NAME: void HelpMenu()                       */
/*  PURPOSE: To draw a menu box for HelpProcess().       */
/*********************************************************/
void HelpMenu(xsize,ysize,color1,color2,color3)
int xsize,ysize,color1,color2,color3;
{
  DrawBar(0,0,xsize,ysize,LIGHTGRAY);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  settextjustify(LEFT_TEXT,TOP_TEXT);
  WindowBox(5,4,170,6*textheight("H")+32,BLACK);
  /* set hightlighting of "Help About Demo" box */
  DrawBar(6,8,169,26,color1);
  setcolor(BLACK);
  outtextxy(16,14,"Help About ");
  setcolor(RED);
  outtextxy(16+11*textwidth("H"),14,"D");
  setcolor(BLACK);
  outtextxy(16+12*textwidth("H"),14,"emo");
  DrawBar(6,30,169,44+textheight("H"),color2);
  setcolor(BLACK);
  outtextxy(16,30+textheight("H"),"Help About ");
  setcolor(RED);
  outtextxy(16+11*textwidth("H"),30+textheight("H"),"O");
  setcolor(BLACK);
  outtextxy(16+12*textwidth("H"),30+textheight("H"),
            "ptions");
  DrawBar(6,48+textheight("H"),169,60+2*textheight("H"),
       color3);
  setcolor(BLACK);
  outtextxy(16,46+2*textheight("H"),"Help About ");
  setcolor(RED);
  outtextxy(16+11*textwidth("H"),46+2*textheight("H"),"Q");
  setcolor(BLACK);
  outtextxy(16+12*textwidth("H"),46+2*textheight("H"),
            "uit");
}

/*********************************************************/
/*  FUNCTION NAME: void GetProcBox()                     */
/*  PURPOSE: To get and save a menu box for functions that */
/*           need it, such as OptionProcess().           */
/*********************************************************/
void GetProcBox(x1,y1,x2,y2,xsize,ysize,opt)
int x1,y1,x2,y2,*xsize,*ysize,opt;
{
  setviewport(x1,y1,x2,y2,1);
  *xsize=x2-x1;
  *ysize=y2-y1;
  if(opt==1) /* for OptionProcess() to store option box */
  {
    opimage[0]=malloc(imagesize(0,0,*xsize,*ysize));
    getimage(0,0,*xsize,*ysize,opimage[0]);
  }
  else
  {
    voidimage[0]=malloc(imagesize(0,0,*xsize,*ysize));
    getimage(0,0,*xsize,*ysize,voidimage[0]);
  }
  if(opt==1)    /* for OptionProcess() */
    OptionMenu(*xsize,*ysize,GREEN,LIGHTGRAY,LIGHTGRAY);
  else if(opt==2)  /* for HelpProcess() */
    HelpMenu(*xsize,*ysize,GREEN,LIGHTGRAY,LIGHTGRAY);
  else if(opt==3)  /* for Enlargement() */
    EnlargeReduceMenu("Enlarge",*xsize,*ysize,
                        GREEN,LIGHTGRAY);
  else if(opt==4)  /* for Reduction() */
    EnlargeReduceMenu("Reduce",*xsize,*ysize,GREEN,LIGHTGRAY);
  getviewsettings(&viewp);
}

/*******************************************************/
/*  FUNCTION NAME: void RemoveProcBox()               */
/*  PURPOSE: To remove and free a menu box for functions   */
/*           that need it, such as OptionProcess().       */
/*******************************************************/
void RemoveProcBox(x1,y1,x2,y2,xsize,ysize,opt,flag)
int x1,y1,x2,y2,xsize,ysize,opt,flag;
{
  setviewport(viewp.left,viewp.top,viewp.right,
              viewp.bottom,1);
  if(opt==1)  /* for OptionProcess() - enlarge item */
    OptionMenu(xsize,ysize,LIGHTGRAY,GREEN,LIGHTGRAY);
  else if(opt==2) /* for OptionProcess() - reduce item */
    OptionMenu(xsize,ysize,LIGHTGRAY,LIGHTGRAY,GREEN);
  else if(opt==3) /* for HelpProcess() - option item */
    HelpMenu(xsize,ysize,LIGHTGRAY,GREEN,LIGHTGRAY);
  else if(opt==4) /* for HelpProcess() - quit item */
    HelpMenu(xsize,ysize,LIGHTGRAY,LIGHTGRAY,GREEN);
  else if(opt==5) /* for Enlargement() */
    EnlargeReduceMenu("Enlarge ",xsize,ysize,LIGHTGRAY,GREEN);
  else if(opt==6) /* for Reduction() */
    EnlargeReduceMenu("Reduce ",xsize,ysize,LIGHTGRAY,GREEN);
  delay(300);
  if((opt>2)&&(opt<9)&&(flag==0))
  {
    putimage(0,0,voidimage[0],COPY_PUT);/* remove option */
    free(voidimage[0]);          /* box of HelpProcess() */
  }
  else if((opt==0)&&(flag==1))
  {
    setviewport(x1,y1,x2,y2,1);  /* remove option box of */
    putimage(0,0,opimage[0],COPY_PUT);/* OptionProcess() */
    free(opimage[0]);
  }
  else if((opt>=5)&&(opt<=8)&&(flag==1))
  {    /* remove option box of Enlargement() or Reduce() */
    putimage(0,0,voidimage[0],COPY_PUT);
    free(voidimage[0]);
    setviewport(x1,y1,x2,y2,1);  /* remove option box of */
    putimage(0,0,opimage[0],COPY_PUT);/* OptionProcess() */
    free(opimage[0]);
  }
}

/*********************************************************/
/*  FUNCTION NAME: void Enlargement()                    */
/*  PURPOSE: To draw enlarged figures for current figures */
/*  LOGIC: If users key in 'o' or 'O', show enlarged      */
/*         figures only.  If users key in 'f' or 'F', show */
/*         enlarged figures and old figures both in one   */
/*         screen.                                        */
/*********************************************************/
void Enlargement()
{
  int xsize,ysize,x,y,i=1;
  char ch;

  GetProcBox(2*MaxX/5,5*textheight("E")+32,
     4*MaxX/5,9*textheight("E")+58,&xsize,&ysize,3);
  while(i)
  {
    GetUserOption(&ch,&x,&y,2);
    if((ch=='o')||(ch=='O')||(IfMouseInWindow(x,y,2*MaxX/5,
    5*textheight("E")+32,4*MaxX/5,6*textheight("E")+48)))
    {
      RemoveProcBox(2*MaxX/5-10,textheight("E")+6,
            2*MaxX/3-10,7*textheight("E")+40,xsize,ysize,7,1);
      ClearUserScreen();
      EnlargeFigureOnly();
      i=0;
    }
    else if((ch=='f')||(ch=='F')||(IfMouseInWindow(x,y,
    2*MaxX/5,6*textheight("E")+49,4*MaxX/5,
    7*textheight("E")+64)))
    {
      RemoveProcBox(2*MaxX/5-10,textheight("E")+6,
            2*MaxX/3-10,7*textheight("E")+40,xsize,ysize,5,1);
      ClearUserScreen();
      EnlargeReduceOldFigure(1);
      i=0;
    }
    else if(((ch!='o')&&(ch!='O')&&(ch!='f')&&(ch!='F'))||
    (!IfMouseInWindow(x,y,2*MaxX/5,5*textheight("E")+32,
    4*MaxX/5,9*textheight("E")+58)))
    {
      RemoveProcBox(2*MaxX/5-10,textheight("E")+6,
            2*MaxX/3-10,7*textheight("E")+40,xsize,ysize,8,1);
      i=0;
    }
  }
}
```

```
/*********************************************************/
/*  FUNCTION NAME: void ReduceFigureOnly()               */
/*  PURPOSE: To draw reduced figures for current figures. */
/*  LOGIC: Set FirstXLabel, FirstYLabel to be timed by a  */
/*         factor, then call OptionDrawFigure(3). Reset   */
/*         FirstXLabel, FirstYLabel to be old value.      */
/*********************************************************/
void ReduceFigureOnly()
{
  double fxlabel,fylabel;

  fxlabel=FirstXLabel; /* store original FirstXLabel */
  fylabel=FirstYLabel; /* store original FirstYLabel */
  FirstXLabel*=Rfactor;
  FirstYLabel*=Rfactor;
  OptionDrawFigure(3);
  FirstXLabel=fxlabel; /* reset original FirstXLabel back */
  FirstYLabel=fylabel; /* reset original FirstYLabel back */
}

/*********************************************************/
/*  FUNCTION NAME: void Reduction()                      */
/*  PURPOSE: To draw reduced figures for current figures  */
/*  LOGIC: If users key in 'o' or 'O', show reduced       */
/*         figures only.  If users key in 'f' or 'F', show */
/*         reduced figures and old figures both in one    */
/*         screen.                                        */
/*********************************************************/
void Reduction()
{
  int xsize,ysize,x,y,i=1;
  char ch;

  GetProcBox(2*MaxX/5,8*textheight("R")+26,
     4*MaxX/5,12*textheight("R")+52,&xsize,&ysize,4);
  while(i)
  {
    GetUserOption(&ch,&x,&y,2);
    if((ch=='o')||(ch=='O')||(IfMouseInWindow(x,y,2*MaxX/5,
    8*textheight("R")+26,4*MaxX/5,9*textheight("R")+42)))
    {
      RemoveProcBox(2*MaxX/5-10,textheight("E")+6,
           2*MaxX/3-10,7*textheight("E")+40,xsize,ysize,7,1);
      ClearUserScreen();
      ReduceFigureOnly();
      i=0;
    }
    else if((ch=='f')||(ch=='F')||(IfMouseInWindow(x,y,
    2*MaxX/5,9*textheight("R")+43,4*MaxX/5,
    10*textheight("R")+58)))
    {
      RemoveProcBox(2*MaxX/5-10,textheight("E")+6,
           2*MaxX/3-10,7*textheight("E")+40,xsize,ysize,6,1);
      ClearUserScreen();
      EnlargeReduceOldFigure(2);
      i=0;
    }
    else if(((ch!='o')&&(ch!='O')&&(ch!='f')&&(ch!='F'))||
    (!IfMouseInWindow(x,y,2*MaxX/5,8*textheight("R")+26,
    4*MaxX/5,12*textheight("R")+52)))
    {
      RemoveProcBox(2*MaxX/5-10,textheight("E")+6,
            2*MaxX/3-10,7*textheight("E")+40,xsize,ysize,8,1);
      i=0;
    }
  }
}

/*********************************************************/
/*  FUNCTION NAME: void OptionProcess()                  */
/*  PURPOSE: To draw figures after get the selected matrix */
/*           and required information.                   */
/*  LOGIC: If users key in 's' or 'S', ask users to key in */
/*         required informantion.  If users key in 'e' or  */
/*         'E', call Enlargement() to display enlarged    */
/*         figures.  If users key in 'r' or 'R', call     */
/*         Reduction() to draw reduced figures.           */
/*********************************************************/
void OptionProcess()
{
  int xsize,ysize,x,y,i=1;
  char ch;

  GetProcBox(2*MaxX/5-10,textheight("O")+6,
     2*MaxX/3-10,7*textheight("O")+40,&xsize,&ysize,1);
  while(i)
  {
    GetUserOption(&ch,&x,&y,2);
    if(((ch=='s')||(ch=='S')||(IfMouseInWindow(x,y,
    2*MaxX/5-10,textheight("O")+7,2*MaxX/3-10,
    2*textheight("O")+22)))&&(Initial>=0))
    {
      RemoveProcBox(2*MaxX/5-10,textheight("O")+6,
            2*MaxX/3-10,7*textheight("O")+40,xsize,ysize,0,1);
      ClearUserScreen();
      SetupVariable();
      i=0;
    }
    else if(((ch=='e')||(ch=='E')||(IfMouseInWindow(x,y,
    2*MaxX/5-10,2*textheight("O")+23,2*MaxX/3-10,
    3*textheight("O")+38)))&&(Initial>=1))
    {
      RemoveProcBox(2*MaxX/5-10,textheight("O")+6,
            2*MaxX/3-10,7*textheight("O")+40,xsize,ysize,1,0);
      Efactor++;
      Enlargement();
      Initial=2;
      i=0;
    }
    else if(((ch=='r')||(ch=='R')||(IfMouseInWindow(x,y,
    2*MaxX/5-10,3*textheight("O")+39,2*MaxX/3-10,
    4*textheight("O")+54)))&&(Initial>=1))
    {
      RemoveProcBox(2*MaxX/5-10,textheight("O")+6,
           2*MaxX/3-10,7*textheight("O")+40,xsize,ysize,2,0);
      Rfactor++;
      Reduction();
      Initial=3;
      i=0;
    }
    else
    if(((ch!='s')&&(ch!='S')&&(ch!='e')&&(ch!='E')&&
        (ch!='r')&&(ch!='R'))||(!IfMouseInWindow(x,y,
       2*MaxX/5-10,textheight("O")+6,2*MaxX/3-10,
       7*textheight("O")+40)))
    {
      RemoveProcBox(2*MaxX/5-10,textheight("O")+6,
           2*MaxX/3-10,7*textheight("O")+40,xsize,ysize,0,1);
      i=0;
    }
    DisplayMouseCursor();
  }
}

/*************************************************************/
/*  FUNCTION NAME: void HelpDemo()                           */
/*  PURPOSE: To display help information about Demo          */
/*           function on screen.                              */
/*************************************************************/
void HelpDemo()
{
  setviewport(0,0,MaxX,MaxY,1);
  WindowBox(LowX+2,LowY+2,HighX-2,HighY-2,WHITE);
  setviewport(LowX+2,LowY+2,HighX-2,HighY-2,1);
  setcolor(LIGHTCYAN);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  settextjustify(LEFT_TEXT,TOP_TEXT);
  outtextxy(18,18,"* Help About Demo ---");
  setcolor(WHITE);
  outtextxy(18,48,"(1) Press ");
  setcolor(LIGHTRED);
  outtextxy(18+10*textwidth("A"),48,"d ");
  setcolor(WHITE);
  outtextxy(18+12*textwidth("A"),48,"or ");
  setcolor(LIGHTRED);
  outtextxy(18+15*textwidth("A"),48,"D ");
  setcolor(WHITE);
  outtextxy(18+17*textwidth("A"),48,
            "to execute Demo function.");
  setcolor(LIGHTGREEN);
  outtextxy(18,78,"(2) Demo function:");
  setcolor(WHITE);
  outtextxy(18,98," -- To demonstrate the 4-D space of 2 x 2
                   matrices by slicing it into a");
  outtextxy(18+4*textwidth("D"),118,"series of 2-D images,
         emphasizing singular and ill-conditioned matrices");
  outtextxy(18+4*textwidth("D"),138,"by highlighting.
      Eigenvectors of a selected matrix are also displayed");
  outtextxy(18+4*textwidth("D"),158,"graphically.");
  outtextxy(18,178," -- Need a input file (named wndw.dat in
            drive B) which gives the");
  outtextxy(18+4*textwidth("D"),198,"components of a selected
              matrix as the ordering: a b c d, for a matrix");
  outtextxy(18+4*textwidth("D"),218,"A=|a b|");
  outtextxy(18+4*textwidth("D"),228,"  |c d|");
}

/************************************************************/
/*  FUNCTION NAME: void HelpOption()                        */
/*  PURPOSE: To display help information about Option      */
/*           function on screen.                            */
/************************************************************/
void HelpOption()
{
  setviewport(0,0,MaxX,MaxY,1);
  WindowBox(LowX+2,LowY+2,HighX-2,HighY-2,WHITE);
  setviewport(LowX+2,LowY+2,HighX-2,HighY-2,1);
  setcolor(LIGHTCYAN);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  settextjustify(LEFT_TEXT,TOP_TEXT);
  outtextxy(18,18,"* Help About Option ---");
  setcolor(WHITE);
  outtextxy(18,48,"(1) Press ");
  setcolor(LIGHTRED);
  outtextxy(18+10*textwidth("A"),48,"o ");
  setcolor(WHITE);
  outtextxy(18+12*textwidth("A"),48,"or ");
  setcolor(LIGHTRED);
  outtextxy(18+15*textwidth("A"),48,"O ");
  setcolor(WHITE);
  outtextxy(18+17*textwidth("A"),48,
            "to execute Option function.");
  setcolor(LIGHTGREEN);
  outtextxy(18,78,"(2) Option function:");
  setcolor(LIGHTRED);
  outtextxy(18,98,"    s or S :");
  setcolor(WHITE);
  outtextxy(18+12*textwidth("A"),98," setup variables");
  outtextxy(18,118,"    - To set the components of a selected
            matrix and setup the relative");
  outtextxy(18+6*textwidth("D"),138,
            "variables to draw the figures.");
  setcolor(LIGHTRED);
  outtextxy(18,158,"    e or E : ");
  setcolor(WHITE);
  outtextxy(18+12*textwidth("A"),158,
            " show enlarged figures");
  outtextxy(18,178,"    - Only executed after executing Setup
            Variables function.");
  outtextxy(18,198,"    - To enlarge current figures which are
            drawn after user setup variables.");
  setcolor(LIGHTRED);
  outtextxy(18+12*textwidth("A"),218,"   o or O : ");
  setcolor(WHITE);
  outtextxy(18+24*textwidth("A"),218,
            " show enlarged figures only.");
  setcolor(LIGHTRED);
  outtextxy(18+12*textwidth("A"),238,"   f or F : ");
  setcolor(WHITE);
  outtextxy(18+24*textwidth("A"),238,
  " show enlarged figures and original figures.");
  setcolor(LIGHTRED);
  outtextxy(18,258,"    r or R : ");
  setcolor(WHITE);
  outtextxy(18+12*textwidth("A"),258," show reduced figures");
  outtextxy(18,278,"    - Only executed after executing Setup
            Variables function.");
  outtextxy(18,298,"    - To reduce current figures which are
            drawn after user setup variables.");
  setcolor(LIGHTRED);
  outtextxy(18+12*textwidth("A"),318,"   o or O : ");
  setcolor(WHITE);
  outtextxy(18+24*textwidth("A"),318,
            " show reduced figures only.");
  setcolor(LIGHTRED);
  outtextxy(18+12*textwidth("A"),338,"   f or F : ");
  setcolor(WHITE);
  outtextxy(18+24*textwidth("A"),338,
  " show reduced figures and original figures.");
}

/************************************************************/
/*  FUNCTION NAME: void HelpQuit()                          */
/*  PURPOSE: To display help information about Quit         */
/*           function on screen.                            */
/************************************************************/
void HelpQuit()
{
  setviewport(0,0,MaxX,MaxY,1);
  WindowBox(LowX+2,LowY+2,HighX-2,HighY-2,WHITE);
  setviewport(LowX+2,LowY+2,HighX-2,HighY-2,1);
  setcolor(LIGHTCYAN);
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  settextjustify(LEFT_TEXT,TOP_TEXT);
  outtextxy(18,18,"* Help About Quit ---");
  setcolor(WHITE);
  outtextxy(18,48,"(1) Press ");
  setcolor(LIGHTRED);
  outtextxy(18+10*textwidth("A"),48,"q ");
  setcolor(WHITE);
  outtextxy(18+12*textwidth("A"),48,"or ");
  setcolor(LIGHTRED);
  outtextxy(18+15*textwidth("A"),48,"Q ");
  setcolor(WHITE);
  outtextxy(18+17*textwidth("A"),48,
            "to execute Quit function.");
  setcolor(LIGHTGREEN);
  outtextxy(18,78,"(2) Quit function:");
  setcolor(WHITE);
  outtextxy(18,98,
       " -- To exit a certain function of the package.");
  outtextxy(18,118," -- To exit this package.");
}

/***********************************************************/
/*  FUNCTION NAME: void HelpProcess()                      */
/*  PURPOSE: To display help information about Demo,       */
/*           Option and Quit function on screen.           */
/*  LOGIC: If users key in 'd' or 'D', display help        */
/*         information about demo.  If users key in 'o' or */
/*         'O', display help information about option.  If */
/*         users key in 'q' or 'Q', display help           */
/*         information about quit.                         */
/***********************************************************/
void HelpProcess()
{
  int xsize,ysize,x,y,i=1;
  char ch;

  GetProcBox(3*MaxX/5-10,textheight("H")+6,
     6*MaxX/7,7*textheight("H")+42,&xsize,&ysize,2);
  while(i)
  {
    GetUserOption(&ch,&x,&y,2);
    if((ch=='d')||(ch=='D')||(IfMouseInWindow(x,y,
    3*MaxX/5-10,textheight("D")+7,6*MaxX/7,
    2*textheight("D")+22)))
    {
      RemoveProcBox(3*MaxX/5-10,textheight("O")+6,
            6*MaxX/7,7*textheight("O")+42,xsize,ysize,7,0);
      ClearUserScreen();
      HelpDemo();
      i=0;
    }
    else if((ch=='o')||(ch=='O')||(IfMouseInWindow(x,y,
    3*MaxX/5-10,2*textheight("O")+23,6*MaxX/7,
    3*textheight("O")+38)))
    {
      RemoveProcBox(3*MaxX/5-10,textheight("O")+6,
            6*MaxX/7,7*textheight("O")+42,xsize,ysize,3,0);
      ClearUserScreen();
      HelpOption();
      i=0;
    }
    else if((ch=='q')||(ch=='Q')||(IfMouseInWindow(x,y,
    3*MaxX/5-10,3*textheight("Q")+39,6*MaxX/7,
    4*textheight("Q")+54)))
    {
      RemoveProcBox(3*MaxX/5-10,textheight("O")+6,
            6*MaxX/7,7*textheight("O")+42,xsize,ysize,4,0);
      HelpMenu(xsize,ysize,LIGHTGRAY,LIGHTGRAY,GREEN);
      ClearUserScreen();
      HelpQuit();
      i=0;
    }
    else if(((ch!='d')&&(ch!='D')&&(ch!='o')&&(ch!='O')
    &&(ch!='q')&&(ch!='Q'))||(!IfMouseInWindow(x,y,
    3*MaxX/5-10,textheight("H")+6,6*MaxX/7,
    7*textheight("H")+42)))
    {
      RemoveProcBox(3*MaxX/5-10,textheight("O")+6,
            6*MaxX/7,7*textheight("O")+42,xsize,ysize,8,0);
      i=0;
    }
    DisplayMouseCursor();
  }
}

/*************************************************************/
/*  FUNCTION NAME: void Interface()                          */
/*  PURPOSE : To communicate with the user and to get the    */
/*            user's response.                               */
/*  LOGIC : Shows a user interface and stores the user's     */
/*          response to matrices. If users key in 'd' or     */
/*          'D', execute demo function. If users key in      */
/*          'o' or 'O', execute option function. If users    */
/*          key in 'h' or 'H', execute help function. If     */
/*          users key in 'q' or 'Q', execute quit function.*/
/*************************************************************/
void Interface()
{
  char ch;
  int x,y,i=1;

  while(i)
  {
    ConditionNo=100.000000;
    ScaleMatrix=0;
    GetUserOption(&ch,&x,&y,1);
    if((ch=='d')||(ch=='D')||(IfMouseInWindow(x,y,MaxX/5,0,
    2*MaxX/5-4*textwidth("D"),6+textheight("D"))))
    {
      Menu(GREEN,LIGHTGRAY,LIGHTGRAY,LIGHTGRAY);
      delay(300);
      DemoProcess();
    }
    else if((ch=='o')||(ch=='O')||(IfMouseInWindow(x,y,
    2*MaxX/5,0,3*MaxX/5-4*textwidth("O"),6+textheight("O"))))
    {
      Menu(LIGHTGRAY,GREEN,LIGHTGRAY,LIGHTGRAY);
      OptionProcess();
    }
    else if((ch=='h')||(ch=='H')||(IfMouseInWindow(x,y,
    3*MaxX/5,0,4*MaxX/5-4*textwidth("H"),6+textheight("H"))))
    {
      Menu(LIGHTGRAY,LIGHTGRAY,GREEN,LIGHTGRAY);
      HelpProcess();
    }
    else if((ch=='q')||(ch=='Q')||(IfMouseInWindow(x,y,
    4*MaxX/5,0,MaxX-4*textwidth("Q"),6+textheight("Q"))))
    {
      Menu(LIGHTGRAY,LIGHTGRAY,LIGHTGRAY,GREEN);
      delay(300);
      i=0;
      return;
    }
    else
      DisplayMouseCursor();
    DisplayMouseCursor();
  }
}

/**********************************************************/
/* FUNCTION NAME: main()                                  */
/* PURPOSE: To provide an educational visualization tool  */
/*          based on user queries.                        */
/* LOGIC:Interactively processes a user query by calling  */
/*       appropriate and corresponding functions.         */
/**********************************************************/
main()
{
  Initialization();
  DrawBar(0,0,MaxX,MaxY,BLUE);
  images[0]=malloc(imagesize(0,0,20,20));
  getimage(0,0,20,20,images[0]);
  if(!InitMouse())
  {
    RemoveMouseCursor();
    outtextxy(0,0,"No mouse detected");
    DisplayMouseCursor();
  }
  RemoveMouseCursor();
  Menu(LIGHTGRAY,LIGHTGRAY,LIGHTGRAY,LIGHTGRAY);
  Interface();
  free(images[0]);
  if(!GotMouse) {
    free(ECursor);
    free(ECimage);
  }
  restorecrtmode();
  return 0;
}
```

VITA

Li-Chen Ko

Candidate for the Degree of

Master of Science

Thesis: VISUALIZATION OF THE FOUR-DIMENSIONAL SPACE OF TWO BY TWO MATRICES WITH EMPHASIS ON SINGULAR AND ILL-CONDITIONED CASES

Major Field: Computer Science

Biographical:

Personal Data: Born in Taipei, Taiwan, R.O.C., November 25, 1964, the daughter of Wen-Pin Ko and Hsiu-Chin Hsieh.

Education: Graduated from Taipei First Girls High School, Taipei, Taiwan, R.O.C., in June, 1983; received Bachelor of Science Degree in Pharmacy Sciences from Taipei Medical College, Taipei, Taiwan, R.O.C., in June, 1987; Completed requirements for the Master of Science Degree at Oklahoma State University in December, 1992.