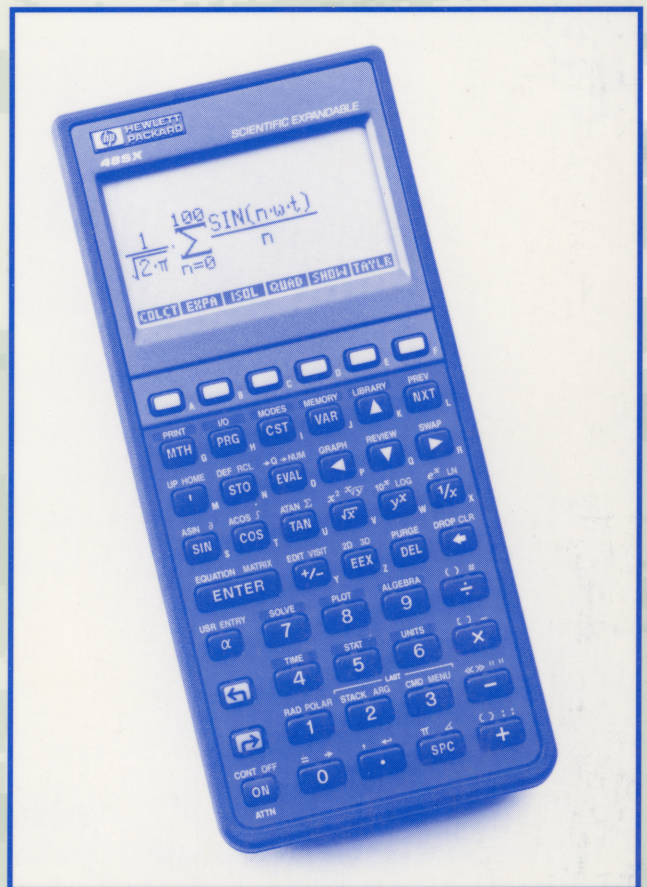


# Calculator Enhancement

---

## for Linear Algebra

A Manual of Applications using HP-48S and  
HP-28S Calculators



D.R. LaTorre, Clemson University





# Calculator Enhancement

---

## for Linear Algebra

**A Manual of Applications using the HP-48S and  
HP-28S Calculators**

**D.R. LaTorre**, Clemson University

**Saunders College Publishing**

A Harcourt Brace Jovanovich College Publisher

Fort Worth Philadelphia San Diego New York Orlando Austin San Antonio

Toronto Montreal London Sydney Tokyo

Copyright© 1992 by Saunders College Publishing

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording or any information storage and retrieval system, without permission in writing from the publisher.

Requests for permission to make copies of any part of the work should be mailed to: Permissions Department, Harcourt Brace Jovanovich, Publishers, 8th Floor, Orlando, Florida 32887.

Printed in the United States of America.

LaTorre: CALCULATOR ENHANCEMENT FOR LINEAR ALGEBRA: A MANUAL OF  
APPLICATIONS USING THE HP-48S and HP-28S  
CALCULATORS, 1/E

ISBN 0-03-092729-3

234 066 987654321



## PREFACE

This book is designed to be used as a course supplement for teaching introductory courses in matrix-oriented linear algebra. Intended to be textbook independent, it presents an appropriate use of the HP-48S and HP-28S graphics programmable calculators to enhance the teaching and learning of the basic concepts of elementary linear algebra.

The material has been developed from classroom experiences in a calculator enhanced, introductory course in matrix-oriented linear algebra taught by the author at Clemson University since mid-1989. The course is at the sophomore level and is taken by students majoring in a variety of fields: the biological and physical sciences, computer information systems and computer science, several engineering disciplines, mathematical sciences, and secondary mathematics education. Ours is not an abstract, proof-oriented course. The main emphasis is on supplying the tools needed to solve problems with matrix techniques and in developing a modest theoretical background in matrix-oriented linear algebra needed for more advanced work in mathematics, science and engineering. We often concentrate on explanations and examples in an effort to increase understanding.

## ORGANIZATION

The book contains calculator-based activities, exercises and projects which complement and extend the basic textbook material. After a brief chapter on Calculator Preliminaries, Chapter 2 is concerned with a number of matrix editing procedures using the calculator and also includes several specialized matrix builder programs. There is no new mathematical content in this chapter, only material intended to make the calculators easy to use.

Chapter 3, Systems of Linear Equations, provides calculator routines which implement Gaussian elimination, back substitution, LU-factorizations and Gauss-Jordan pivoting. The latter is especially useful in a variety of settings, from checking vectors in  $\mathbb{R}^n$  for independence, to basis, dimension and eigenvector calculations. We review the relevant mathematics as we proceed.

Orthogonality is highlighted in Chapter 4, a topic of fundamental importance in many applications. We summarize the principal mathematical theory while developing calculator-based procedures for least squares solutions, fitting polynomials to data, orthonormal bases and QR-factorizations.

Chapter 5, Eigenvalues and Eigenvectors, includes calculator programs for finding the characteristic polynomial and eigenvalues of a matrix, with applications to real symmetric matrices.

Chapter 6 is an introduction to iterative methods and features the Jacobi and Gauss-Seidel iterative techniques for solving large, sparse linear systems and the power method for approximating dominant eigenvalues.

I have included four appendices. The first two are concerned with procedures used to enter, store, run, edit and purge programs on the calculators and the efficient organization of the programs for a course in linear algebra. Appendix 3 is devoted to vector and matrix norms (needed for Chapter 6), while Appendix 4 contains some useful polynomial routines developed at Hewlett-Packard.

## THE ROLE OF CALCULATORS

Within the last two years, graphics programmable calculators with symbol manipulation capabilities, the Hewlett-Packard 48S and 28S, have exploded into undergraduate mathematics. For the first time, students have real graphical, numerical and symbolic computing power in the palms of their hands - often more



than the campus mainframes of 25 years ago. Though most of the early interest in these devices has been directed towards their use in calculus, they are also proving to be an attractive choice of technology to use in matrix-oriented linear algebra - the course recommended as the first course in linear algebra by the 1990 Linear Algebra Curriculum Study Group<sup>1</sup>.

The benefits of using technology in a first linear algebra course are readily apparent:

- to remove the computational burden often associated with hand performance of matrix algorithms, thus allowing beginning students to focus more clearly on the underlying concepts and theory;
- to encourage and enable students to engage in some exploratory/discovery work on their own;
- to consider more interesting and realistic applications;
- to begin to think about some of the computational aspects of linear algebra; and
- to demonstrate some of the advantages and power of technology in mathematics.

To the extent that the HP calculators can help achieve these benefits, in a *fairly substantial way*, I believe them to be an entirely appropriate form of technology.

When every student is equipped with his or her own calculator, they use it almost daily for homework and classwork, there is immediate feedback, and a strong element of active participation and interaction. My lecturing has given way to a

---

<sup>1</sup>Funded by an NSF grant to "initiate substantial and sustained national interest in improving the undergraduate linear algebra curriculum".

more informal discussion, interspersed with appropriate explanations and examples, which students find more interesting.

The presence of calculators introduces a new dynamics into the learning process. Students generally seem to be more involved in their thinking about the material, and when they are able to effectively use these devices—in hand—to help achieve a desired result, they often exhibit a strong sense of "personal ownership" of that result. Indeed, it may well be the highly personalized nature of the HP's which, in addition to their portability, makes them so attractive. Students see them as especially applicable to their needs for they work equally well in hallways, in the library, at park benches and lab benches, and are a constant companion in their backpacks. They need not confine their explorations to central facilities nor spend a lot of money on a computer. There is a genuine aura of excitement surrounding their use, which can only be interpreted in a positive sense. At this level, *I am primarily interested in increasing students' interest, involvement, comprehension and retention of the course material.*

## THEMES AND OPPORTUNITIES

There are four major themes which tend to characterize introductory courses in linear algebra, all well suited to serious calculator enhancement: Gaussian elimination and LU-factorizations, vector space theory associated with matrices, geometrical notions and orthogonality, and eigenvalues and eigenvectors. Because of this, the calculator routines in this book are structured around these themes and are organized into menus, each menu associated with a theme. In my course, calculator use is regular (almost daily), but I am not interested in students acquiring program-development skills. We use the basic built-in keystroke commands whenever possible (e.g., matrix addition and multiplication, scalar multiplication, and the routine computation of dot products, inverses, transposes and norms). Iterative methods



extend the traditional material and point the way toward the powerful numerical approaches used to solve large linear systems and to obtain eigenvalues and eigenvectors.

In preparing the routines, I have been careful not to produce programs which present final results at the expense of students becoming involved with the underlying mathematical processes. Generally, the calculator programs are interactive, requiring input and control at key steps. Thus, for example, the Gaussian elimination routine requires that the user control the process by deciding when and where to pivot, and which row interchanges are needed. And the practical pivoting strategy known as partial pivoting, designed to avoid extremely small pivots which tend to introduce error, is easy to implement in this setting. Likewise, in studying Gram-Schmidt orthonormalization I recommend that students enter and evaluate a simple program at each step; in so doing they demonstrate to me (and reinforce for themselves) their understanding of the basic algorithm.

I am not concerned with introducing a substantial amount of new material into the course. Nevertheless, the calculators have enabled me to achieve good results with two modern topics which were often omitted in earlier versions of the course: the interpretation of Gaussian elimination as an LU-factorization and its application to linear systems with multiple right-hand sides, and the interpretation of the Gram-Schmidt process as a QR-factorization and its application to least squares problems. These topics are important today because they lie close to the heart of many modern computer codes used to handle large linear systems.

By concentrating primarily on the vector spaces naturally associated with a matrix, i.e., the row-space, column-space, null-space and their subspaces, I have been more successful than before in helping students understand the concepts of spanning, independence, bases and dimension. The calculators are a real help here since they

facilitate students seeing these vector space notions in the context of linear systems. Students make frequent use of the Gauss-Jordan pivot routine (which they control at each stage) to reduce the associated matrices to their reduced row-echelon forms. Many of the basic questions can be answered by examining these forms.

Eigenvalues and eigenvectors have always been difficult for students, primarily because they were often unable to obtain the characteristic polynomial of even a small, integer-valued matrix and, even when they could, had trouble finding the eigenvalues as the roots of this polynomial. Stuck in unproductive hand computations, only the most talented were able to move much beyond the basics. But the HP's have changed all that. A simple calculator routine finds the coefficients of the characteristic polynomial of a matrix  $A$  in terms of traces of powers of  $A$ . For modest-sized matrices (say,  $6 \times 6$  or smaller) having low-order, integer-valued entries, the coefficients are accurately determined. A root-finder routine can then be used to find the roots. Such an approach is, of course, unsuitable for finding eigenvalues of the matrices encountered in most legitimate applications in science and engineering, and I am careful to make that point. But it enables students to computationally experience some of the theory and to move on to a consideration of real symmetric matrices.

Calculators also provide an opportunity to dispel some of the computational misconceptions which students tend to carry away from introductory courses. Misconceptions like testing for a matrix's invertibility by computing its determinant, solving non-singular linear systems by finding matrix inverses and, indeed, the entire role of determinants and inverses in the practical application of matrices. And although I have not restructured my course into one emphasizing numerical linear algebra, I am at least able to give students a first-hand glimpse at some of the troublesome numerical issues which are present.



More than anything else, regular and systematic use of the calculators throughout the conduct of the course changes not only what and how we teach, but also what and how we test. I allow free use of the devices on all tests, and part of the learning process is to determine just when, and when not, to use them. There is plenty of room for both theoretical questioning (...*"explain to me your complete understanding of the concept of basis"*...) as well as more computational questioning (...*"obtain a least squares solution to the following overdetermined linear system by finding and applying a QR-factorization"*...). I have been unable to obtain this level of testing in a more computationally restricted environment.

Do students learn more linear algebra with the calculators? Do they better understand what they learn? These are tough questions, questions which I am unable to answer with any strong sense of accuracy. But my students have certainly seen the material in a different light, have clearly shown me that they can grasp some of the concepts better than before, and to the extent that they are all more interested and involved in their learning I see this as a positive enhancement. They are genuinely complimentary in their assessment of the role of calculators in the course.

### SUPPLEMENTARY MATERIAL

The programs in this book have been structured into a calculator directory (the MTRX directory) and loaded onto a computer disk which is available from the publisher to instructors who wish to adopt the book for classroom use. The MTRX directory may be transferred to students using the HP-48S either by calculator-to-calculator infrared transmission or by computer-to-calculator serial transmission as directed in the Owner's Manual. No such option is available for HP-28S users, and the programs must be entered and organized by the individual users themselves, as directed by Appendices 1 and 2.

## ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to The Fund for the Improvement of Postsecondary Education (FIPSE) which has provided substantial funding for this project. I, my students and others who may ultimately benefit from the project am thankful for FIPSE's dedication to innovation and improvement of American postsecondary education.

Also, I wish to acknowledge the editors and production staff at Harcourt Brace Jovanovich for their commitment to excellence in education through excellence in publishing.

Don LaTorre

## Contents

Preface.....	iii
Chapter 1: Calculator Preliminaries.....	1
Number Objects.....	3
Data Entry.....	4
Programming.....	4
Chapter 2: Matrices.....	6
Entering Arrays.....	6
Editing Arrays.....	12
Matrix Arithmetic.....	25
Determinants and Inverses.....	33
Matrix Builder Routines.....	40
Chapter 3: Systems of Linear Equations.....	48
Gaussian Elimination.....	48
LU-factorizations.....	55
Gauss-Jordan Reduction.....	66
Other Variants.....	71
Applications to Vector Spaces.....	72
Linear Combinations and Spanning Sets.....	73
Dependence and Independence.....	74
Bases and Dimension.....	76
Change of Basis.....	78

Chapter 4: Orthogonality.....	82
Basic Concepts.....	83
Orthogonality.....	84
Projections and Least Squares.....	88
Fitting Curves to Data .....	92
Orthonormal Bases.....	98
The Gram-Schmidt Process .....	99
Orthogonal Matrices and QR-factorizations .....	102
Chapter 5: Eigenvalues and Eigenvectors .....	110
The Characteristic Polynomial.....	111
Eigenvalue Calculations.....	115
Similarity.....	123
Real Symmetric Matrices .....	128
Chapter 6: Iterative Methods.....	130
The Jacobi and Gauss-Seidel Methods.....	131
The Power Method.....	140
Appendix 1: Program Housekeeping .....	148
Appendix 2: Program Organization: Directories .....	153
Appendix 3: Vector and Matrix Norms.....	157
Appendix 4: Polynomial Routines.....	163
Program Index I.....	170
Program Index II.....	173
Subject Index.....	175

# CHAPTER 1

## CALCULATOR PRELIMINARIES

Before using this book, the reader should have a basic familiarity with the HP-48S or HP-28S calculator and its operation, to the extent of being able to do elementary keyboard calculations, perform routine real and complex number arithmetic, and understand the basics of the stack. To acquire this basic familiarity on the 48S, study pages 20-21 and 24-27 of Chapter 1 (Getting Started), Chapter 2 (The Keyboard and Display) and Chapter 3 (The Stack and Command Line) of the Owner's Manual. On the 28S, study Chapter 1 (Getting Started), Chapter 2 (Doing Arithmetic) and Appendix C (Notes for Algebraic Calculator Users) of the Owner's Manual.

No further background with the calculator is required in order to begin to explore its capabilities relative to elementary linear algebra, for we shall develop our skills and understanding as we proceed. Though you should have the calculator manuals available to use if needed, our exposition is intended to be self-contained. Readers who wish to acquire an increased level of insight and understanding into the theory and operation of the calculator are strongly advised to obtain one of the books "HP-48 Insights"<sup>1</sup> or "HP-28 Insights"<sup>2</sup> , by William C. Wickes.

To help you recognize various calculator keystrokes and commands, we shall adopt certain notational conventions.























- With the exception of the six white keys on the top row, keys will be represented by helvetica characters in a box: `ENTER`, `EVAL`, `+`, `STO`, etc.





---







<sup>1</sup> 1991 by Larken Publications, 4517 NW Queens Avenue, Corvallis, Oregon 97330

<sup>2</sup> 1988 by Larken Publications, 4517 NW Queens Avenue, Corvallis, Oregon 97330



- On the 28S the symbol  will represent the cursor menu key.
- Shifted keys on the 48S may occasionally have the key name in a box preceded by the appropriate shifts(s):  ,  ,  ,   ,   , etc. But ordinarily, we will not show the shifts. Similarly, shifted keys on the 28S may occasionally have the key name in a box preceded by a black square  to represent the red shift key:  ,  ,  ,  , etc. But ordinarily, we will not show the shift key.
- Menu keys for commands on various menus will show the key name in outline form in a box:

	48S	28S
	PRG STK menu	STACK menu
	PRG OBJ menu	LIST menu
	MTH PROB menu	REAL menu
	PRG OBJ menu	ARRAY menu

- Menus generally have more labels than can be shown above the six white menu keys, and the  key will display the next row (page) of labels. Return to the previous page with . For simplicity, we will not show these two keys in this book.
- The four white arrow keys will be indicated by , ,  and  .

- We will not put boxes around the comma, letter or number keys. For instance, we shall write 4, 63 instead of  $\boxed{4}\boxed{,}\boxed{6}\boxed{3}$ , and ROW instead of  $\boxed{R}\boxed{O}\boxed{W}$ .
- Calculator operations and commands that appear in programs or in the text material will be in helvetica characters, e.g., DUP SWAP INV.
- Finally, you should note that in certain entry modes some keys display different characters from what is on them. For example, in algebraic entry mode (activated by  $\boxed{,}$ ) or program entry mode (activated by  $\boxed{\text{« »}}$  on the 48S,  $\boxed{\text{«}}$  on the 28S):

Key	Display
$\boxed{+}$	/
$\boxed{1/x}$	INV
$\boxed{\times}$	*
$\boxed{x^2}$	SQ
$\boxed{\sqrt{x}}$	$\sqrt{\phantom{x}}$
$\boxed{y^x}$	$\wedge$

**NUMBER OBJECTS.** Real and complex numbers are two of the many different types of "objects" that the calculator can recognize, manipulate and store.

A real number object is the calculator's representation of a 12-digit floating point number:

$$\text{mantissa} \times 10^{\text{exponent}}$$

where the mantissa is a 12-digit number between 1 and 9.99999999999, and  $-499 \leq \text{exponent} \leq 499$ . Although the current display mode (STD, FIX, SCI or ENG)

determines how real number objects are *displayed*, all internal calculations begin by first expanding mantissas to 15 digits and exponents to 5 digits, performing the calculations in that environment, then rounding back to 12-digit mantissas and 3-digit exponents. However, this does not mean that all calculations are accurate to 12 digits: round-off errors from intermediate results may compound as the calculation proceeds.

A complex number object is an ordered pair  $(x,y)$  of real numbers, and most arithmetic, logarithmic, exponential and trigonometric operations treat real and complex number objects uniformly. You are free to mix these two object types, and the calculator will return a complex number if any input argument is complex.

**DATA ENTRY.** When keying a sequence of real numbers into the command line, say 1.1, 2.2 and 3.3, you must separate the numbers with spaces or commas for proper recognition, as in 1.1 2.2 3.3 or 1.1, 2.2, 3.3. We recommend that you use spaces on the 48S and commas on the 28S for ease of use. For consistency throughout this manual we will show commas, but you should always interpret them as spaces on the 48S. You need not insert commas or spaces between a real number and a complex number (an ordered pair), or between two complex numbers, because the calculator recognizes the parentheses as object delimiters.

Unless we specify otherwise, the examples and exercises in this book assume the calculator is set to STD display mode.

**PROGRAMMING.** We shall require no skill or experience in writing programs but, unless you obtain our routines by calculator-to-calculator infrared transmission or by computer-to-calculator serial transmission (both options available for the 48S), you will need to copy and enter simple programs into your calculator. In doing so, you must be careful and copy the programs exactly as we show them. Special attention should be given to correct spacing because the calculator recognizes commands that

are separated by spaces. Instead of spelling out commands from the alphabet keyboard, we recommend that you use the keystroke commands which appear either as shifted keys (e.g., SWAP, DROP, PURGE) or as labels on the various menus; keystroke commands will automatically insert spaces around each command. To use the menu commands requires some familiarity with their location, but this can be acquired in the course of entering programs. For the 48S the Operation Index, which begins on page 707 of the Owner's Manual, gives the name, a description, and the location of all HP-48S commands. The Operation Index for the 28S begins on page 323 of the Reference Manual.

Using keystroke commands will also increase your speed in keying in programs and help avoid errors due to the insertion of extra spaces. For instance, using the  $\boxed{R}$ ,  $\boxed{\rightarrow}$ , and  $\boxed{C}$  keys to enter the command  $R \rightarrow C$  will produce  $R \rightarrow C$ , and the spaces surrounding the arrow will result in an error. The desired  $R \rightarrow C$  command (real-to-complex conversion) can be found on the second page of the PRG OBJ (program object) menu of the 48S, and on the first page of the COMPLEX menu of the 28S.

Appendix 1 is a brief review of the procedures for entering, naming, storing, editing, visiting, recalling and purging programs. Unless you are experienced in these matters, you should read this appendix now before you begin to encounter programs in Chapter 2.

Appendix 2 is a discussion of how the programs in the book should be organized, and we recommend (especially to 28S users) that you read it immediately after finishing Appendix 1.

## CHAPTER 2

### MATRICES

On the HP-48S and HP-28S, rectangular arrangements of real or complex numbers are called *arrays*. Arrays can be one-dimensional (vectors) or two-dimensional (matrices) and are considered to be single objects. Consequently, they can be manipulated with many of the same basic commands used in ordinary arithmetic. We shall begin by examining some of the ways of entering, editing, and manipulating arrays.

#### 2.1 ENTERING ARRAYS

A one-dimensional array (vector) is represented on the calculator by enclosing a sequence of real or complex numbers in square brackets, as in [ 1 2 3 ] or [(1,2) (3,4) (5,6)]. A two-dimensional array (matrix) is distinguished by an initial square bracket [ , followed by each row vector, and ends with a closing square bracket ] . For example, in standard display mode the 2×3 real matrix  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$  will

appear as  $\begin{bmatrix} [ [ 1 & 2 & 3 ] \\ [ 4 & 5 & 6 ] ] \end{bmatrix}$  and the 3×2 complex matrix  $\begin{bmatrix} 1+i & 1+2i \\ 2+i & 2+2i \\ 3+i & 3+2i \end{bmatrix}$  will appear as

[ [ (1,1), (1,2) ]  
 [ (2,1), (2,2) ] .  
 [ (3,1), (3,2) ] ]



ON THE 48S.

USING THE COMMAND LINE. The vector [ 1 2 3 ] is entered with keystrokes  $\boxed{\leftarrow}$   $\boxed{[ ]}$  1, 2, 3  $\boxed{\text{ENTER}}$ . Be sure to insert spaces between the 1, 2, 3 with the  $\boxed{\text{SPC}}$  key. There are two ways of entering a matrix:

- *row-by-row*: start the matrix with [ [ by pressing the  $\boxed{[ ]}$  key twice, enter the first row and press  $\boxed{\blacktriangleright}$ , then continue entering the remaining entries in row order and press  $\boxed{\text{ENTER}}$ .

EXAMPLE:

Keystrokes

$\boxed{[ ]}$   $\boxed{[ ]}$  1, 2, 3  $\boxed{\blacktriangleright}$  4, 5, 6, 7, 8, 9  $\boxed{\text{ENTER}}$

[[ 1 2 3 ]  
will produce the matrix [ 4 5 6 ] .  
[ 7 8 9 ]]












The  $\boxed{\blacktriangleright}$  key simply defines the number of columns. Now press  $\boxed{\text{DROP}}$  to drop this matrix from the stack. (When no command line is present you need not press the  $\boxed{\leftarrow}$  to DROP.)



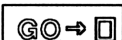
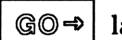




- *as a dimensioned array*: enter the numbers into the command line from left-to-right in row order separated by spaces, then the dimensions as a list, {no. rows, no. columns}, and press  $\boxed{\text{ENTER}}$  to place all this on the stack. Then press  $\boxed{\rightarrow \text{ARR}}$  (on the PRG OBJ menu).







EXAMPLE:

Keystrokes 1, 2, 3, 4, 5, 6  $\boxed{\{ \}}$  2, 3  $\boxed{\text{ENTER}}$   $\boxed{\rightarrow \text{ARR}}$

return the matrix  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ .

The numbers may be any mixture of real or complex numbers (ordered pairs), but if any one entry is complex then the entire array will be complex. To enter a single complex number, say (1,2), press   1, 2 . To enter the two complex numbers (3,4) and (5,6) press   3, 4    5, 6 . Notice the action of the  key and that the  completes the entry.

**USING THE MATRIXWRITER.** Enter the MatrixWriter application by pressing  . This activates a spreadsheet-type display, with a dark cursor resting in the 1-1 position. Check to see that the  command is active by noting a small white box within this menu label (if the box is not present, simply press the white key beneath the  label to activate it.) Key in the numbers of the first row of the matrix in row order separated by spaces and then press . When you are ready to go to the second row press . This will define the number of columns and position the cursor at the 2-1 entry. Now key in the remaining entries of the matrix in row order (separated by spaces) and press . A final  will put the matrix onto the stack.

**EXAMPLE.** The keystrokes   1, 2, 3   4, 5, 6, 7, 8, 9   will produce this matrix on the stack:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Clearly, for entering simple matrices (say, with integer entries) the command line is faster and easier to use than the MatrixWriter application. But the

MatrixWriter has the advantage that for more complicated matrices, an entry can be calculated (using RPN syntax) on the command line within the MatrixWriter environment before it is entered into its position. As an example, construct the matrix

$$\begin{bmatrix} \sqrt{17} & \ln 3 \\ e & \pi/2 \end{bmatrix}.$$

Although the term MatrixWriter suggests that it can be used only for matrices, it is actually an extremely versatile environment for entering, reviewing and editing both *vectors* and *matrices*. To enter a vector using the MatrixWriter, say vector  $[1 \ 2 \ 3 \ 4]$ , start with an empty stack and enter the MatrixWriter environment with  $\boxed{\rightarrow}$   $\boxed{\text{MATRIX}}$ . Note that the menu key  $\boxed{\text{VEC}\square}$  appears. If you press 1, 2, 3, 4  $\boxed{\text{ENTER}}$   $\boxed{\text{ENTER}}$ , *vector*  $[1 \ 2 \ 3 \ 4]$  will show on the stack. The presence of the white box in  $\boxed{\text{VEC}\square}$  indicates that vector entry is active. If you toggle off this key to see  $\boxed{\text{VEC}}$  without the box, the keystrokes 1, 2, 3, 4  $\boxed{\text{ENTER}}$   $\boxed{\text{ENTER}}$  will return the *matrix*  $\begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$ .

Whenever you enter the MatrixWriter with  $\boxed{\rightarrow}$   $\boxed{\text{MATRIX}}$ , the vector entry mode  $\boxed{\text{VEC}\square}$  is active by default. But if you enter the MatrixWriter with  $\boxed{\nabla}$  to review an array on level 1, the status of  $\boxed{\text{VEC}}$  reflects the nature of the array:  $\boxed{\text{VEC}\square}$  for a vector and  $\boxed{\text{VEC}}$  for a matrix. Finally, note that you can quickly convert the vector  $[1 \ 2 \ 3 \ 4]$  to the matrix  $\begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$  and vice-versa by starting with either one on level 1, pressing  $\boxed{\nabla}$  to enter the MatrixWriter, then changing the status of  $\boxed{\text{VEC}}$  and pressing enter.

A final note about entering arrays using the MatrixWriter application. Array entries may be real or complex numbers, but when you use the MatrixWriter to initially enter a matrix into the calculator, the array object type (real or complex) is

determined by the 1-1 entry. Thus, if the 1-1 entry is real, you cannot enter a subsequent entry as a complex number. But, if the 1-1 entry is a complex number (an ordered pair), any subsequent entry of a real number  $x$  will be accepted and written as the complex number  $(x, 0)$ .

ON THE 28S. The vector  $[1\ 2\ 3]$  is entered with keystrokes  $\boxed{[}\ 1,\ 2,\ 3\ \boxed{ENTER}$ . Note that the enter command actually produces the closing bracket for you. There are two ways of entering a matrix:

- *row-by-row*: start the matrix with  $[$ , enter each row as a separate vector, then press  $\boxed{ENTER}$

EXAMPLE:                      Keystrokes

$\boxed{[}\ \boxed{[}\ 1,\ 2,\ 3$

$\boxed{[}\ 4,\ 5,\ 6\ \boxed{ENTER}$

produce the matrix  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ .

Now press  $\boxed{DROP}$  to drop this matrix from the stack.

- *as a dimensioned array*: enter the numbers from left-to-right in row order separated by commas, then the dimensions as a list, {no. rows, no. columns}, then press  $\boxed{\Rightarrow ARRAY}$  (found on the  $\boxed{■}\ \boxed{ARRAY}$  menu).

EXAMPLE:                      Keystrokes  $1,\ 2,\ 3,\ 4,\ 5,\ 6\ \boxed{\{}\ 2,\ 3\ \boxed{\Rightarrow ARRAY}$

return the matrix  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ .

Note, again, that pressing  $\boxed{\Rightarrow ARRAY}$  has the effect of closing the curly braces on  $\{2,\ 3\}$ .

The numbers may be any mixture of real or complex numbers (ordered pairs), but if any one entry is complex then the entire array will be complex (try it!). Also, you may use a space instead of a comma to separate number entries, but on the 28S the **SPACE** key is located on the left keyboard, making it awkward to use.

**NOTE:** On the 48S and the 28S, as a matter of convenience, any  $n$ -vector  $x = [x_1 \ x_2 \ \dots \ x_n]$  may be premultiplied by any  $m \times n$  matrix  $A$  to obtain  $Ax$ . Thus, in this context,  $x$  is treated as if it were an  $n \times 1$  matrix. But you should note that this treatment of  $x$  is peculiar to this context: *in all other applications,  $x$  is a vector ... not a matrix.* You may not, e.g., perform a multiplication like  $xA$  for a matrix  $A$ , nor can you transpose or take the determinant of a 1-vector  $[x]$ .

## EXERCISES 2.1

1. Set the number display mode to **STD** and practice entering the following integer-valued matrices row-by-row. To see the hidden entries on the 48S, simply press **▽** to view the matrix in the MatrixWriter environment, where the four white arrow keys enable you to move to any position. The entry in the cursor position is identified on the command line. On the 28S, when the command line is inactive ( press **< >** to clear or display it ) you can see 4 rows. The keys **VIEW↑** and **VIEW↓** allow you to scroll vertically to view more rows. Alternatively, on either calculator, with the matrix in level 1 you may use **EDIT** to put it on the command line, where the white arrow keys can be used to see any part of the matrix; press **ON** when you're finished viewing.

$$\begin{array}{lll}
 \text{(a)} \begin{bmatrix} 3 & -1 & 2 \\ -4 & 0 & 1 \\ 2 & 3 & 5 \end{bmatrix} & \text{(b)} \begin{bmatrix} 4 & 2 & -1 \\ 5 & 0 & 3 \\ -6 & 1 & 8 \\ 0 & -2 & 9 \end{bmatrix} & \text{(c)} \begin{bmatrix} 1 & -1 \\ -2 & 2 \\ 3 & -3 \\ -4 & 4 \\ 5 & -5 \\ -6 & 6 \end{bmatrix} \\
 \text{(d)} \begin{bmatrix} 0 & 1 & -1 & 0 & 2 & -2 & 0 & 3 & -3 & 0 & 4 & -4 \\ 0 & 5 & -5 & 0 & 6 & -6 & 0 & 7 & -7 & 0 & 8 & -8 \end{bmatrix}
 \end{array}$$

- Enter each of the matrices in Exercise 1 as a dimensioned array.
- Set the number display mode to 2 FIX, and go to the MTH PROB menu on the 48S and the REAL menu on the 28S. Each press of the RAND key will produce a random number, rounded to 2 decimal digits. Practice entering a few "random matrices" *as dimensioned arrays*. What happens if you try *row-by-row entry* of random matrices?

## 2.2 EDITING ARRAYS

In working with arrays you will sometimes have to perform various editing procedures such as taking a matrix apart, determining dimensions, extracting entries, changing entries, extracting rows or columns, deleting rows or columns, separating into rows or inserting additional rows or columns. These are the kinds of editing procedures we are accustomed to performing when working with pencil and paper.

**TAKING AN ARRAY APART.** Just as the menu key ⇒ARRY creates arrays, the keys OBJ⇒ on the 48S and ARRY⇒ on the 28S take an array apart. For

example, with the matrix  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$  on level 1, pressing OBJ⇒ or ARRY⇒ puts the entries on the stack in row order and the dimensions in level 1. For example:

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$   $\boxed{\text{OBJ} \rightarrow}$  returns 1, 2, 3, 4, 5, 6, { 2 3 }. (This sequence is interpreted as

follows: with  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$  on level 1, pressing  $\boxed{\text{OBJ} \rightarrow}$  returns 1, 2, 3, 4, 5, 6 and { 2 3 } to the stack with { 2 3 } on level 1).

**DETERMINING DIMENSIONS.** The menu key  $\boxed{\text{SIZE}}$ , located on the third page of the PRG OBJ menu of the 48S and on the second page of the ARRAY (or LIST) menu of the 28S, returns the dimensions of an array on level 1:

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$   $\boxed{\text{SIZE}}$  returns { 2 3 }, and  $[ 1 \ 2 \ 3 ]$   $\boxed{\text{SIZE}}$  returns { 3 }.

**EXTRACTING ENTRIES.** To get a particular entry from an array use the menu key  $\boxed{\text{GET}}$  ( $\boxed{\text{GET}}$  is on the PRG OBJ menu of the 48S and on the ARRAY menu of the 28S):

$\begin{bmatrix} 4 & 5 & 6 \\ 1 & 2 & 3 \end{bmatrix}$  { 1 2 }  $\boxed{\text{GET}}$  returns 5. (Meaning: with the indicated array on level 2, and the list { 1 2 } on level 1,  $\boxed{\text{GET}}$  returns 5 to level 1.)

On the 48S there is an even better way to extract an array entry. Starting with the array on level 1, press  $\boxed{\nabla}$  to view it in the MatrixWriter environment and move the cursor to the position whose entry you want to extract. Then press  $\boxed{\text{NXT}}$  to turn to the next page of the MATRIX menu and press  $\boxed{\rightarrow \text{STK}}$  to copy the entry to the stack. Press  $\boxed{\text{ENTER}}$  to see the array on level 1 and the extracted entry on level 2.



**CHANGING ENTRIES.** There are several ways to change entries in an array.

- (i) You can copy the array from level 1 to the command line with **EDIT**, where the white arrow keys then let you move to any desired entry and change it. With the 48S you use the **DEL** key to delete characters, then simply key in the new characters. With the 28S the **INS** key toggles between Replace Mode (a box cursor), in which new characters *replace* existing ones, and Insert Mode (an arrow cursor), in which new characters are *inserted* between existing ones. Return the edited matrix to level 1 with **ENTER**.
- (ii) Another way to change an entry in an array is to use the menu keys **PUT** (on the last page of the PRG OBJ menu on the 48S and the first page of the ARRAY menu on the 28S).

**EXAMPLE:** With  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$  on level 1, the keystrokes **{}** 1, 2 **ENTER**

7 **PUT** return  $\begin{bmatrix} 1 & 7 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ . (On the 28S use **{** instead of **{}**).

**CAUTION:** *If the number to be put into the array is complex, the array itself must be complex.*

- (iii) On the 48S you may copy the array into the MatrixWriter with **▽**, position the cursor over the entry to be changed, key the new entry into the command line and press **ENTER** to insert it at the cursor location. Return to the stack with another **ENTER**. This method is especially useful because you can calculate the new entry on the command line in RPN before entering it.

- (iv) You may use the following simple program NU.EL; but *note that it is written to handle only matrices and not vectors.*

**NU.EL**      (New matrix element)

*Inputs:*    level 4: a matrix  
                  level 3: an integer I  
                  level 2: an integer J  
                  level 1: a number NU

*Effect:*    returns to level 1 the input matrix having  
                  the number NU as the ( I, J )-entry.

« → I J NU « { I J } NU PUT » »

Key the program into the command line and press **ENTER** to put it onto level 1 of the stack. Then press **↓**, key in NU.EL, and press **STO** to store the program under the name NU.EL in your user memory. Pressing **VAR** on the 48S or **USER** on the 28S will display a new menu key **NU.EL**. (Now is a good time to read Appendices 1 and 2, if you have not done so already.)

EXAMPLE. Starting with matrix  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$  on level 1 and performing the keystrokes 1, 2, 7 **NU.EL** will return  $\begin{bmatrix} 1 & 7 & 3 \\ 4 & 5 & 6 \end{bmatrix}$  to level 1.

You may use program NU.EL to change the sign of an entry. But on the 48S it is much easier to put the matrix into the MatrixWriter environment, position the cursor

over the desired entry, and then press **EDIT** **+/-** **ENTER** **ENTER**. Since this option is not available on the 28S, you may find it convenient to use the following program CH.SN.

**CH.SN** (change sign)

*Inputs:* level 3: a matrix A

level 2: an integer I

level 1: an integer K

*Effect:* changes the sign of the (I, K)-entry of A

« → A I K « A 'A(I, K)' EVAL NEG { I K } SWAP PUT » »

Press **▢** CH.SN **STO** to name and store the program in user memory. Press **VAR** or **USER** to see the new menu key **CH.SN**.

EXAMPLE. With matrix  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$  on level 1, keystrokes 2, 3 **CH.SN** return  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & -6 \end{bmatrix}$ .

**EXTRACTING ROWS OR COLUMNS.** There are times when you may want to extract a particular row or column to use in some way. The following program will extract a specified row from a matrix, put the extracted row onto level 1 and the original matrix on level 2.

**C.ROW** (Extract a matrix row)

*Inputs:* level 2: a matrix

level 1: an integer L

*Effect:* puts row L of the matrix onto level 1  
and the matrix onto level 2

```
« → A L « A SIZE 2 GET → N « 1 N FOR I 'A(L, I)'
  EVAL NEXT N →ARRY A SWAP » » »
```

As with the earlier programs, key the program into the command line and press **ENTER** to place it onto level 1 of the stack. Then press **,**, key in C.ROW, and press **STO** to store the program in user memory under its name. Pressing **VAR** or **USER** will display a new menu key **C.ROW**.

EXAMPLE. With matrix  $\begin{bmatrix} 6 & 3 & 5 \\ 4 & 8 & 7 \\ 9 & -1 & 2 \\ 3 & 4 & 1 \\ -2 & 7 & 9 \end{bmatrix}$  on level 1, the keystrokes 5 **C.ROW** will

return this matrix to level 2 and the vector  $[-2 \ 7 \ 9]$  to level 1.

The column version of program C.ROW is C.COL:

**C.COL** (Extract a matrix column)

```
« SWAP TRN CONJ SWAP C.ROW SWAP TRN CONJ
  SWAP »
```

NOTE: The command CONJ appears twice in this program, each time immediately after TRN. The command TRN returns the *conjugate transpose* of a matrix, which is the transpose if the matrix is real. But if the matrix is complex, TRN must be followed by CONJ to see the ordinary transpose of the matrix. See Section 2.3.

$$\begin{bmatrix} (3,0) & (1,1) & (3,4) \end{bmatrix}$$

Try this program on the matrix  $\begin{bmatrix} (1,-1) & (4,0) & (2,-3) \\ (3,-4) & (2,3) & (5,0) \end{bmatrix}$ .

**DELETING A ROW OR COLUMN.** On the 48S, the MatrixWriter provides a way to delete a specified row or column from a matrix on level 1. Simply display the matrix in the MatrixWriter environment and move the cursor to any position in the row (or column) you wish to delete, turn to the next page of the MATRIX menu with NXT and then press -ROW (or -COL) to delete the row (or column). As usual, return the new matrix to level 1 with ENTER.

On the 28S, we recommend that you use the program DELROW [Wickes<sup>2</sup>, p. 275] given below. By transposing the matrix, executing DELROW and then transposing the result, you can also delete a column. But, it is more convenient to use a simple program DELCOL which performs these operations for you. Finally, to delete both a specified row and a specified column (i.e., to compute a specified minor of the input matrix), program MINOR [Wickes, p. 275] may be used.

**DELROW** (Delete a matrix row)

*Inputs:* level 2: a matrix

level 1: an integer  $r$

*Effect:* Deletes row  $r$  of the matrix and returns the modified matrix to level 1.

« →  $r$  « ARRAY → LIST → DROP →  $n\ m$  «  $n\ r - m$  \*  
 → LIST →  $s$  «  $m$  DROPN  $s$  LIST → DROP »  $n\ 1 - m\ 2$   
 → LIST → ARRAY » » »

The column version of DELROW is DELCOL:

**DELCOL** (Delete a matrix column)

« SWAP TRN SWAP DELROW TRN »

EXAMPLE. With  $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$  on level 1, keystrokes 3 DELRO return

$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$ . Then 2 DELCO returns  $\begin{bmatrix} 1 & 3 & 4 \\ 5 & 7 & 8 \end{bmatrix}$ .

**MINOR** (Computes a matrix minor)

*Inputs:* level 3: a matrix

level 2: an integer r

level 1: an integer c

*Effect:* returns the matrix with row r and column c deleted.

« 3 ROLLD DELROW TRN SWAP DELROW TRN »

EXAMPLE. With matrix  $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$  on level 1, keystrokes 1, 3 MINO

return  $\begin{bmatrix} 5 & 6 & 8 \\ 9 & 10 & 12 \end{bmatrix}$ .

**SEPARATING INTO ROWS.** After copying a matrix into the command line with EDIT, you can remove the starting and ending brackets of the matrix to separate it into its row vectors.

You should try it on a matrix of your choice. A program which does all this in a single keystroke is `→ROW` [*Vectors and Matrices*, Copyright Hewlett-Packard Company, 1987. Reproduced with permission.].

**→ROW** (Separate into rows)

*Inputs:* level 1: a matrix

*Effect:* separates a matrix into its row vectors

```
« ARRAY→ LIST→ DROP → n m « 1 n FOR i m 1
→LIST →ARRAY n i - m * i + ROLLD NEXT » »
```

EXAMPLE. With  $\begin{bmatrix} 5 & 2 & 3 \\ 1 & 6 & 2 \\ 8 & 9 & 1 \end{bmatrix}$  on level 1, **→ROW** returns  $\begin{bmatrix} 5 & 2 & 3 \\ 1 & 6 & 2 \\ 8 & 9 & 1 \end{bmatrix}$ .

A companion program, also by Hewlett-Packard [p. 41 in *Vectors and Matrices*], is ROW→, which assembles a stack of row vectors into a matrix.

**ROW→** (Assemble row vectors into a matrix)

*Inputs:* levels 2 through  $n + 1$ : vectors  $v_n, v_{n-1}, \dots, v_1$   
respectively

level 1: the integer  $n$

*Effect:* assembles the  $n$  vectors into a matrix having  $v_i$  as row  $i$ .

```
« OVER SIZE LIST→ DROP → n m « 0 n 1 - FOR i i m
* n i - + ROLL ARRAY→ DROP NEXT n m 2 →LIST →ARRAY
» »
```



**EXAMPLE.** With the stack showing

3: [ 1 2 3 ]

2: [ 4 5 6 ]

1: [ 7 8 9 ]

3 ROW→ returns  $\begin{bmatrix} [1 & 2 & 3] \\ [4 & 5 & 6] \\ [7 & 8 & 9] \end{bmatrix}$ .

**INSERTING ADDITIONAL ROWS.** The MatrixWriter application also includes a way to insert additional rows or columns into a matrix. With the matrix displayed in the MatrixWriter, the menu key +ROW (or +COL) will insert a row (or column) of zeros at the cursor position. You may then inset new entries in place of the zeros.

But it is generally more convenient (and a *necessity* on the 28S) to use a simple program to do the insertion. Now that we have program →ROW which separates a matrix into its (say, n) row vectors, it would seem natural that we merely insert an additional vector into the list at the right place, then reassemble the (n + 1) vectors into a new matrix. Program AD.ROW (*additional row*), does just that. Its column analogue, AD.COL, can be used to insert an additional column.

**AD.ROW** (Additional row)

*Inputs:* level 3: a matrix  
 level 2: a vector  $\bar{v}$   
 level 1: an integer K

*Effect:* inserts vector  $\bar{v}$  into the input matrix as an additional row, row K.

« → A V K « A SIZE 1 GET → M « A →ROW V M 1  
 + K - 1 + ROLL M 1 + ROW→ » » »

**AD.COL** (Additional column)

*Inputs:* level 3: a matrix  
 level 2: a vector  $\bar{v}$   
 level 1: an integer K

*Effect:* inserts vector  $\bar{v}$  into the input matrix as an additional column, column K.

« ROT TRN CONJ 3 ROLL AD.ROW TRN CONJ »

EXAMPLE. To enlarge  $\begin{bmatrix} 1 & -2 & 3 & 7 \\ 0 & 5 & -8 & 6 \\ -2 & -3 & 4 & 0 \end{bmatrix}$  by inserting a row of alternating 1's and 0's as

row 3, we start with  $\begin{bmatrix} 1 & -2 & 3 & 7 \\ 0 & 5 & -8 & 6 \\ -2 & -3 & 4 & 0 \end{bmatrix}$  on level 2 and  $[1 \ 0 \ 1 \ 0]$  on level 1 and

press 3 AD.RO. The result 
$$\begin{bmatrix} 1 & -2 & 3 & 7 \\ 0 & 5 & -8 & 6 \\ 1 & 0 & 1 & 0 \\ -2 & -3 & 4 & 0 \end{bmatrix}$$
 is returned. After entering

$\begin{bmatrix} -1 & 6 & 5 & 0 \end{bmatrix}$ , keystrokes 2 AD.CO return 
$$\begin{bmatrix} 1 & -1 & -2 & 3 & 7 \\ 0 & 6 & 5 & -8 & 6 \\ 1 & 5 & 0 & 1 & 0 \\ -2 & 0 & -3 & 4 & 0 \end{bmatrix}.$$

**INTERCHANGING ROWS.** One of the most useful editing procedures is to interchange two rows, say rows K and L. The following program enables you to do this.

**RO.KL** (Interchange rows K and L)

*Inputs:* level 3: a matrix A

level 2: an integer K

level 1: an integer L

*Effect:* interchanges rows K and L of matrix A

« → A K L « A SIZE 2 GET → N « A 1 N FOR I

'A(K,I)' EVAL { L I } SWAP PUT NEXT 1 N FOR J 'A(L,J)'

EVAL { K J } SWAP PUT NEXT » » »

EXAMPLE. With matrix  $\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}$  on level 1, press 1, 3  $\boxed{\text{RO.KL}}$  to return

$\begin{bmatrix} 3 & 3 & 3 \\ 2 & 2 & 2 \\ 1 & 1 & 1 \end{bmatrix}$ .

## 2.3 MATRIX ARITHMETIC

Addition and subtraction of matrices proceeds just as for real numbers. To calculate  $A+B$  simply key in matrix A and press  $\boxed{\text{ENTER}}$ , then key in B and press  $\boxed{+}$ . Pressing  $\boxed{-}$  instead of  $\boxed{+}$  calculates  $A-B$ . Note that the commands  $\boxed{+}$  and  $\boxed{-}$  add or subtract the object on level 1 to or from the one on level 2. In case A and B are stored in user memory, you have two choices:

- with the menu keys  $\boxed{\text{A}}$  and  $\boxed{\text{B}}$  showing, press  $\boxed{\text{A}}$   $\boxed{\text{B}}$   $\boxed{+}$  to add;
- alternatively, you may use algebraic entry mode and press  $\boxed{,}$  A + B  $\boxed{\text{EVAL}}$  to add.

**SCALAR MULTIPLICATION.** To multiply a matrix by a scalar  $c$ , key in the matrix and press  $\boxed{\text{ENTER}}$ , then key in scalar  $c$  and press  $\boxed{*}$ . If either the matrix or the scalar is complex then the result will be a complex matrix. If the matrix is in user memory, you may use the proper menu key or algebraic entry mode as above. Multiplying by -1 can be done with a single keystroke by pressing the  $\boxed{+/-}$  key on the 48S or the  $\boxed{\text{CHS}}$  key on the 28S.

**MATRIX MULTIPLICATION.** To calculate a matrix product  $AB$ , proceed as in forming  $A+B$  but press  $\boxed{*}$  instead of  $\boxed{+}$ . The important point to keep in mind is that in calculating  $AB$ , matrix A must be on level 2 and matrix B on level 1. That is,

the number of columns of the matrix on level 2 must equal the number of rows of the matrix on level 1. Ordinarily, this means that you should enter the left-hand factor first.

**MATRIX POWERS.** Unlike the case for real or complex numbers, you cannot use the  $\boxed{\wedge}$  key to calculate powers of a square matrix A. You can, however, obtain  $A^2$  by using the  $\boxed{x^2}$  key or executing the command SQ. For more general powers of A, say  $A^K$  where  $K = 1, 2, 3, \dots$ , you can use the following program.

**A.KTH**      ( $K^{\text{th}}$  power of a matrix)  
*Inputs:*    level 2: a square matrix A  
                  level 1: an integer K  
*Effect:*    returns  $A^K$ , the  $K^{\text{th}}$  power of A

« → A K « A SIZE 1 GET IDN 1 K FOR I A \* NEXT » »

**EXAMPLE.** To calculate  $B^5 + B^3 + B$  for  $B = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 \\ -1 & 0 & 1 & 0 \end{bmatrix}$ , begin by entering the given matrix and storing it in memory as matrix B. Now press  $\boxed{B}$  5  $\boxed{A.KTH}$  to

put  $\begin{bmatrix} 0 & 16 & 0 & -16 \\ 16 & 0 & -16 & 0 \\ 0 & -16 & 0 & 16 \\ -16 & 0 & 16 & 0 \end{bmatrix}$  on level 1. Next, press  $\boxed{B}$  3  $\boxed{A.KTH}$  to put

$$\begin{bmatrix} 0 & 4 & 0 & -4 \\ 4 & 0 & -4 & 0 \\ 0 & -4 & 0 & 4 \\ -4 & 0 & 4 & 0 \end{bmatrix}$$
 on level 1. Finally, press B ENTER + + to show  $B^5 + B^3 + B$  as

$$\begin{bmatrix} 0 & 21 & 0 & -21 \\ 21 & 0 & -21 & 0 \\ 0 & -21 & 0 & 21 \\ -21 & 0 & 21 & 0 \end{bmatrix}$$

More generally, given a square matrix  $A$  and an arbitrary polynomial  $p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ , we may want to find  $p(A) = A^n + a_{n-1} A^{n-1} + \dots + a_1 A + a_0 I$ . The following program, P.OFA, does just that.

**P.OFA** (Polynomial evaluation at A)

*Inputs:* level 2: a list {  $a_n$   $a_{n-1}$  ...  $a_1$   $a_0$  } of coefficients  
 level 1: a square matrix A

*Effect:* returns  $p(A) = a_n A^n + a_{n-1} A^{n-1} + \dots + a_1 A + a_0 I$

« → L A « A SIZE 1 GET → K « L 1 GET 2 L SIZE  
 FOR N A \* L N GET K IDN \* + NEXT » » »

**EXAMPLE.** Find  $p(A)$  for  $p(x) = 1.3x^5 - .4x^4 + 2.1x^2 + 5x + 6.2$  and

$$A = \begin{bmatrix} .1 & .2 & .3 & .4 \\ .5 & .6 & .7 & .8 \\ .9 & .8 & .7 & .6 \\ .5 & .4 & .3 & .2 \end{bmatrix}$$
 . Write the coefficients as a list {1.3 -4 0 2.1 5 6.2}.

Next enter matrix A. Pressing P.OFA and using 3 FIX display mode we have

$$p(A) = \begin{bmatrix} 12.455 & 6.677 & 7.099 & 7.521 \\ 16.975 & 23.597 & 17.819 & 18.241 \\ 20.545 & 20.123 & 25.901 & 19.279 \\ 9.825 & 9.403 & 8.981 & 14.759 \end{bmatrix}.$$

**CAUTION:** You must use caution when calculating powers of a matrix. Because your calculator only shows 12 digit mantissas, powers of even *small* matrices may lead to

computational inaccuracies. For example, if  $A = \begin{bmatrix} 9 & 9 & 9 & 9 \\ 9 & 9 & 9 & 9 \\ 9 & 9 & 9 & 9 \\ 9 & 9 & 9 & 9 \end{bmatrix}$ , then  $A^8$  can be

found correctly on the calculator to be the constant  $4 \times 4$  matrix whose entries are  $4^7 \cdot 9^8 = 705,277,476,864$ . But  $A^9$  has entries  $4^8 \cdot 9^9$ , a number which the calculator can only represent as 2.5389989167E13, but which is 104 short of the actual 2.5389989167104E13.

**ADDING A CONSTANT.** To add a constant  $c$  to each entry of a matrix  $A$ , use the **CON** key (on the MTH MATR menu of the 48S and on the ARRY menu of the 28S) to create a constant matrix whose entries are  $c$ , then add this new matrix to  $A$ . With  $A$  on level 1 of the stack, you have two choices in creating the constant matrix with the same dimensions as  $A$ :

- enter the dimensions of  $A$  as a list  $\{ m \ n \}$ , where  $m$  is the number of rows and  $n$  is the number of columns of  $A$ , then key in the constant  $c$  and press **CON**.
- alternatively, and *preferably*, put a copy of  $A$  on level 2 with **ENTER** (or a command, like DUP, which performs **ENTER**), then key in the constant  $c$  and press **CON**.

**EXAMPLE.** To add 1.95 to each entry in matrix  $A = \begin{bmatrix} .78 & -.16 & .66 \\ -.89 & -.21 & .69 \end{bmatrix}$ , start with A on level 1. Press **ENTER** to copy this matrix onto level 2, then 1.95 **CON**. These last two keystrokes replace the copy of A on level 1 with the constant matrix  $\begin{bmatrix} 1.95 & 1.95 & 1.95 \\ 1.95 & 1.95 & 1.95 \end{bmatrix}$ . Now press **+** to add the constant matrix to the copy of A on level 2. The result is  $\begin{bmatrix} 2.73 & 1.79 & 2.61 \\ 1.06 & 1.74 & 2.64 \end{bmatrix}$ .

**TRANSPOSING.** With a matrix on level 1, the menu key **TRN** returns the conjugate transpose (*i.e.*, the conjugate of the transpose). Thus, if the matrix on level 1 is real, **TRN** returns its ordinary transpose. To obtain the ordinary transpose of a complex matrix, press **TRN** then **CONJ**. The **CONJ** command returns the complex conjugate of its input argument. On the 48S, you will find **TRN** on the MTH MATR menu and **CONJ** on the MTH PARTS menu. On the 28S, both commands are on the ARRAY menu.

**EXAMPLE.** With  $\begin{bmatrix} 2+3i & 7-4i & 3 \\ -i & 2 & 5+i \end{bmatrix}$  on level 1, press **TRN** to see the conjugate

transpose  $\begin{bmatrix} 2-3i & i \\ 7+4i & 2 \\ 3 & 5-i \end{bmatrix}$ . Now press **CONJ** to see the ordinary transpose

$\begin{bmatrix} 2+3i & -i \\ 7-4i & 2 \\ 3 & 5+i \end{bmatrix}$  of the original matrix.



## EXERCISES 2.3

1. Enter the matrix  $A = \begin{bmatrix} 3 & 2 & -1 & 5 & 0 \\ -6 & 7 & 3 & 0 & 1 \\ 2 & -4 & 8 & 7 & 9 \\ 5 & 2 & -6 & 1 & 3 \end{bmatrix}$  as a dimensioned array, and verify its

correct entry.

- (a) Disassemble A into its entries, drop the last 5 entries, and rearrange the first 15 entries (in row order) into a new 5×3 matrix B.
  - (b) Use **EDIT** to change the 7 in row three of B to 4 and the 8 in the last row to 0. Transpose the result.
  - (c) Now change the 9 in B to 11/8. Try this first using **EDIT** and see what happens - press **ON** to return to level 1 when you want to. Now use **PUT** to effect the change. Verify your success by viewing the result.
  - (d) Finally, use **NU.EL** to change the 5 to  $\sqrt{306.25}$ . After viewing the final matrix drop it from the stack.
2. (a) Create a 5×4 matrix  $A = (a_{ij})$  where  $a_{ij} = .ij$ .
- (b) Extract the submatrix B consisting of rows 2, 3 and 5.
- (c) Remove col 3 from B and square the result.
3. (a) Enter matrix A by starting with the constant matrix of all 0's and then inserting each non-0 entry

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 5 & 6 & 7 & 0 \end{bmatrix}.$$

- (b) Calculate  $A - A^3 + 2I$  using keystroke commands.

4. Enter  $A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}$ . Enlarge A by inserting an additional row on the bottom

and an additional column on the right. Do this as follows:

- (a) Insert a row of 4's, then a column of 5's.
- (b) Now start over with A, and first insert a column of 5's, then a row of 4's.
- (c) Are the results in (a) and (b) the same?

5. Enter  $A = \begin{bmatrix} 1 & -3 & 4 \\ 2 & 5 & 0 \\ 6 & -3 & 8 \end{bmatrix}$  and  $B = \begin{bmatrix} 7 & 0 & -1 \\ 5 & 3 & 2 \\ 9 & -6 & 0 \end{bmatrix}$ .

- (a) Form the partitioned matrix  $\begin{bmatrix} A \\ B \end{bmatrix}$ .
- (b) Form the partitioned matrix  $[A \ B]$ .

$$(\text{Hint: } \begin{bmatrix} X \\ Y \end{bmatrix}^T = \begin{bmatrix} X^T & Y^T \end{bmatrix} )$$

- (c) Form the partitioned matrix  $\begin{bmatrix} A & B \\ I & O \end{bmatrix}$ .
- (d) Form the partitioned matrix  $\begin{bmatrix} A & O \\ O & B \end{bmatrix}$ .

6. Enter  $A = \begin{bmatrix} 1 & 0 & 2 & 3 \\ 2 & 3 & 0 & -1 \\ 5 & -2 & 3 & 1 \end{bmatrix}$ .

(a) Form  $B = \begin{bmatrix} x^T \\ y^T \end{bmatrix}$  where  $x$  is column 2 and  $y$  is column 3 of  $A$ ; calculate  $BA$ .

(b) Now let  $C$  be the submatrix of  $A$  consisting of columns 1 and 4 of  $A$ ; calculate  $CB$ .

7. Let  $A = \begin{bmatrix} 1 & -2 & 3 & 5 & 4 \\ 7 & 9 & 0 & -1 & 3 \\ -3 & 8 & 6 & 2 & 1 \end{bmatrix}$  and  $B = \begin{bmatrix} 6 & 2 & -1 \\ 3 & 5 & 4 \\ -2 & 8 & 0 \\ 7 & 1 & 6 \\ 1 & -3 & 2 \end{bmatrix}$ .

(a) Get the submatrix  $C$  of  $B$  consisting of rows 2 and 4.

(b) Form the partitioned matrix  $[A \ C^T] = D$  and get the submatrix consisting of the odd-numbered columns.

(c) Interchange rows 1 and 3 of  $D$ , then columns 2 and 4 of the result.

8. Let  $A = \begin{bmatrix} 5+i & 2-3i & 1 \\ 4i & 6-i & 3+4i \end{bmatrix}$  and  $B = \begin{bmatrix} -3+i & 6 \\ 0 & 2-i \\ 4-3i & 1+i \end{bmatrix}$ .

(a) Find the conjugate transpose  $A^*$  of  $A$  and the transpose  $B^T$  of  $B$ .

(b) Calculate  $A^* + B$ ,  $A + B^T$ ,  $AA^*$ ,  $B^TB$  and  $(2 - 3i)A$ .

9. For  $A = \begin{bmatrix} 2-i & -2i & 3-2i \\ 1+5i & 3+2i & 5 \\ i & 6 & -1+2i \end{bmatrix}$ , find  $A^2 - 4A^* + 3A^T - I$ .

10. Given  $A = \begin{bmatrix} 7 + i & -5 + 6i & 3 + 2i & -9 - 5i \\ 11 - i & -2 + 2i & 1 & -2i \\ -3 + 6i & 4 + 4i & 6 - 8i & 8 - 2i \\ -6 & 7 + 4i & -3 - 3i & 9 + 5i \end{bmatrix}$ , separate A into its real and

imaginary parts with  $\boxed{\mathbb{C} \rightarrow \mathbb{R}}$ , transpose the real part and recombine it (use  $\boxed{\mathbb{R} \rightarrow \mathbb{C}}$ ) with the imaginary part. Then get column 3 of the result. (The commands  $\boxed{\mathbb{C} \rightarrow \mathbb{R}}$  and  $\boxed{\mathbb{R} \rightarrow \mathbb{C}}$  appear in the PGM OBJ menu of the 48S and in the REAL menu of the 28S.)

11. For this exercise, set your calculator to 3 FIX mode. Let  $A = \begin{bmatrix} .3 & .3 & .3 & .2 \\ .4 & .3 & .2 & 0 \\ .1 & .2 & .2 & .3 \\ .2 & .2 & .3 & .5 \end{bmatrix}$

and  $x = [.1 \ .2 \ .3 \ .4]^T$ .

(a) Examine the sequence  $A, A^2, A^3, \dots$  to find  $\lim_{n \rightarrow \infty} \{A^n\}$ .

(b) Examine the sequence  $Ax, A^2x, A^3x, \dots$  to find  $\lim_{n \rightarrow \infty} \{A^n x\}$ .

(c) What is the connection between the two limits in (a) and (b)?

12. Repeat parts (b) - (c) of exercise 11 using any vector  $x = [a \ b \ c \ d]^T$  of your choice where  $a + b + c + d = 1$ .

## 2.4 DETERMINANTS AND INVERSES.

With a square matrix A on stack level 1, pressing  $\boxed{\text{DET}}$  will return the determinant of A, and  $\boxed{1/x}$  will return  $A^{-1}$  in the event that  $\det A \neq 0$ . On the 48S  $\boxed{\text{DET}}$  is on the MTH MATR menu. On the 28S you will find  $\boxed{\text{DET}}$  on the third line of the ARRAY menu.

EXAMPLE. (a) Key in matrix  $A = \begin{bmatrix} 2 & 4 & 2 \\ 4 & 10 & 6 \\ 4 & 6 & 4 \end{bmatrix}$  and press **ENTER** **ENTER**

**ENTER** to put 3 copies of A on the stack.

(b) Press **DET** to show  $\det A = 8$ .

(c) Use **DROP**, then **1/x** to show  $A^{-1} = \begin{bmatrix} .5 & -.5 & .5 \\ 1 & 0 & -.5 \\ -2 & .5 & .5 \end{bmatrix}$ .

(d) Now press **\*** to check  $AA^{-1} = I$ .

However, as with any tool - no matter how sophisticated - some care must be exercised with these commands in order to obtain results that are mathematically correct. To make the point, you should complete the following experimental exercise.

### EXPERIMENTAL EXERCISE

(a) Enter and store  $A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 4 & 6 \\ 2 & 4 & 6 \end{bmatrix}$  and  $B = \begin{bmatrix} 1 & 1 & 1 \\ 3 & 6 & 4 \\ 3 & 6 & 4 \end{bmatrix}$ .

(b) Press **A** **ENTER** **ENTER**, then **DET** to see  $\det A$ . Since  $\det A = 0$ , A is singular and has no inverse. Check this by executing **DROP** to remove the 0, then **1/x** to see what happens. The error message "INV Error: Infinite result" alerts you to the fact that the calculator is unable to calculate an inverse for A.

(c) None of the above was unexpected; after all, A has two identical rows, so  $\det A$  is obviously 0 and thus A has no inverse.

- (d) Now put three copies of B on the stack and press  $\boxed{\text{DET}}$  to show  $\det B = 3 \times 10^{-11}$ . Use  $\boxed{\text{DROP}}$ , then  $\boxed{1/x}$  to obtain

$$B^{-1} = \begin{bmatrix} 2 & 6666666666.6 & -6666666666.7 \\ -1 & 3333333333.3 & -3333333333.3 \\ 0 & -9999999999.9 & 10000000000.0 \end{bmatrix}.$$

This looks suspicious, so

check by pressing  $\boxed{*}$  to show  $BB^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$ . Since  $BB^{-1} \neq I$ , this result is

clearly incorrect.

- (e) Recapture the last arguments with  $\boxed{\rightarrow}$   $\boxed{\text{ARG}}$  on the 48S or with  $\blacksquare$   $\boxed{\text{UNDO}}$  on the 28S, use  $\boxed{\text{SWAP}}$  to reverse the order of B and  $B^{-1}$ , then press  $\boxed{*}$  to see

$$B^{-1}B = \begin{bmatrix} .8 & -.4 & .4 \\ -.1 & .8 & .2 \\ .3 & .6 & .4 \end{bmatrix},$$

which is even worse!

- (f) Matrix B, like A, has two identical rows. This guarantees that  $\det B = 0$ , so B has no inverse. Why has the calculator failed us in this case, and not with matrix A?

One thing is clear: *use of the calculator to calculate determinants and matrix inverses may yield incorrect results.* Certainly, a little forethought given to matrices A and B would have been in order. That forethought may have told us that both matrices have 0 determinants and therefore have no inverses.

Why, then, did we not get these results for matrix B? The answer is due to the calculator's built-in routine for finding determinants and inverses and the fact that it

uses floating point arithmetic. Even without considering what that routine might be (we shall say more about it in Chapter 3) the above determinants come down to calculating

$$\det A = 2 * [ 6 - (2 * 3) ], \text{ and}$$

$$\det B = 3 * [ 4 - (3 * 4/3) ].$$

In exact arithmetic, both are 0. The floating point calculation of  $\det A$  is clearly 0, but for  $\det B$  we have

$$\begin{aligned} \det B &= 3 * [ 4 - (3 * 1.333333333333) ] \\ &= 3 * [ 4 - 3.999999999999 ] \\ &= 3 * [ 1 \times 10^{-11} ] \\ &= 3 \times 10^{-11}. \end{aligned}$$

And, given that the routine returned a non-0 value for  $\det B$ , it went on to calculate an inverse.

In view of all this, how should a beginning linear algebra student use the calculator to find determinants and inverses? Certainly, *calculating a determinant as a test for matrix invertibility is computationally impractical, due to the floating point environment of the calculator. As in the above example, the calculator may well return a non-0 value (the result of round-off) for the determinant of a singular matrix.* And the size of the determinant also has no bearing on the invertibility, for while multiplying an  $n \times n$  matrix  $A$  by a non-0 number  $k$  does not change the invertibility, the determinant of the resulting matrix is  $k^n(\det A)$ . In fact, there is little, if any, need to calculate determinants for matrices other than by hand for the low order, integer-valued matrices used in elementary courses to reinforce the learning of the basic theory. The inclusion of determinants in linear algebra courses is largely a carry-

over from late 19<sup>th</sup> century algebra and they play a key role in helping to develop certain theoretical concepts; *but they have no use in computational mathematics.* Regarding the calculation of matrix inverses, *it is seldom necessary to actually calculate a matrix inverse,  $A^{-1}$ .* For non-singular linear systems  $Ax = b$ , *there are better and more efficient ways to solve them than calculating  $x = A^{-1}b$ ,* and most other apparent needs to calculate  $A^{-1}$  can be circumvented by an appropriate reformulation of the problem. In summary: *the numerical calculation of matrix determinants and inverses is extremely sensitive to round-off error and choice of numerical algorithm in a floating point environment.* Although sophisticated, professional-level computer software is generally responsive to this sensitivity, *our advice is to proceed with extreme caution in a calculator environment and, whenever possible, avoid calculating determinants and inverses.*

To clean up calculator round-off error, we recommend that you use a simple program CLEAN.

#### 48S VERSION

<b>CLEAN</b>	(Clean-up routine)
<i>Input:</i>	level 2: an array level 1: a positive integer N
<i>Effect:</i>	cleans-up array entries which exhibit round-off. Rounds-off to N decimal places.
« RND »	



## 28S VERSION

**CLEAN** (Clean-up routine)*Input:* level 2: an array

level 1: a positive integer N

*Effect:* cleans-up array entries which exhibit round-off.

Rounds-off to N decimal places.

« FIX RND STD »

EXAMPLE. Put two copies of  $A = \begin{bmatrix} -8 & 5 & 6 \\ 0 & -4 & -3 \\ 0 & 0 & 7 \end{bmatrix}$  on the stack. Since A is triangular, its determinant is clearly 224, so  $A^{-1}$  exists. Use  $\boxed{1/x}$  to find  $A^{-1}$  on your calculator, then press  $\boxed{*}$  to see  $AA^{-1}$  with round-off error:

$$AA^{-1} = \begin{bmatrix} 1 & 0 & -.000000000001 \\ 0 & 1 & .000000000001 \\ 0 & 0 & .999999999999 \end{bmatrix}$$

Now press 11  $\boxed{\text{CLEAN}}$  to see  $AA^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$  correct to 11 decimal places.

## EXERCISES 2.4.

- Cofactor expansions tell us that a matrix with all integer entries will have an integer-valued determinant, so you may reason that, even if we failed to recognize that  $B = \begin{bmatrix} 1 & 1 & 1 \\ 3 & 6 & 4 \\ 3 & 6 & 4 \end{bmatrix}$  has two identical rows,  $\det B$  is an integer. Thus,

the calculator's result  $\det B = 3 \times 10^{-11}$  obviously shows a little round-off and, in fact,  $\det B = 0$ . But things aren't always that simple.

(a) Enter matrix

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 3 & 6 & 4 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 3 & 6 & 4 & 1 \\ 0 & 1 & 1 & 1 & 1 & 3 \\ 1 & 1 & 3 & 6 & 4 & 1 \end{bmatrix},$$

multiply it by 100 and ask you calculator to calculate the determinant of the result. is the calculator's result correct? Do you see a little round-off error?

(b) Examine rows 2 and 6 of matrix A. What does this tell you about  $\det A$ ? About  $\det[100A]$ ?

(c) Use the fact that for an  $n \times n$  matrix A,  $\det(kA) = k^n \det A$  to explain how round-off error contributed to the result in (a).

(d) Go back and read again the italicized statements in Section 2.4.

2. Suppose you ignore our advice about using  $x = A^{-1}b$  to solve a linear system  $Ax = b$ , and routinely apply this technique to solve  $Ax = b$  where

$$A = \begin{bmatrix} 1.3 & .6 & .35 \\ .6 & .4 & .3 \\ .35 & .3 & .25 \end{bmatrix} \quad \text{and} \quad b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

That is, put vector  $b$  on level 2 of the stack,  $A$  on level 1 and press  $\boxed{+}$  to find  $x = A^{-1}b$ .

(a) What is your calculated solution? Is it reasonable?

(b) Let  $U = \begin{bmatrix} 0 & .9 & .7 \\ 0 & .2 & .6 \\ 0 & 0 & .5 \end{bmatrix}$ . What does determinant theory tell you about

$\det U$ ? Put two copies of  $U$  on the stack and calculate  $UU^T$ . What does determinant theory tell you about  $\det(UU^T)$ ? What does this tell you about your answer to (a)?

(c) Go back and read again the italicized statements in Section 2.4.

## 2.5 MATRIX BUILDER ROUTINES.

In beginning courses in linear algebra it is especially helpful to manipulate a number of simple matrices. The matrices should be easy to use, have integer entries from  $Z_{10} = \{ 0, \pm 1, \pm 2, \dots, \pm 9 \}$ , and sometimes be of a special type: diagonal, tridiagonal, triangular or symmetric. Such matrices may be readily generated with the calculator and they can be used in a variety of discovery activities, as well as to help formulate, disprove or verify conjectures.

We have included several calculator programs for this purpose.

**RAN.Z** - builds a random matrix over  $Z_{10}$

**DIAG** - builds a random diagonal matrix over  $Z_{10}$

**U.TRI** - builds a random upper-triangular matrix over  $Z_{10}$

**L.TRI** - builds a random unit lower-triangular matrix over  $Z_{10}$

**TRIDIA** - builds a random tridiagonal matrix over  $Z_{10}$

**SYMM** - builds a random symmetric matrix over  $Z_{10}$ .

Each of these programs calls upon the calculator's random number generator RAND to construct a random matrix of the desired type over  $Z_{10}$ , with a random assignment of  $\pm$  signs to the entries. The calculator command RAND (found on the MTH PROB menu of the 48S and the REAL menu of the 28S) generates uniformly distributed pseudo-random numbers  $x$ , where each  $x$  lies in the range  $0 < x < 1$ . Each execution of RAND returns a value calculated from a *seed* based upon the previous RAND value, and the seed can be changed by using the command RDZ (adjacent to RAND in the proper menu). RDZ takes a real number  $z$  as a seed for the RAND command. If  $z$  is 0, the seed is based upon the system clock. After a complete memory reset, a built-in seed is used.

For classwork, it is often convenient to begin a particular discussion, example or exercise by having everyone in the class use the same non-0 seed. In this event, subsequent synchronous use of the RAND command by the class members will result in a common sequence of random numbers. Such will occur, for example, with a common non-0 seed and then synchronous use of any of the above six programs. Thus, with only a few simple keystrokes, each member of the class can generate the same random matrix over  $Z_{10}$ . We have found this to be an effective classroom procedure for class activities and for testing. Here are the six programs with illustrations of their use. They should all be stored in the BILDR subdirectory.

**RAN.Z** (Random Matrix Generator)*Inputs:* level 2: an integer M

level 1: an integer N

*Effect:* returns a random M by N matrix over  $Z_{10}$  with a random assignment of  $\pm$  to the entries.

```
« → M N « 1 M N * FOR I RAND 10 * FLOOR RAND 10 *
FLOOR → X « X 5 < -1 1 IFTE » EVAL * NEXT M N 2 →LIST
→ARRY » »
```

EXAMPLE. Press 6 RDZ to use the seed which begins this example, then press 4,  
5 RAN.Z to generate

```
[ [-2  4  3  0 -3 ]
  [ 6 -5  0 -8  8 ]
  [ 2 -2 -6  7  0 ]
  [ 7 -4  2  0 -5 ]]
```

**DIAG** (Diagonal Matrix Generator)*Input:* level 1: an integer N*Effect:* returns a random N by N diagonal matrix over  $Z_{10}$  with a random assignment of  $\pm$  to the entries.

```
« → N « 1 N FOR I 1 N FOR J IF I J - ABS 1 ≥ THEN 0 ELSE
RAND 10 * FLOOR RAND 10 * FLOOR → X « X 5 < -1 1 IFTE »
EVAL * END NEXT { N N } →ARRY » »
```

EXAMPLE. Press 5 RDZ to use the seed which begins this example, then press 4 DIAG to generate

$$\begin{bmatrix} -5 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & -7 \end{bmatrix}$$

**U.TRI** (Upper Triangular Matrix Generator)

*Input:* level 1: an integer N

*Effect:* returns a random N by N upper triangular matrix over  $Z_{10}$  with a random assignment of  $\pm$  to the entries.

```
« → N « 1 N FOR I 1 N FOR J IF I J > THEN 0 ELSE RAND 10
* FLOOR RAND 10 * FLOOR → X « X 5 < -1 1 IFTE » EVAL *
END NEXT NEXT { N N } →ARRY » »
```

EXAMPLE. Press 4 RDZ to use the seed which begins this example, then press 4 U.TRI to generate

$$\begin{bmatrix} -8 & 8 & 9 & -4 \\ 0 & -9 & 0 & 1 \\ 0 & 0 & 4 & -4 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

**L.TRI** (Unit Lower Triangular Matrix Generator)*Input:* level 1: an integer N*Effect:* returns a random N by N unit lower triangular matrix over  $Z_{10}$  with a random assignment of  $\pm$  to the entries.

```
« → N « 1 N FOR I 1 N FOR J IF I J ≤ THEN 0 ELSE RAND 10
* FLOOR RAND 10 * FLOOR → X « X 5 < -1 1 IFTE » EVAL *
END NEXT NEXT { N N } →ARRY DUP IDN + » »
```

EXAMPLE. Press 3 RDZ to use the seed which begins this example, then press 4

L.TRI to generate

```
[[ 1  0  0  0]
 [ 1  1  0  0]
 [-5  2  1  0]
 [-1  2 -7  1]]
```

**TRIDIA** (Tridiagonal Matrix Generator)*Input:* an integer N*Effect:* returns a random N by N tridiagonal matrix over  $Z_{10}$  with a random assignment of  $\pm$  to the entries

```
« → N « 1 N FOR I 1 N FOR J IF I J - ABS 1 > THEN 0 ELSE
RAND 10 * FLOOR RAND 10 * FLOOR → X « X 5 < -1 1 IFTE »
EVAL * END NEXT NEXT { N N } →ARRY » »
```

EXAMPLE. Press 3 **RDZ** to use the seed which begins this example, then press 5 **TRIDIA** to generate

$$\begin{bmatrix} 1 & -5 & 0 & 0 & 0 \\ 2 & -1 & 2 & 0 & 0 \\ 0 & -7 & 4 & 1 & 0 \\ 0 & 0 & -7 & 9 & 1 \\ 0 & 0 & 0 & 3 & 5 \end{bmatrix}.$$

**SYMM** (Symmetric Matrix Generator)

*Input:* level 1: an integer N

*Effect:* returns a random N by N symmetric matrix over  $Z_{10}$  with a random assignment of  $\pm$  to the entries.

*Required program:* DIAG

```
« DUP → N « 1 N FOR I 1 N FOR J IF I J ≥ THEN 0 ELSE
RAND 10 * FLOOR RAND 10 * FLOOR → X « X 5 < -1 1 IFTE »
EVAL * END NEXT NEXT { N N } →ARRY DUP TRN » 3 ROLL
DIAG + + »
```

EXAMPLE. Press 1 **RDZ** to use the seed which begins this example, then press 5 **SYMM** to generate

$$\begin{bmatrix} -7 & 7 & -9 & -8 & -5 \\ 7 & 7 & 8 & -1 & 0 \\ -9 & 8 & 1 & 5 & 3 \\ -8 & -1 & 5 & -2 & -3 \\ -5 & 0 & 3 & -3 & -5 \end{bmatrix}.$$



**EXERCISES 2.5.**

1. Generate and store a random  $4 \times 3$  matrix  $A$  and a random  $3 \times 5$  matrix  $B$ , both over  $Z_{10}$ .
  - (a) Separate  $A$  into its  $4 \times 1$  column matrices and store them in their natural order as  $A_1$ ,  $A_2$  and  $A_3$ .
  - (b) Separate  $B$  into its  $1 \times 5$  row matrices and store them in their natural order as  $B_1$ ,  $B_2$  and  $B_3$ .
  - (c) Calculate the matrix  $A_1B_1 + A_2B_2 + A_3B_3$  and compare with the matrix product  $AB$ .
  - (d) Repeat parts (a) - (c) using the columns and rows of a random  $3 \times 4$  matrix  $A$  and a random  $4 \times 5$  matrix  $B$  over  $Z_{10}$ .
  - (e) Summarize your findings and formulate a conjecture based upon them, being sure to write in complete English sentences. Be prepared to hand-in your write-up and to discuss it in class. (Note: each of the products  $A_iB_i$  is called an *outer product*.)
2.
  - (a) For  $n = 3, 4, 5$ : generate two random  $n \times n$  upper triangular matrices over  $Z_{10}$  and calculate their product. What do you observe? Would you expect similar results for lower triangular matrices? Why?
  - (b) For  $n = 3, 4, 5$ : generate two random  $n \times n$  unit lower triangular matrices over  $Z_{10}$  and calculate their product. What do you observe? Would you expect similar results for unit upper triangular matrices? Why?
  - (c) Is the product of two tridiagonal matrices also tridiagonal?

3. (a) For  $n = 3, 4, 5$ : generate a random  $n \times n$  unit lower triangular matrix  $L$  and find  $L^{-1}$  with  $\boxed{1/x}$ . Clean up any round-off error with 6  $\boxed{\text{CLEAN}}$ .

What do you observe?

- (b) Repeat (a) using arbitrary random upper triangular matrices.
- (c) Repeat (a) using random tridiagonal matrices.
- (d) Write-up your findings and formulate several conjectures based upon them, being sure to write in complete English sentences. Be prepared to hand-in your write-up and to discuss it in class.

**ORGANIZING YOUR PROGRAMS.** In working through this chapter you have encountered a variety of matrix editing programs. As you proceed through subsequent chapters, you will continue to meet programs which are pertinent to the topic of the chapter. To make efficient use of the calculator, programs should be grouped by topic and stored in a way that makes them easy to access. In particular, since the programs in sections 2.2 - 2.4 of this chapter are largely matrix editing routines, we recommend that you store them in a subdirectory called ED.1T. The matrix builder routines from the current section should be stored in a subdirectory called BILDR. Appendix 2, Program Organization, outlines the appropriate organizational scheme. Before proceeding to Chapter 3, you should turn to Appendix 2 and structure your calculator accordingly.

## CHAPTER 3

### SYSTEMS OF LINEAR EQUATIONS

Of the many topics studied in elementary linear algebra, none is more fundamental than systems of linear equations. Such systems arise in practically every field of mathematical application and their importance in beginning courses cannot be overemphasized. For brevity, we shall refer to systems of linear equations as *linear systems* and denote their matrix formulation as  $Ax = b$ .

The most popular methods for dealing with linear systems in introductory linear algebra courses are the elimination methods, consisting of several variants of *Gaussian elimination* with back substitution. Many beginning courses blur the distinction between these variants in the interest of expediency. But with an eye toward subsequent study in linear analysis or numerical methods and the use of professional elimination codes, it is important that students carefully distinguish between the traditional Gaussian elimination algorithm, the back substitution process, partial pivoting and Gauss-Jordan reduction. Likewise, it is important to understand Gaussian elimination for square matrices as a factoring process which factors a matrix  $A$  into triangular factors,  $A = LU$ .

#### 3.1 GAUSSIAN ELIMINATION

In its traditional form, the Gaussian elimination algorithm for solving a square linear system  $Ax = b$  adds suitable multiples of one equation to the others with the goal of obtaining an equivalent upper triangular system  $Ux = b'$ , where the coefficient matrix  $U$  has 0's below the diagonal. It may be necessary to interchange equations at various times for the elimination process to continue. Back substitution then solves  $Ux = b'$  systematically by solving the last equation for its single

unknown, then putting this value into the next-to-last equation and solving for the next-to-last unknown, and so on until all values for the unknowns have been determined. All this is usually carried out without reference to the unknowns by working with the augmented matrices  $[A|b]$  and  $[U|b']$ . Computationally, the only source of error is round-off, induced by the computational device itself. It is especially important that students view the elimination as an orderly, arithmetic process which proceeds in a top-to-bottom, left-to-right fashion.

Once a basic understanding of Gaussian elimination has been established and several examples have been worked by hand, the calculator can be used to efficiently perform the row operations which transform  $[A|b]$  into  $[U|b']$ . Program ELIM, given below, pivots on a specified entry - the pivot - to produce 0's below that entry. The program is written to handle both real and complex matrices and can be used, more generally, to convert a matrix to row-echelon form. Notice that the program will abort and print the error message "PIVOT ENTRY IS 0" in case the intended pivot is 0.

**ELIM** (Gaussian elimination)

*Inputs:* level 3: a matrix  
level 2: an integer K  
level 1: an integer L

*Effect:* pivots on the (K,L)-entry of the matrix to produce 0's below the pivot.

```
« → A K L « IF 'A(K, L)' EVAL 0 == THEN "PIVOT ENTRY IS 0"
ELSE A SIZE 1 GET → M « M IDN 'A(1, 1)' EVAL TYPE IF THEN
DUP 0 CON R→C END K M FOR I 'A(I, L)' EVAL { I K } SWAP PUT
NEXT INV { K K } 1 PUT A * » 8 RND END » »
```

**Note:** The command 8 RND at the end of the program rounds the display to 8 decimal places to clean up round-off error. On the 28S, the appropriate command is 8 FIX RND STD.

Store this program as ELIM in subdirectory GAUSS (see Appendix 2).

**EXAMPLE 1.** Apply Gaussian elimination to the linear system

$$2x + 4y + 8z = 6$$

$$x - y + 2z = 3$$





$$4x - y + 7z = 8$$

$$\begin{bmatrix} 2 & 4 & 8 & 6 \\ 1 & -1 & 2 & 3 \\ 4 & -1 & 7 & 8 \end{bmatrix} \xrightarrow{1,1 \text{ ELIM}} \begin{bmatrix} 2 & 4 & 8 & 6 \\ 0 & -3 & -2 & 0 \\ 0 & -9 & -9 & -4 \end{bmatrix} \xrightarrow{2,2 \text{ ELIM}} \begin{bmatrix} 2 & 4 & 8 & 6 \\ 0 & -3 & -2 & 0 \\ 0 & 0 & -3 & -4 \end{bmatrix}$$

Back substitution then gives the solution  $[-5/9 \ -8/9 \ 4/3]^T$ .

To be genuinely useful, program ELIM must be used together with two other routines, program RO.KL which interchanges rows K and L of a matrix, and program BACK which performs the back substitution process. Program RO.KL may be found at the end of section 2.2 of Chapter 2. You should keep a copy of RO.KL in your GAUSS subdirectory.

**BACK** (Back substitution)*Inputs:* level 2: an  $n \times n$  upper triangular matrix  $U$ level 1: an  $n$ -vector  $b$ *Effect:* Solves the linear system  $Ux=b$  by back substitution.Solves for  $x_n$  and halts until you press 

 (or   on the 28S), then  
 backsolves for  $x_{n-1}$  and halts, etc. After  $x_n, x_{n-1}, \dots,$   
 $x_1$  are on the stack, a final  returns  
 $x = [x_1, x_2, \dots, x_n]$ .

```
« → A b « A SIZE 1 GET → N « { N } 0 CON 'A(1,1)' EVAL TYPE
IF THEN DUP R→C END → X « ED.IT N 1 A FOR J 'b(J)' EVAL A J
C.ROW SWAP DROP X DOT - 'A(J,J)' EVAL / 8 RND GAUSS HALT
ED.IT DUP X { J } ROT PUT 'X' STO -1 STEP GAUSS N DROPN X »
» » »
```

Store this as program **BACK** in subdirectory **GAUSS**.

**COMMENTS:** **BACK** calls upon program **C.ROW** which is assumed stored in the **ED.IT** subdirectory. Thus, **BACK** switches to the **ED.IT** subdirectory to use **C.ROW** and then switches back to the **GAUSS** subdirectory. **BACK** halts after each backsolve step so that beginning students may exercise the desired control over the entire back substitution process. For the 28S, replace 8 RND with 8 FIX RND STD.

**EXAMPLE 2.** Backsolve  $2x_1 + 6x_2 - 4x_3 - 13x_4 = -24$   
 $-3x_2 + 4x_3 + 3x_4 = -3$   
 $4x_3 - x_4 = -14$   
 $-2x_4 = -4$

Put  $\begin{bmatrix} 2 & 6 & -4 & -13 \\ 0 & -3 & 4 & 3 \\ 0 & 0 & 4 & -1 \\ 0 & 0 & 0 & -2 \end{bmatrix}$  on level 2, and  $\begin{bmatrix} -24 & -3 & -14 & -4 \end{bmatrix}$  on level 1. After using

**BACK** and four applications of **CONT**, the solution is found to be  $\begin{bmatrix} -2 & -1 & -3 & 2 \end{bmatrix}$ .

To effectively use the calculator to apply Gaussian elimination and back substitution to a (non-singular) linear system  $Ax=b$ , first apply ELIM to the augmented matrix  $[A|b]$  to obtain an equivalent upper triangular system  $[U|b']$ , split off vector  $b'$  from  $U$  with program SPLIT (below), then apply BACK to the arguments  $U$  and  $b'$ .

**SPLIT** (Split off last column)

*Input:* level 1: a matrix

*Effect:* splits off the last column of the matrix, returns the column to level 1 and the modified matrix to level 2.

« ED.IT DUP SIZE 2 GET DUP 3 ROLLD C.COL 3 ROLLD SWAP  
DELCOL SWAP GAUSS »

Store this as SPLIT in subdirectory GAUSS.

COMMENT. SPLIT calls upon programs C.COL and DELCOL in subdirectory ED.IT, so it switches to ED.IT and then back to GAUSS.

**EXAMPLE 3.** To use Gaussian elimination and back substitution to solve the linear system

$$\begin{array}{rclclcl} 5x_1 & - & 9x_2 & + & 16x_3 & + & 6x_4 & = & 48 \\ -5x_1 & + & 9x_2 & - & 16x_3 & - & 8x_4 & = & -45 \\ 10x_1 & - & 9x_2 & + & 24x_3 & + & 8x_4 & = & 72 \\ -5x_1 & - & 9x_2 & + & 8x_3 & + & 8x_4 & = & 3 \end{array}$$

begin with the augmented matrix  $[A|b]$

$$\begin{bmatrix} 5 & -9 & 16 & 6 & 48 \\ -5 & 9 & -16 & -8 & -45 \\ 10 & -9 & 24 & 8 & 72 \\ -5 & -9 & 8 & 8 & 3 \end{bmatrix}$$

on level 1. The sequence of commands 1,1 **ELIM**; 2, 3 **RO.KL**; 2, 2 **ELIM**; 3, 4 **RO.KL** returns the equivalent upper triangular system  $[U|b]$

$$\begin{bmatrix} 5 & -9 & 16 & 6 & 48 \\ 0 & 9 & -8 & -4 & -24 \\ 0 & 0 & 8 & 6 & 3 \\ 0 & 0 & 0 & -2 & 3 \end{bmatrix}$$

Press **SPLIT** to split off the last column. Then, **BACK** followed by four applications of **CONT** show  $[3 \ -2 \ 1.5 \ -1.5]$  as the solution.

We shall soon provide a calculator routine for the variant of Gaussian elimination known as *Gauss-Jordan reduction*, the effect of which is to do both elimination and back substitution in one routine.

We have already seen that row interchanges may be needed in order for Gaussian elimination to proceed to its natural conclusion. In so doing we are simply avoiding 0 pivots. For the practical real-world solution of large-scale linear systems, it is just as important to avoid using pivots which are extremely small, at least in relation to the other elements in the pivot column. This is because division by small numbers in floating point arithmetic may ultimately induce considerable error. To avoid this, a common pivoting strategy is to choose as the pivot any element in the pivot column whose absolute value is maximum. This so-called *partial pivoting strategy* is difficult to illustrate on the calculator because of its use of 12 digit mantissas. Nevertheless, it is advisable that beginning students occasionally adopt



the partial pivoting strategy by using the RO.KL program to reinforce their understanding of this technique.

### EXERCISES 3.1

1. Use partial pivoting to find row echelon matrices row-equivalent to each of the following matrices:

$$A = \begin{bmatrix} 2 & 2 & 5.6 & 6.8 \\ 4 & 3 & 7 & 5 \\ -2 & 1 & -5 & -1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 2 & 1 & 3 \\ 1 & 3 & -4 & -8 & 6 \\ 2 & 7 & -10 & -19 & 13 \end{bmatrix},$$

$$C = \begin{bmatrix} i & -1 & 1 & 1 \\ -i & 0 & i & 0 \\ 0 & 1 & 1+i & 0 \\ 1 & -i & i & -i \end{bmatrix}, \quad D = \begin{bmatrix} 4 & 5 & -1 & 3 & -2 \\ 8 & 9 & 0 & 7 & 6 \\ -4 & 3 & 2 & 8 & -7 \\ 5 & 1 & 1 & -6 & 2 \\ 3 & 10 & 1 & -1 & 9 \end{bmatrix}$$

2. Solve the following linear systems using Gaussian elimination with partial pivoting and back substitution.

$$(a) \quad 3x_1 + x_2 - 3x_3 = 4$$

$$4x_1 + 2x_2 - 2x_3 = -3$$

$$5x_1 + 6x_2 + 8x_3 = 8$$

$$(b) \quad -x_1 - 4x_2 - 3x_3 + 3x_5 = 2$$

$$x_1 + 2x_2 + 2x_3 + 4x_4 - x_5 = 0$$

$$x_3 - 12x_4 + 5x_5 = 3$$

$$2x_1 + 4x_2 + 4x_3 + x_5 = 2$$

$$x_1 + 2x_2 + x_3 + 4x_4 - x_5 = -2$$

$$\begin{aligned}
 \text{(c)} \quad & 3x_1 - 6x_2 + 9x_3 - 9x_4 = 57 \\
 & -24x_1 + 56x_2 - 80x_3 + 69x_4 = -482 \\
 & 12x_1 - 72x_2 + 83x_3 - 19x_4 = 383 \\
 & 24x_1 + 16x_2 + 8x_3 - 103x_4 = 262
 \end{aligned}$$

$$\begin{aligned}
 \text{(d)} \quad & x_1 + 2x_2 - 3x_3 + 4x_4 = -1 \\
 & 2x_1 + 6x_2 + 10x_3 - 8x_4 = 2 \\
 & x_1 + 2x_2 - 2x_3 + 5x_4 = 6 \\
 & x_1 + 3x_2 + 5x_3 + 4x_4 = 1
 \end{aligned}$$

$$\begin{aligned}
 \text{(e)} \quad & x_1 + 2x_2 \quad \quad + 3x_4 + x_5 = 2 \\
 & 3x_1 + 6x_2 - 2x_3 + 7x_4 + 5x_5 = 6 \\
 & x_1 + 3x_2 - x_3 + 3x_4 + 2x_5 = 2 \\
 & x_1 + x_2 - 2x_3 + 2x_4 + 3x_5 = 2 \\
 & -x_1 + 2x_2 \quad \quad \quad - 3x_5 = -4
 \end{aligned}$$

### 3.2 LU-FACTORIZATIONS

In addition to recognizing Gaussian elimination as an orderly process for converting a square matrix to upper triangular form, it is important that students also understand it as a factorization process. In its simplest form - when no row interchanges are involved - the coefficient matrix  $A$  of a linear system  $Ax = b$  is factored into two triangular matrices  $A = LU$ , where  $L$  is lower triangular and has 1's on its diagonal (*unit* lower triangular). This viewpoint is not only interesting from a purely algebraic standpoint; it also lies at the heart of many modern computer codes (such as those in the LINPACK library) used to handle linear systems. Quite properly, most linear algebra courses today include discussions of LU-factorizations, and this topic is one well-suited to calculator enhancement.

When the matrix  $A$  in a linear system  $Ax = b$  can be brought to upper triangular form  $U$  by Gaussian elimination without row interchanges, then  $A = LU$  where  $L$  is lower triangular with 1's along its diagonal and the entries below the diagonal are the negatives of the multipliers used in the elimination process. For example, if 3 times row 1 was added to row 2 to produce a 0 in the (2, 1)-entry of  $U$ , then the (2, 1)-entry of  $L$  is -3. When row interchanges are needed to avoid 0 pivots, then  $A = LU$  is no longer valid; it is replaced by a factorization of the form  $PA = LU$  where  $P$  is a *permutation matrix* which accounts for the various row interchanges.

In the simplest case - no row interchanges - it is easy to modify program ELIM so that it will record the lower triangular entries of  $L$  beneath the diagonal entries of  $U$ . But the modifications become more involved in the presence of row interchanges. Since we are primarily interested in the *pedagogical aspects* of LU-factorizations, we have chosen to use a calculator program which the student must control at each step, just as in the case of hand calculations. Program LU, given below, is but a slight modification of ELIM. In addition to performing the basic elimination step LU stores the negatives of the multipliers below the diagonal in a matrix called "ELL", which initially is the 0 matrix. Program MAKL creates the initial ELL. If row interchanges are needed, the proper use of RO.KL must be made with both ELL and  $U$  in order to continue. At the end, the calculator shows  $U$ , and the lower triangle of  $L$  in matrix ELL. As before, complex matrices are allowed.

**LU** (Used to get LU-factorizations)

*Inputs:* As a stored variable: a variable 'ELL', obtained from program MAKL (below) and containing a 0 matrix.

level 3: a square matrix A

level 2: an integer K

level 1: the integer K

*Effect:* Pivots on the (K, K)-entry to return a row-equivalent matrix with 0's below the pivot; also puts the negatives of the multipliers into column K of ELL below the diagonal. Press ELL to view ELL. Used iteratively to obtain an LU-factorization.

```
« → A K L « IF 'A(K, L)' EVAL 0 == THEN "PIVOT ENTRY IS 0"
ELSE A SIZE 1 GET → M « M IDN 'A(1, 1)' EVAL TYPE IF THEN
DUP 0 CON R→C END K M FOR I 'A(I, L)' EVAL { I K } SWAP PUT
NEXT INV { K K } 1 PUT DUP A * SWAP K 1 + M FOR I DUP { I K }
GET NEG 8 RND 'ELL(I, K)' STO NEXT DROP » 8 RND END » »
```

Store the program as variable LU in the GAUSS subdirectory.

**NOTE:** For the 28S version, replace both occurrences of 8 RND with 8 FIX RND STD.

**MAKL** (Make ELL and P)*Input:* level 1: a square matrix A*Effect:* Creates a variable ELL containing a 0 matrix, and a variable P containing an identity matrix, both the same size as A. Used as the initial start-up to obtain an LU-factorization.

« DUP 0 CON 'ELL' STO DUP IDN 'P' STO »

Store the program as variable MAKL in the GAUSS subdirectory next to LU.

**EXAMPLE 1.** To get an LU-factorization of  $A = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 7 & 1 \\ -1 & 10 & -3 \end{bmatrix}$ , start with matrix A on level 1.

Step 1: Press **MAKL** to create starting matrices ELL and P in user memory. (You may press **ELL** to verify that you have stored a 3×3 zero matrix as ELL, and then **P** to see a 3×3 identity matrix. Press **DROP** twice to remove these from the stack.)

Step 2: Press 1, 1 **LU** to see  $\begin{bmatrix} 1 & 2 & 1 \\ 0 & 3 & -1 \\ 0 & 12 & -2 \end{bmatrix}$ , then **ELL** to see  $\begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}$ .

Now **DROP** this last matrix from the stack.

Step 3: Press 2, 2  $\boxed{\text{LU}}$  to see 
$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 3 & -1 \\ 0 & 0 & 2 \end{bmatrix} = U.$$
 Now press  $\boxed{\text{ELL}}$  to see

$$\begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ -1 & 4 & 0 \end{bmatrix}.$$
 Execute 3  $\boxed{\text{IDN}}$   $\boxed{+}$  to see 
$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 4 & 1 \end{bmatrix} = L.$$

Step 4: (Check) Press  $\boxed{\text{SWAP}}$   $\boxed{*}$  to check that  $LU = A$ . Now purge ELL and P (which was not used in this example).

EXAMPLE 2. Get an LU-factorization of  $A = \begin{bmatrix} 2 & 3 & -1 & 2 \\ -4 & -6 & 2 & 1 \\ 2 & 4 & -4 & 1 \\ 4 & 8 & 2 & 7 \end{bmatrix}.$

Step 1: Enter A onto level 1, press  $\boxed{\text{MAKL}}$  to create appropriate starting

matrices ELL and P, and press 1, 1  $\boxed{\text{LU}}$  to see 
$$\begin{bmatrix} 2 & 3 & -1 & 2 \\ 0 & 0 & 0 & 5 \\ 0 & 1 & -3 & -1 \\ 0 & 2 & 4 & 3 \end{bmatrix}.$$

Step 2: Since the (2, 2)-entry of this last matrix is 0, we must interchange row 2 with some lower row, say row 3. Thus press 2, 3  $\boxed{\text{RO.KL}}$  to effect the interchange, then bring ELL to level 1 with  $\boxed{\text{ELL}}$ , make the same row interchange and store the result in ELL. Now bring P to level 1 with  $\boxed{\text{P}}$ , make the same row interchange and store the result in P.

Step 3: Now execute 2, 2  $\boxed{\text{LU}}$  to see

$$\begin{bmatrix} 2 & 3 & -1 & 2 \\ 0 & 1 & -3 & -1 \\ 0 & 0 & 0 & 5 \\ 0 & 0 & 10 & 5 \end{bmatrix}.$$

Step 4: Interchange rows 3 and 4 with 3, 4  $\boxed{\text{RO.KL}}$ , bring ELL to level 1 with  $\boxed{\text{ELL}}$  and make the same interchange, and store in ELL. Bring P to level 1 with  $\boxed{\text{P}}$ , make the same interchange and store in P.

Step 5: See  $U = \begin{bmatrix} 2 & 3 & -1 & 2 \\ 0 & 1 & -3 & -1 \\ 0 & 0 & 10 & 5 \\ 0 & 0 & 0 & 5 \end{bmatrix}$  and  $\text{ELL} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 \\ -2 & 0 & 0 & 0 \end{bmatrix}.$

Get  $L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 2 & 1 & 0 \\ -2 & 0 & 0 & 1 \end{bmatrix}$  with 4  $\boxed{\text{IDN}}$   $\boxed{+}$ , then do  $\boxed{\text{SWAP}}$   $\boxed{*}$  to see

$$\text{LU} = \begin{bmatrix} 2 & 3 & -1 & 2 \\ 2 & 4 & -4 & 1 \\ 4 & 8 & 2 & 7 \\ -4 & -6 & 2 & 1 \end{bmatrix}.$$

Step 6: (Check)  $\text{PA} = \text{LU}$  where  $\text{P} = \text{P}_{34}\text{P}_{23}$ . Since P is a permutation matrix, we know that  $\text{P}^{-1} = \text{P}^T$ . Thus  $\text{P}^{-1}\text{LU} = \text{P}^T\text{LU} = \text{A}$ . Recall P to level 1 and get  $\text{P}^T$ , SWAP levels with LU and then use  $\boxed{*}$  to see  $\text{P}^T\text{LU} = \text{A}$

Although most elementary texts present discussions of LU-factorizations, it will not hurt to briefly summarize why this topic is so important.

- (i) As noted earlier, factorizations such as  $A = LU$  and  $PA = LU$  into triangular matrices lie at the heart of modern computer codes for dealing with large, square, linear systems.
- (ii) In particular, in the case of  $A = LU$ , all the information regarding Gaussian elimination on  $A$  is stored in the factors  $L$  and  $U$ . Matrix  $L$  maintains a record of the multipliers used in the elimination process and  $U$  records the results of that elimination. Thus,  $L$  and  $U$  may be viewed as the storehouses of information about  $A$  which may be exploited later in a variety of situations. With  $PA = LU$ ,  $P$  records the row interchanges.
- (iii) Once we have  $A = LU$  we can solve  $Ax = b$  for different  $b$ 's by first using forward substitution to solve  $Ly = b$  for  $y$ , then back substitution to solve  $Ux = y$  for  $x$ . (In the case of  $PA = LU$ , we solve  $Ly = Pb$  in the first step. Indeed, this is the preferred method for solving large scale linear systems. Why? Assume that  $A$  is  $n \times n$  and that both  $A^{-1}$  and the factors  $L$  and  $U$  are available. Using  $A^{-1}$  to obtain  $x = A^{-1}b$  requires  $n^2$  multiplications. Solving  $Ly = b$  for  $y$  by forward substitution and then solving  $Ux = y$  for  $x$  by back substitution also requires  $n^2$  multiplications. But the difference is seen in comparing the number of multiplications required to obtain  $A^{-1}$  to the number of multiplications required to obtain the factors  $L$  and  $U$ :  $n^3$  verses  $\frac{n^3}{3}$ . For large  $n$ , the savings in using  $L$  and  $U$  is substantial. For example, using  $n = 1,000$  (not an unrealistic occurrence in some of today's applications) and assuming your computer performs  $10^6$  multiplications per second (very fast!), the savings in using  $LU$  over  $A^{-1}$  is over 11 seconds of computer time. And if we have multiple  $b$ 's to use in  $Ax = b$ , these savings rapidly accumulate.








To apply forward substitution to  $Ly = Pb$  on the calculator, use the following program FWD.

**FWD** (Forward substitution)

*Inputs:* level 2: an  $n \times n$  lower triangular matrix L

level 1: an  $n$ -vector b

*Effect:* Solves the linear system  $Lx = b$  by forward substitution. Solves for  $x_1$  and halts until you press

  (or   on the 28S), then solves for  $x_2$  and halts, etc. After  $x_1, x_2, \dots, x_n$  are on the stack, a final  returns  $x = [x_1, x_2, \dots, x_n]$ .

```
« → A b « A SIZE 1 GET → N « { N } 0 CON 'A(1,1)' EVAL TYPE
IF THEN DUP R→C END → Y « ED.IT 1 N FOR J 'b(J)' EVAL A J
C.ROW SWAP DROP Y DOT - 'A(J,J)' EVAL / 8 RND GAUSS HALT
ED.IT DUP Y { J } ROT PUT 'Y' STO NEXT GAUSS N DROPN Y » »
» »
```

Store this as program FWD in subdirectory GAUSS.

**COMMENTS:** FWD calls upon program C.ROW which is assumed stored in the ED.IT subdirectory. Thus, FWD switches to the ED.IT subdirectory to use C.ROW and then switches back to the GAUSS subdirectory. FWD halts after each forward-solve step so that beginning students may exercise the desired control over the entire forward substitution process. For the 28S, replace 8 RND with 8 FIX RND STD.

EXAMPLE 3. To solve

$$\begin{aligned} 2x_1 + 3x_2 - x_3 + 2x_4 &= 1 \\ -4x_1 - 6x_2 + 2x_3 + x_4 &= 2 \\ 2x_1 + 4x_2 - 4x_3 + x_4 &= 3 \\ 4x_1 + 8x_2 + 2x_3 + 7x_4 &= 4 \end{aligned}$$

by using an LU-factorization, we first obtain a  $PA = LU$  factorization of the coefficient matrix

$$A = \begin{bmatrix} 2 & 3 & -1 & 2 \\ -4 & -6 & 2 & 1 \\ 2 & 4 & -4 & 1 \\ 4 & 8 & 2 & 7 \end{bmatrix}.$$

Since  $A$  is the matrix of our last example, we shall use the  $P$ ,  $L$  and  $U$  obtained there:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 2 & 1 & 0 \\ -2 & 0 & 0 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 2 & 3 & -1 & 2 \\ 0 & 1 & -3 & -1 \\ 0 & 0 & 10 & 5 \\ 0 & 0 & 0 & 5 \end{bmatrix}.$$

Let  $b = [1 \ 2 \ 3 \ 4]$ . To solve  $Ly = Pb$  for  $y$  by forward substitution, calculate  $Pb = [1 \ 3 \ 4 \ 2]$ . Then, with  $L$  on level 2 and  $Pb$  on level 1, **FWD** and four applications of **CONT** show  $y$  to be  $[1 \ 2 \ -2 \ 4]$ . Then with  $U$  on level 2 and  $[1 \ 2 \ -2 \ 4]$  on level 1, **BACK** and four applications of **CONT** show the solution  $x$  of  $Ax = b$  to be  $[-2.1 \ 1 \ -.6 \ .8]$ .

Finally, the factorization  $PA = LU$  of a matrix  $A$  is not unique. Consider, for instance, the effect of choosing different pivots in Gaussian elimination. Each choice of a pivot will give rise to a new LU-factorization. Some examples are included in the exercises. We recommend that you keep a copy of **CLEAN** in GAUSS to use as necessary.

## Exercises 3.2

1. Find an LU factorization of each of the following matrices; do not interchange rows. Check your answers; use CLEAN as necessary.

$$(a) \quad A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 5 \\ 12 & 3 & 4 \end{bmatrix}$$

$$(b) \quad B = \begin{bmatrix} 2 & 2 & 8 \\ 1 & 3 & -2 \\ 2 & -1 & 5 \end{bmatrix}$$

$$(c) \quad C = \begin{bmatrix} 3 & 2 & -1 & -2 \\ 1 & 1 & 1 & 0 \\ -1 & 1 & 2 & 3 \\ 3 & 4 & 2 & 1 \end{bmatrix}$$

$$(d) \quad D = \begin{bmatrix} 2 & 1+i & 3+4i \\ 1-i & 4 & 2-3i \\ 3-4i & 2+3i & 5 \end{bmatrix}$$

2. Re-do Exercise 1 using partial pivoting *throughout*.
3. For each of the following matrices A:
- (i) find a permutation matrix P and matrices L and U so that  $PA = LU$  is an LU factorization; if you need to interchange rows, use the first available row. Check your answers; use CLEAN as necessary.
  - (ii) Use your  $PA = LU$  factorization to solve the linear system  $Ax=b$  for the given b. Check your answers, using CLEAN as necessary.

$$(a) \quad A = \begin{bmatrix} 2 & 4 & 6 \\ 1 & 2 & 9 \\ 3 & 5 & 7 \end{bmatrix}, \quad b = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

$$(b) \quad A = \begin{bmatrix} 3 & 6 & 3 & 2 \\ 2 & 4 & 5 & 2 \\ 3 & 7 & 8 & 5 \\ 2 & 9 & 3 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ 4 \\ -1 \\ 2 \end{bmatrix}$$

$$(c) \quad A = \begin{bmatrix} -2 & 4 & 0 & 1 \\ 3 & -5 & 2 & 7 \\ 1 & 3 & 10 & -6 \\ -4 & 5 & 1 & -1 \end{bmatrix}, b = \begin{bmatrix} 8 \\ 6 \\ 8 \\ 6 \end{bmatrix} \quad (d) \quad A = \begin{bmatrix} 0 & -2 & 3 & 1 & 6 \\ 1 & 3 & -5 & 2 & -1 \\ -2 & -2 & 4 & 1 & 3 \\ 3 & 1 & -3 & 3 & 2 \\ -1 & 1 & 5 & 1 & 2 \end{bmatrix}, b = \begin{bmatrix} -1 \\ 2 \\ 0 \\ 2 \\ -1 \end{bmatrix}$$

4. Use LU-factorizations to solve  $Ax=b$ . Check your answers.

$$(a) \quad A = \begin{bmatrix} 5 & -9 & 16 & 6 \\ -5 & 9 & -16 & -8 \\ 10 & -9 & 24 & 8 \\ -5 & -9 & 8 & 8 \end{bmatrix}, b = \begin{bmatrix} 48 \\ -45 \\ 72 \\ 3 \end{bmatrix} \quad (\text{see Example 3, Section 3.1})$$

$$(b) \quad A = \begin{bmatrix} 2 & -1 & -6 & 3 \\ -4 & 2 & 12 & -7 \\ 6 & 2 & -3 & 4 \\ -4 & 0 & 9 & -7 \end{bmatrix}, b = \begin{bmatrix} -10 \\ 25 \\ -20 \\ 25 \end{bmatrix}$$

5. (a) Find an LU-factorization for each of the following tridiagonal matrices and note the structure of L and U.

(b) Formulate a conjecture based upon your observations.

$$A = \begin{bmatrix} -4 & 2 & 0 & 0 \\ -8 & 2 & 1 & 0 \\ 0 & 6 & -2 & -2 \\ 0 & 0 & 5 & -7 \end{bmatrix} \quad B = \begin{bmatrix} 2 & -3 & 0 & 0 & 0 \\ 4 & -7 & 4 & 0 & 0 \\ 0 & 3 & -9 & -5 & 0 \\ 0 & 0 & 12 & -22 & 6 \\ 0 & 0 & 0 & -10 & 34 \end{bmatrix}$$

$$C = \begin{bmatrix} 3 & -2 & 0 & 0 & 0 & 0 \\ 27 & -23 & 4 & 0 & 0 & 0 \\ 0 & -40 & 33 & -2 & 0 & 0 \\ 0 & 0 & -4 & 9 & 5 & 0 \\ 0 & 0 & 0 & -5 & -21 & -2 \\ 0 & 0 & 0 & 0 & -16 & 4 \end{bmatrix}$$

### 3.3 GAUSS-JORDAN REDUCTION

Earlier, in connection with linear systems, we remarked that we would soon provide a calculator routine for the variant of Gaussian elimination known as *Gauss-Jordan reduction*, the effect of which is to do both elimination and back substitution in one routine. Such a routine will provide a tool which is extremely useful not only for classwork and homework dealing with linear systems per se, but also with other concepts associated with these systems (e.g., linear independence). Though Gaussian elimination with back substitution is more efficient than Gauss-Jordan reduction for dealing with linear systems in general, and is certainly the preferred method in professional computer libraries, most students prefer to use Gauss-Jordan reduction for the small-scale problems employed to learn the basic concepts.

Gauss-Jordan reduction differs from Gaussian elimination in two ways:

- (i) all the pivots are converted to 1.
- (ii) the basic pivot process is used to produce 0's both below *and above* the pivot element.

Thus, Gauss-Jordan reduction, when applied to a non-0 matrix  $A$ , yields what is popularly called the **reduced row echelon form (RREF)** of  $A$ :

- (a) any 0 rows lie at the bottom;
- (b) the first non-0 entry in any non-0 row (the pivot) is a 1, and lies to the right of the pivot in any preceding row;
- (c) the pivot is the only non-0 entry in its column.

The reduced row echelon form of  $A$  is important because it represents the *ultimate* we can get from  $A$  by applying elementary row operations. As such, it is *uniquely* associated with  $A$ ; that is, each non-0 matrix  $A$  has one and only one RREF.

When Gauss-Jordan reduction is applied to the augmented matrix  $[A|b]$  of a linear system  $Ax = b$  we obtain an equivalent linear system  $Ux = b'$  whose augmented matrix  $[U|b']$  is the RREF and whose solutions are practically obvious. Specifically, any variable (or unknown) associated with a pivot is called a *pivot variable* while the other variables, if any, are called *free variables*. If the last non-0 row of  $[U|b']$  looks like  $[0 \ 0 \ \dots \ 0 \ 1]$ , the system has no solution. In any other case there is at least one solution: a unique solution if there are no free variables, and infinitely many when free variables are present. The pivot variables are usually expressed in terms of the free variables whose values may be arbitrarily (*i.e.*, freely) chosen. Although impractical for large linear systems, Gauss-Jordan reduction is in popular use as a device to solve small systems. And it is easy to devise a program for the calculator to carry out the reduction process.

The following program, GJ.PV (Gauss-Jordan Pivot) pivots on a specified entry to convert the pivot to 1 and produce 0's above and below the pivot. When used in conjunction with program RO.KL, it is reasonably effective in returning the RREF matrix. The program accommodates complex matrices and is actually a simpler version of program ELIM.

**GJ.PV** (Gauss-Jordan Pivot)*Inputs:* level 3: a matrix

level 2: an integer K

level 1: an integer L

*Effect:* converts the (K, L)-entry to 1 and then pivots on that entry to produce 0's above and below the pivot.

```

« → A K L « IF 'A(K, L)' EVAL 0 == THEN "PIVOT ENTRY IS 0"
ELSE A SIZE 1 GET → M « M IDN 'A(1, 1)' EVAL TYPE IF THEN
DUP 0 CON R→C END 1 M FOR I 'A(I, L)' EVAL {I K} SWAP PUT
NEXT INV A * » 8 RND END » »

```

Store this program as variable GJ.PV in your GAUSS subdirectory. On the 28S, replace 8 RND with 8 FIX RND STD.

EXAMPLE 1. Solve the linear system

$$\begin{aligned}
 3x_1 + x_2 + 2x_3 - 2x_4 &= 2 \\
 -3x_1 + x_3 + x_4 &= 0 \\
 -6x_1 - 2x_2 - 4x_3 + x_4 &= 2
 \end{aligned}$$

by applying GJ.PV to the augmented matrix. The result is  $\begin{bmatrix} 1 & 0 & -1/3 & 0 & -2/3 \\ 0 & 1 & 3 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix}$ .

Thus  $x_3$  is a free variable and all solutions are given by  $x = [1/3x_3 - 2/3, -3x_3, x_3, -2]^T$ .

**EXAMPLE 2.** Solve the linear system

$$2x_1 + 3x_2 - x_3 + 4x_4 = 7$$

$$6x_1 - 5x_2 + 3x_3 = 6$$

$$x_1 + 3x_2 + 5x_3 + 7x_4 = 2$$

$$4x_1 + 2x_2 + 6x_3 + 8x_4 = 1$$

by applying GJ.PV to the augmented matrix. You should get

$$\begin{bmatrix} 1 & 0 & 0 & 0 & -7.37499997 \\ 0 & 1 & 0 & 0 & -15.75000012 \\ 0 & 0 & 1 & 0 & -9.50000002 \\ 0 & 0 & 0 & 1 & 14.87500001 \end{bmatrix}. \text{ Use } \boxed{\text{CLEAN}} \text{ to see } x = [-7.375, -15.75, -9.5, 14.875].$$

### EXERCISES 3.3

1. Solve the following linear systems:

(a)  $x_1 + 2x_2 - 3x_3 + 4x_4 = -1$

$$2x_1 + 6x_2 + 10x_3 - 8x_4 = 2$$

$$x_1 + 2x_2 - 2x_3 + 5x_4 = 6$$

$$x_1 + 3x_2 + 5x_3 + 4x_4 = 1$$

(b)  $4x_1 + x_2 + 3x_3 - 2x_4 + x_5 = 5$

$$-8x_1 - 2x_2 - x_3 + 5x_4 = -13$$

$$-4x_1 - x_2 - 8x_3 + x_4 - 2x_5 = -7$$

$$8x_1 + 2x_2 - 9x_3 - 7x_4 - 2x_5 = 9$$



$$(c) \quad x_1 + 2x_2 + 2x_3 + x_4 + 3x_5 = 0$$

$$3x_1 + 6x_2 + 6x_3 + 7x_4 - 2x_5 = 5$$

$$x_1 + 2x_2 + x_3 + 3x_4 - x_5 = 2$$

$$x_1 + 2x_2 + x_3 + 2x_4 - 2x_5 = 3$$

$$-x_1 - 2x_2 - 2x_3 - x_5 = 0$$

$$(d) \quad x_1 - 2x_2 + x_3 + x_4 + 5x_5 + 13x_6 - 9x_7 = -36$$

$$2x_1 - 4x_2 + 3x_3 + 4x_4 + 15x_5 + 41x_6 - 25x_7 = -102$$

$$-3x_1 + 6x_2 + 4x_4 + x_5 + 12x_6 + 7x_7 = 25$$

$$4x_1 - 8x_2 - 3x_4 + 2x_5 - 9x_7 = -26$$

$$-5x_1 + 10x_2 + 7x_4 + x_5 + 20x_6 + 15x_7 = 59$$

$$x_1 - 2x_2 + 2x_3 + 6x_4 + 15x_5 + 50x_6 - 15x_7 = -51$$

2. Solve the following linear systems; check your answers.

$$(a) \quad .235x_1 + 3.273x_2 + 1.564x_3 = 3.879$$

$$2.144x_1 + 5.029x_2 - 9.328x_3 = 7.790$$

$$8.224x_1 - 3.568x_2 + 2.806x_3 = 2.893$$

$$(b) \quad (1+4i)x_1 + (-1+2i)x_2 + (4+7i)x_3 = 2$$

$$5ix_1 + (5+3i)x_2 + (-9+8i)x_3 = -1+6i$$

$$3x_1 + (-7+9i)x_2 + (1+2i)x_3 = 8i$$

3. Solve the following linear systems

$$(a) \quad 2x_1 + (1+i)x_2 + (3+4i)x_3 = 2$$

$$(1-i)x_1 + 4x_2 + (2-3i)x_3 = -1+i$$

$$(3-4i)x_1 + (2+3i)x_2 + 5x_3 = 1-i$$

$$(b) \quad 2.6x_1 - 3.9x_2 + 5.2x_3 + 1.3x_4 = 2.6$$

$$-2.2x_1 + 3.8x_2 - 3.4x_3 - .6x_4 = -4.2$$

$$3.4x_1 - 5.6x_2 + 5.8x_3 + 1.2x_4 = 5.4$$

4. (From Gareth Williams, Stetson University, FL) Use the RAN.Z key to generate six  $3 \times 3$  matrices. In each case, use GJ.PV to get the RREF.

- (a) What do you observe about the RREF ?
- (b) Why does the answer turn out this way? (Hint: think geometrically)

### 3.4 OTHER VARIANTS

There are many variants of Gaussian elimination and its associated natural LU-factorization. When the HP-48S or HP-28S is called upon to calculate  $\det A$  or produce  $A^{-1}$ , it begins by finding a factorization  $A = LU$  or  $PA = LU$  which differs from the natural one in that  $U$  has 1's along its diagonal and the pivots appear on the diagonal of  $L$ . The method used to obtain this factorization is known as the *Crout decomposition* and is especially well-suited to calculator use because it efficiently overwrites  $A$  with  $L$  and the upper triangle of  $U$ :  $L$  fits on and below the diagonal of  $A$  and the upper triangle of  $U$  fits on the corresponding part of  $A$ . This is done in a manner which is very fast and more accurate than in a traditional LU-factorization. The determinant of  $A$  is then calculated as a signed product of the diagonal entries of  $L$ . Since  $A = P^{-1}LU$ , we have  $A^{-1} = U^{-1}L^{-1}P$ . Because  $U$  and  $L$  are triangular, their inverses are easy to calculate and  $A^{-1}$  is found by rearranging the columns in the product  $U^{-1}L^{-1}$  as directed by  $P$ .

Although LU-factorizations are not unique, there is a similar factorization for certain invertible matrices which is unique. Suppose an invertible matrix  $A$  can be brought to upper triangular form  $U$  without row interchanges. Then  $A = LU$  where  $L$  is lower triangular with 1's on its diagonal and  $U$  is upper triangular with non-zero diagonal entries  $u_{11}, u_{22}, \dots, u_{nn}$ . If  $D$  is the diagonal matrix  $D = \text{diag} [u_{11} \ u_{22} \ \dots \ u_{nn}]$  then  $D^{-1} = \text{diag} [u_{11}^{-1} \ u_{22}^{-1} \ \dots \ u_{nn}^{-1}]$  and  $A = LDD^{-1}U = LDU_1$ , where the upper triangular

matrix  $U_1 = D^{-1}U$  also has 1's on its diagonal. This is the **LDU-factorization** of  $A$  and it can be shown to be unique.

The LDU-factorization is especially nice for symmetric matrices  $A$ . For then, in addition to  $A = LDU_1$ , we also have  $A = A^T = (LDU_1)^T = U_1^T D^T L^T = U_1^T D L^T$  so the uniqueness tells us that  $L^T = U_1$ . Thus, *the LDU-factorization of a symmetric matrix has the structure  $A = LDL^T$* . Some examples are in the exercises.

### EXERCISES 3.4

1. Find the LDU-factorization of each of the following symmetric matrices.

$$A = \begin{bmatrix} 2 & 4 & -4 \\ 4 & 12 & -20 \\ -4 & -20 & 50 \end{bmatrix}, \quad B = \begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix}$$

$$2. (a) \text{ Find the LDU-factorization of } A = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}.$$

- (b) Calculate  $A^{-1}$  and find its LDU-factorization.

### 3.5 APPLICATIONS TO VECTOR SPACES

Linear systems are an effective tool to help understand some of the basic concepts encountered in a beginning study of vector spaces: linear combinations and spanning sets, dependence and independence, bases and dimension, change of basis. Though these concepts initially may appear to be somewhat foreign to linear systems, exactly the opposite is true: in the historical development of linear algebra

it was from a study of linear systems and their associated matrices that these vector space concepts emerged.

**LINEAR COMBINATIONS AND SPANNING SETS.** Recall that by a *linear combination* of vectors  $v_1, v_2, \dots, v_k$  in a vector space  $V$  (you may regard  $V$  as  $\mathbb{R}^n$  if that helps) we mean any vector of the form  $x_1v_1 + x_2v_2 + \dots + x_kv_k$  where the  $x_i$ 's are scalars (*i.e.*, numbers). The set of all possible linear combinations of  $v_1, v_2, \dots, v_k$  is a subspace of  $V$ , often denoted by  $\text{Span} [v_1, v_2, \dots, v_k]$ , and the vectors  $v_i$  are said to *span* this subspace. To determine whether a given vector  $u$  lies in  $\text{Span} [v_1, v_2, \dots, v_k]$  we must determine whether  $u = x_1v_1 + x_2v_2 + \dots + x_kv_k$  for suitable scalars  $x_i$ .

The connection to linear systems comes from the fact that, symbolically, the matrix equation  $Ax = b$  expresses  $b$  as a linear combination of the columns  $A$ :  $b = x_1A_1 + x_2A_2 + \dots + x_nA_n$ , where  $A_j$  is column  $j$  of matrix  $A$  and  $x = [x_1, x_2, \dots, x_n]^T$ . Thus, the columns of matrix  $A$  span  $\text{CS}(A) = \text{Span} [A_1, A_2, \dots, A_n]$ , otherwise known as the *column space* of  $A$ . Vector  $b$  is a linear combination of the column's of  $A$  iff  $Ax = b$  has a solution; and any solution to  $Ax = b$  expresses  $b$  as a linear combination of these columns.

**EXAMPLE 1.** To investigate whether  $u = [3 \ 10 \ -2 \ 18]^T$  is a linear combination of  $v_1 = [1 \ -2 \ 3 \ 0]^T$ ,  $v_2 = [-1 \ 4 \ 2 \ 3]^T$  and  $v_3 = [2 \ 0 \ -1 \ 4]^T$ , we set up the linear system  $Ax = u$  where  $A$  has  $v_1, v_2$  and  $v_3$  as its columns. Program ELIM can be used to determine whether a solution exists, but an even better choice would be to use GJ.PV because it will also give us all solutions. Applying GJ.PV to  $[A \ u]$ , we see that

$$\begin{bmatrix} 1 & -1 & 2 & 3 \\ -2 & 4 & 0 & 10 \\ 3 & 2 & -1 & -2 \\ 0 & 3 & 4 & 18 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

from which we see  $u = -v_1 + 2v_2 + 3v_3$ .

**EXAMPLE 2.** Which of  $u_1 = [0 \ 3 \ -6 \ 3]^T$ ,  $u_2 = [-4 \ 7 \ -4 \ 0]^T$  and  $u_3 = [6 \ -4.5 \ 2 \ 2]^T$  are in the span of  $v_1 = [4 \ -1 \ 0 \ 2]^T$  and  $v_2 = [0 \ 3 \ -2 \ 1]^T$ ? We investigate  $Ax = u_i$  ( $i = 1, 2, 3$ ) with  $A = [v_1 \ v_2]$ . Applying GJ.PV to the triple augmented matrix  $[A | u_1 \ u_2 \ u_3]$  to reduce  $A$  to its RREF we find that

$$\begin{bmatrix} 4 & 0 & 0 & -4 & 6 \\ -1 & 3 & 3 & 7 & -4.5 \\ 0 & -2 & -6 & -4 & 2 \\ 2 & 1 & -3 & 0 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & -1 & 1.5 \\ 0 & 1 & 0 & 2 & -1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Column 3 tells us that  $u_1$  is not in  $\text{Span}[v_1, v_2]$  and columns 4 and 5 show that  $u_2 = -v_1 + 2v_2$ ,  $u_3 = 1.5v_1 - v_2$ .

**DEPENDENCE AND INDEPENDENCE.** When a vector  $u$  is a linear combination of some other vectors  $\{v_j\}$ ,  $u$  *depends* linearly upon the  $v_j$ 's and we say that the entire set of vectors is a "linearly dependent" set. More precisely, a set of vectors  $\{v_1, v_2, \dots, v_k\}$  is called (*linearly*) *dependent* if one of these vectors is a linear combination of the others. To the contrary,  $\{v_1, v_2, \dots, v_k\}$  is called (*linearly*) *independent* if no one of these vectors is a linear combination of the others.

To relate these notions to linear systems, recall that they may be reformulated, equivalently, as follows:

- (a)  $\{v_1, v_2, \dots, v_k\}$  is dependent iff there are scalars  $x_1, x_2, \dots, x_k$ , *not all 0*, such that  $x_1v_1 + x_2v_2 + \dots + x_kv_k = 0_v$ ; thus
- (b)  $\{v_1, v_2, \dots, v_k\}$  is independent iff whenever  $x_1v_1 + x_2v_2 + \dots + x_kv_k = 0_v$  then *all*  $x_i = 0$ .

These are the standard notions of dependence and independence found in most elementary texts, but you should not lose sight of the fact that they are the

mathematically equivalent reformulations of the more intuitive ideas given earlier. In terms of linear systems: if matrix  $A$  has vectors  $v_1, v_2, \dots, v_k$  as its columns then we have

- (a)  $\{v_1, v_2, \dots, v_k\}$  is dependent iff  $Ax = 0$  has a non-0 solution; and
- (b)  $\{v_1, v_2, \dots, v_k\}$  is independent iff  $Ax = 0$  has only the 0 solution.

To put this to use, remember some of the conditions under which  $Ax = 0$  has non-0 solutions:  $Ax = 0$  has non-0 solutions iff

- (i)  $A$  has fewer rows than columns, or
- (ii)  $A$  is row-equivalent to a row echelon matrix having fewer non-0 rows than columns; or
- (iii) When  $A$  is square,  $A$  is singular.

(For in each of these cases, Gaussian elimination shows the existence of free variables ... hence non-0 solutions.)

**EXAMPLE 3.** Investigate the dependence/independence of vectors  $v_1 = [-1 \ 2 \ -1 \ 3]$ ,  $v_2 = [2 \ -1 \ 4 \ 1]$  and  $v_3 = [-4 \ 5 \ -6 \ 5]$  in  $\mathbb{R}^4$ . If dependent, write a general dependency equation.

$$A = \begin{bmatrix} -1 & 2 & -4 \\ 2 & -1 & 5 \\ -1 & 4 & -6 \\ 3 & 1 & 5 \end{bmatrix} \xrightarrow{\text{GJ.PV}} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \text{ so by (ii) we see that } Ax = 0 \text{ has}$$

non-0 solutions and  $\{v_1, v_2, v_3\}$  is dependent. In fact, all solutions are given by  $x = [-2\alpha, \alpha, \alpha]^T$ , where  $\alpha$  is freely chosen. Choosing  $\alpha = 1$  we get the particular solution  $x = [-2 \ 1 \ 1]^T$  which says that  $-2v_1 + v_2 + v_3 = 0$ , an equation which expresses the general dependency among these vectors.

It is almost obvious that the non-0 rows of any row echelon matrix are independent, as are the columns which contain the pivots.

**BASES AND DIMENSION.** Spanning sets which are independent are especially desirable because no one of the spanning vectors depends linearly upon the others. By a *basis* for a subspace  $W$  of a vector space  $V$  (again, you may imagine  $V$  to be  $\mathbb{R}^n$  if it helps) we mean a collection of vectors from  $W$  which

- (i) is independent, and
- (ii) spans  $W$ .

When you choose a basis for  $W$ , you have chosen a well-behaved set of vectors to use in describing, or understanding  $W$ . Basis vectors are well-behaved in the sense that they are independent vectors ... hence no dependency upon one another. They may be used to describe  $W$  because each vector in  $W$  is a linear combination of them. Together, we know that each vector in  $W$  can be written as a linear combination of the basis vectors in only one way. Moreover, for finite-dimensional, non-0 vector spaces, *i.e.*, those non-0 spaces having finite spanning sets, the number of vectors in any basis is invariant: *all bases for  $W$  contain the same total number of vectors*. This is the *dimension* of  $W$ ,  $\dim W$ .

We are interested in three important subspaces associated with an  $m \times n$  matrix  $A$ :

- the *row space*  $RS(A)$ : the subspace of  $\mathbb{R}^n$  [ or  $\mathbb{C}^n$  ] spanned by the rows of  $A$ ;
- the *column space*  $CS(A)$ : the subspace of  $\mathbb{R}^m$  [ or  $\mathbb{C}^m$  ] spanned by the columns of  $A$ ; and
- the *null space* of  $A$ ,  $NS(A)$ : the set of all solutions  $x$  to  $Ax = 0$ .

You should recall how we get bases for each of these subspaces: convert  $A$  to row echelon form  $U$  by row operations; then

- the non-0 rows of  $U$  form a basis for  $RS(A)$ ;
- the columns in  $A$  corresponding to the pivot columns in  $U$  form a basis for  $CS(A)$ ; and
- if  $NS(A) = \{0\}$ , we have no basis. Otherwise, we have free variables and all solutions to  $Ax = 0$  are obtained by choosing values for the free variables. Construct special solutions as follows: assign, in turn, the value 1 to each free variable and the value 0 to the other free variables. These special solutions form a basis for  $NS(A)$ . (It sounds more difficult than it is to do!)

Programs `ELIM` or `GJ.PV` may be used to get bases for  $RS(A)$  and  $CS(A)$ ; but `GJ.PV` should be used to get a basis for  $NS(A)$ .

**EXAMPLE 4.** Find bases for the row space, column space and null space of the following matrix:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 4 & 5 & 5 \\ 3 & 6 & 9 & 2 & -5 \\ 2 & 4 & 6 & 1 & -4 \end{bmatrix}.$$

$$\text{Applying } \boxed{\text{GJ.PV}} \text{ to } A \text{ to get the RREF, we have } A \rightarrow \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & -2 \\ 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = U.$$

Thus, the first three rows of  $U$  are a basis for  $RS(A)$  while columns 1, 2, and 4 of  $A$



form a basis for  $CS(A)$ . Clearly  $x_3$  and  $x_5$  are free variables, and all solutions to  $Ax = 0$  look like

$$x = \begin{bmatrix} -x_3 - x_5 \\ -x_3 + 2x_5 \\ x_3 \\ -2x_5 \\ x_5 \end{bmatrix} = x_3 \begin{bmatrix} -1 \\ -1 \\ 1 \\ 0 \\ 0 \end{bmatrix} + x_5 \begin{bmatrix} -1 \\ 2 \\ 0 \\ -2 \\ 1 \end{bmatrix}.$$

The two vectors on the right-hand side are a basis for  $NS(A)$ . They were obtained from the general solution by factoring out  $x_3$  and  $x_5$ ; but notice that they are the special solutions described earlier when you set  $x_3 = 1$  and  $x_5 = 0$ , then  $x_3 = 0$  and  $x_5 = 1$ .

**EXAMPLE 5.** The basis for the row space of  $A$  obtained in Example 4 consisted of the non-0 rows of  $U$ . Since the rows of  $A$  span  $RS(A)$ , we know they can be cut down to obtain a basis for  $RS(A)$ . Which of the *original* rows form a basis for  $RS(A)$ ?

Convert  $A^T$  to RREF and, as above, choose a basis for  $CS(A^T)$  consisting of columns of  $A^T$ . Since  $CS(A^T) = RS(A)$ , transposing the basis vectors will give a basis for  $RS(A)$  which is chosen from among the original rows of  $A$ . Try it. You should get rows 1, 2, and 3 of matrix  $A$ .

**CHANGE OF BASIS.** The ability to change from one basis to another is of fundamental importance in linear algebra. On a finite-dimensional vector space, each linear operator may be represented in a concrete fashion by a matrix, and the matrix itself depends upon the choice of the basis. Changing to a new basis may well provide us with an easier, or more well-structured, matrix.

Although the particular notation used to discuss change of basis ideas will vary from textbook to textbook, most follow a style somewhat as follows. Let

$B = \{ u_1, u_2, \dots, u_n \}$  be an ordered basis for a finite-dimensional vector space  $W$  (a subspace of  $\mathbb{R}^n$ , if you wish). Any vector  $w$  in  $W$  can be written in exactly one way as a linear combination of the vectors in  $B$ :

$$w = x_1 u_1 + x_2 u_2 + \dots + x_n u_n.$$

The column vector  $[w]_B = [x_1 \ x_2 \ \dots \ x_n]^T$  is called the *coordinate matrix* of  $w$  relative to the  $B$ -basis. In  $\mathbb{R}^n$  (or  $\mathbb{C}^n$ ), finding the coordinate matrix  $[w]_B$  for a given vector  $w$  and basis  $B$  usually entails solving a linear system. But we are primarily interested in how we move from the "old" ordered basis  $B$  to a "new" ordered basis  $B' = \{ v_1, v_2, \dots, v_n \}$ . The theorem describing how to do this is as follows:

*Let  $B = \{ u_1, u_2, \dots, u_n \}$  and  $B' = \{ v_1, v_2, \dots, v_n \}$  be ordered bases for a vector space  $W$ . Write each of the old basis vectors in terms of the new basis  $B'$  and consider the coordinate matrices  $[u_1]_{B'}$ ,  $[u_2]_{B'}$ , ...,  $[u_n]_{B'}$ . If  $P$  is the  $n \times n$  matrix whose  $j^{\text{th}}$  column is  $[u_j]_{B'}$ , then  $P$  is invertible and is the only matrix for which  $P[w]_B = [w]_{B'}$ , for all vectors  $w$  in  $W$ . We call  $P$  the *change-of-basis matrix from the  $B$ -basis to the  $B'$  basis*. (Note that  $P$  depends upon the order of the basis vectors as well as the vectors themselves.)*

#### EXAMPLE 6.

- (a) Find the change of basis matrix  $P$  from the "old" basis  $B$  to the "new" basis  $B'$  given below.
- (b) Use  $P$  to write  $w = [3 \ -2 \ -11 \ 17]^T$  in terms of the new basis.

$$B = \{ [10 \ -3 \ -3 \ 10]^T, [-3 \ 23 \ 10 \ -21]^T, [-3 \ 10 \ 7 \ -13]^T, [10 \ -21 \ -13 \ 30]^T \}.$$

$$B' = \{ [2 \ 1 \ -1 \ 2]^T, [1 \ 3 \ 2 \ -3]^T, [-1 \ 2 \ 1 \ -1]^T, [2 \ -3 \ -1 \ 4]^T \}.$$

**SOLUTION**

- (a) To find
- $P$
- we must write each vector in the
- $B$
- basis in terms of the
- $B'$
- basis.

Thus we consider a quadruple-augmented matrix and use GJ.PV:

$$\left[ \begin{array}{cccc|cccc} 2 & 1 & -1 & 2 & 10 & -3 & -3 & 10 \\ 1 & 3 & 2 & -3 & -3 & 23 & 10 & -21 \\ -1 & 2 & 1 & -1 & -3 & 10 & 7 & -13 \\ 2 & -3 & -1 & 4 & 10 & -21 & -13 & 30 \end{array} \right] \rightarrow \left[ \begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 2 & 1 & -1 & 2 \\ 0 & 1 & 0 & 0 & 1 & 3 & 2 & -3 \\ 0 & 0 & 1 & 0 & -1 & 2 & 1 & -1 \\ 0 & 0 & 0 & 1 & 2 & -3 & -1 & 4 \end{array} \right].$$

$$\text{Thus } P = \begin{bmatrix} 2 & 1 & -1 & 2 \\ 1 & 3 & 2 & -3 \\ -1 & 2 & 1 & -1 \\ 2 & -3 & -1 & 4 \end{bmatrix}.$$

- (b) For
- $w = [3 \ -2 \ -11 \ 17]^T$
- , we must first find
- $[w]_B$
- :

$$\left[ \begin{array}{cccc|c} 10 & -3 & -3 & 10 & 3 \\ -3 & 23 & 10 & -21 & -2 \\ -3 & 10 & 7 & -13 & -11 \\ 10 & -21 & -13 & 30 & 17 \end{array} \right] \rightarrow \left[ \begin{array}{cccc|c} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 \end{array} \right], \text{ so } [w]_B = [-1 \ 2 \ -3 \ 1]^T$$

$$\text{and thus } P[w]_B = [5 \ -4 \ 1 \ -1]^T.$$

**EXERCISES 3.5**

- Which of the vectors  $u_1 = [-4 \ -3 \ -1 \ 7]^T$ ,  $u_2 = [-1 \ 1 \ 3 \ 0]^T$  and  $u_3 = [1 \ 2 \ 6 \ 1]^T$  are linear combinations of  $v_1 = [1 \ -3 \ -3 \ 2]^T$ ,  $v_2 = [-2 \ 7 \ 6 \ -5]^T$ ,  $v_3 = [-3 \ -1 \ 9 \ 4]^T$  and  $v_4 = [2 \ 1 \ -6 \ -3]^T$ ? Write any general dependency relations you find.
- Investigate the dependence/independence of each of the following sets of vectors. If dependent, write a general dependency equation.
  - $v_1 = [1 \ i \ 0]^T$ ,  $v_2 = [0 \ i \ 1+i]^T$ ,  $v_3 = [1-i \ 3i \ 0]^T$

(b)  $v_1 = [3 \ 1 \ 1 \ 0]^T$ ,  $v_2 = [6 \ 3 \ 0 \ 1]^T$ ,  $v_3 = [10 \ -1 \ 4 \ 1]^T$ ,  $v_4 = [7 \ 0 \ 2 \ 1]^T$

(c) The rows of  $\begin{bmatrix} [.72 \ .42 \ .58 \ .83 \ .52] \\ [.60 \ .24 \ .90 \ .21 \ .32] \\ [.29 \ .27 \ .44 \ .37 \ .87] \\ [.12 \ .45 \ .82 \ .04 \ .76] \end{bmatrix}$ .

(d) The columns of  $\begin{bmatrix} 4 & 6+8i & 2+2i \\ 2+2i & 4+6i & 8 \\ 6+8i & 10 & 4+6i \\ 1-i & -2+3i & 2 \end{bmatrix}$ .

3. Find bases for the row space, column space and null space of each of the following matrices.

(a)  $A = \begin{bmatrix} -1 & 2 & -3 & 1 \\ 1 & -1 & 2 & 1 \\ 2 & 1 & 2 & 6 \\ -1 & 3 & -3 & 1 \end{bmatrix}$

(b)  $B = \begin{bmatrix} 2 & 0 & 4 & -2 & 0 & 2 \\ -1 & 5 & 6 & 2 & -1 & 0 \\ 3 & -8 & -6 & -5 & 1 & 2 \\ 4 & 10 & 9 & 5 & -4 & -1 \\ 3 & 5 & 14 & -2 & -1 & 4 \\ 5 & -14 & -9 & -10 & -1 & 5 \end{bmatrix}$

4. Consider the two sets  $B = \{u_1, u_2, u_3\}$  and  $B' = \{v_1, v_2, v_3\}$  in  $\mathbb{R}^4$ , where

$$u_1 = [1 \ 0 \ 2 \ 0]^T, u_2 = [2 \ 0 \ 4 \ -3]^T, u_3 = [1 \ 2 \ 2 \ 1]^T \text{ and}$$

$$v_1 = [2 \ 1 \ 4 \ -1]^T, v_2 = [1 \ 2 \ 2 \ 4]^T, v_3 = [0 \ 2 \ 0 \ 1]^T.$$

- (a) Show that both  $B$  and  $B'$  are independent sets of vectors and that  $\text{Span } B = \text{Span } B'$ .
- (b) Let  $W = \text{Span } B = \text{Span } B'$ . Show that  $w = [1 \ 2 \ 2 \ -2]$  is in  $W$ .
- (c) Find the change-of-basis matrix  $P$  from the  $B$ -basis to the  $B'$  basis for  $W$ . Then use  $P$  to express  $w$  in terms of the  $B'$ -basis.

## CHAPTER 4

### ORTHOAGONALITY

Geometrically, a basis for  $\mathbb{R}^n$  is coordinate system. You can certainly see this in the case where  $n = 2$  or  $3$ . To change from one basis to another amounts to changing to a new coordinate system, and our experience with  $\mathbb{R}^2$  and  $\mathbb{R}^3$  suggests that we naturally prefer *rectangular* coordinate systems, *i.e.*, where the coordinate vectors are perpendicular. But what if we change from one such system to another? What do we know about the matrix which effects this change?

The answer to this question, and others associated with rectangular coordinate changes, is provided by a study in  $\mathbb{R}^n$  of the generalized notions of length, distance and perpendicularity in  $\mathbb{R}^3$ . Ultimately, these notions come to focus on the really important one, *orthogonality*, and every beginning course in linear algebra must devote serious attention to it. Orthogonal subspaces, orthogonal bases and orthogonal projections all play a key role in

- (i) least squares solutions to inconsistent linear systems,
- (ii) least squares fits to data, and
- (iii) the Gram-Schmidt process for building orthonormal bases.

Though the Gram-Schmidt process is now standard fare for elementary linear algebra courses, often missing is its interpretation as a factorization process: when applied to the (independent) columns of a matrix  $A$ , we get  $A = QR$  where  $Q$  and  $R$  are well-behaved matrices ( $Q$  has orthonormal columns and  $R$  is upper triangular). QR-factorizations are important because if anything, they have as much applicability as do the LU-factorizations associated with Gaussian elimination.

## 4.1 BASIC CONCEPTS

The geometry of  $\mathbb{R}^3$  is readily extended to  $\mathbb{R}^n$  and  $\mathbb{C}^n$  by means of the standard inner product. In  $\mathbb{R}^n$  the standard inner product is the *dot product*: for column vectors  $x = [x_1 \ x_2 \ \dots \ x_n]^T$  and  $y = [y_1 \ y_2 \ \dots \ y_n]^T$ ,  $x \bullet y = x^T y = x_1 y_1 + y_2 y_2 + \dots + x_n y_n$ . In  $\mathbb{C}^n$ , where the underlying scalars are the complex numbers, the standard inner product of  $x$  and  $y$  is their *Hermitian product*  $x^* y$  (where  $x^*$  is the conjugate transpose of  $x$ ). On the calculators, the menu key DOT returns the dot product of two vectors (real or complex). To get the Hermitian product you must apply the menu key CONJ to the first vector. On the 48S, DOT is on the MTH VECTOR menu and CONJ is on the MTH PARTS menu. On the 28S, both commands are on the ARRAY menu.

### EXAMPLE 1.

(a) For  $x = [1 \ 2 \ 3]^T$  and  $y = [4 \ 5 \ 6]^T$  DOT returns  $x \bullet y = 32$ .

(b) For  $x = [-1+2i \ 3+4i]^T$  and  $y = [1+i \ 2i]^T$  DOT returns  $x \bullet y = (-11, 7)$ ; by first applying CONJ to  $x$ , DOT returns  $x^* y = (9, 3)$ .

For  $x = [x_1 \ x_2 \ \dots \ x_n]^T$  the menu key ABS returns the Euclidean length (norm)  $\|x\|_2 = \sqrt{|x_1|^2 + |x_2|^2 + \dots + |x_n|^2}$  of  $x$ , which is the usual notion of length in  $\mathbb{R}^n$  or  $\mathbb{C}^n$ . ABS is found on the MTH VECTR and MTH MATR menus of the 48S and on the ARRAY menu of the 28S. When applied to an  $n \times n$  matrix  $A = (a_{ij})$ , ABS returns

the *Frobenius matrix norm*  $\|A\|_F = \left[ \sum_{i,j} |a_{ij}|^2 \right]^{1/2}$ . Two other vector and matrix norms are

provided on the calculators, but we will not use them in this chapter. You will find a brief summary of norms in Appendix 3, which also includes the calculators' approach.

**ORTHOGONALITY.** From now on, we shall restrict our attention to  $\mathbb{R}^n$ . Recall that two vectors  $x, y$  are called *orthogonal* if  $x \bullet y = 0$ . In  $\mathbb{R}^2$  or  $\mathbb{R}^3$  this amounts to saying that  $x$  and  $y$  are perpendicular. A set  $B = \{ v_1, v_2, \dots, v_k \}$  of mutually orthogonal vectors ( $v_i \bullet v_j = 0$  for  $i \neq j$ ) is called an *orthogonal set*, and any such set of non-0 vectors is linearly independent. Thus  $B$  is a basis for the subspace  $W = \text{Span} [ v_1, v_2, \dots, v_k ]$ . An attractive feature of such a basis is the ease with which we can obtain the coordinates of any vector  $w$  in  $W$ :

$$(1) \quad w = \frac{(v_1^T w) v_1}{\|v_1\|^2} + \frac{(v_2^T w) v_2}{\|v_2\|^2} + \dots + \frac{(v_k^T w) v_k}{\|v_k\|^2}$$

Even better is the case where each basis vector has length 1, for then (1) becomes

$$w = (v_1^T w) v_1 + (v_2^T w) v_2 + \dots + (v_k^T w) v_k$$

Vectors of length 1 are called *normal* vectors, and we can "normalize" any vector  $v$  by dividing by its length:

$$\frac{v}{\|v\|} \text{ has length 1.}$$

By normalizing any orthogonal basis for  $W$  we can obtain a basis of orthogonal, normal vectors - an *orthonormal basis* - and it is of fundamental importance that any non-0 subspace  $W$  of  $\mathbb{R}^n$  has such a basis. The proof of this fact is the content of the **Gram-Schmidt process**, which we shall examine later.

Look again at the criterion for the orthogonality of a set  $\{ v_1, v_2, \dots, v_k \}$  in  $\mathbb{R}^n$ :  $v_i \bullet v_j = 0$  for  $i \neq j$ . Since  $v_i \bullet v_j = v_i^T v_j$  is the  $(i,j)$ -entry of the  $k \times k$  matrix  $A^T A$ , where  $A$  is the  $n \times k$  matrix having columns  $v_1, v_2, \dots, v_k$  we see that  $\{ v_1, v_2, \dots, v_k \}$  is orthogonal iff

$$A^T A = \begin{bmatrix} v_1^T v_1 & & & \\ & v_2^T v_2 & & \\ & & \ddots & \\ & & & v_k^T v_k \end{bmatrix},$$

i.e., iff  $A^T A$  is a diagonal matrix. Clearly  $\{v_1, v_2, \dots, v_k\}$  is orthonormal iff  $A^T A = I_k$ , the  $k \times k$  identity matrix.

On the calculators, matrices are entered in row order. Thus, to determine

whether vectors  $v_1, v_2, \dots, v_k$  are orthogonal we build  $A = \begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \\ - & v_k & - \end{bmatrix}$  having the

given vectors as rows, get  $A^T$  and check to see if  $A^T A = \begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \\ - & v_k & - \end{bmatrix}$

$\begin{bmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_k \\ | & | & & | \end{bmatrix}$  is diagonal.



**EXAMPLE 2.** To determine whether  $v_1 = [1 \ 0 \ 1 \ -1]^T$ ,  $v_2 = [4 \ -6 \ 3 \ 7]^T$  and  $v_3 =$

$[-2 \ 3 \ 4 \ 2]^T$  are orthogonal, key in  $\begin{bmatrix} 1 & 0 & 1 & -1 \\ 4 & -6 & 3 & 7 \\ -2 & 3 & 4 & 2 \end{bmatrix}$  and press **ENTER** twice; then

use **TRN** and **\*** to see that  $AA^T = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 110 & 0 \\ 0 & 0 & 33 \end{bmatrix}$ , so the vectors are orthogonal.

More generally, since the  $(i, j)$ -entry in  $A^* B$  is the dot product (row of  $A$ )\*(col  $j$  of  $B$ ), we see that  $A^* B = 0$  iff the rows of  $A$  are orthogonal to the columns of  $B$ .

Finally, we note that the row space, column space and null space of a matrix  $A$  all give rise to orthogonal subspaces. Indeed,

$$x \in \text{NS}(A) \iff Ax = 0$$

$$\iff x \text{ is orthogonal to the rows of } A$$

$$\iff x \text{ is orthogonal to } \text{RS}(A).$$

Since the same is true for  $A^T$ , and the row space of  $A^T$  is the column space of  $A$ , we have the important result:

$$x \in \text{NS}(A^T) \iff x \text{ is orthogonal to } \text{CS}(A).$$

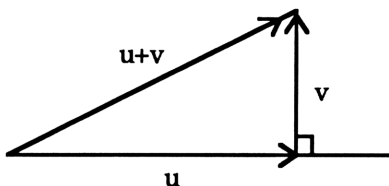
## EXERCISES 4.1

- Let  $\langle x, y \rangle$  denote the standard inner product of vectors  $x, y$  in  $\mathbb{R}^n$  or  $\mathbb{C}^n$ . Use the following pairs of vectors to verify the *Cauchy-Schwartz Inequality*:  $|\langle x, y \rangle| \leq \|x\| \|y\|$ .

(a)  $x = [1 \ -2 \ 3 \ -5]$ ,  $y = [6 \ 9 \ -7 \ 3]$  in  $Z_{10}$ ,

(b)  $x = [1+i \ -2+3i \ i]$ ,  $y = [3-4i \ -i \ 5+i]$  in  $\mathbb{C}^3$ ,

- (c) Any two random vectors of your choosing in  $Z_{10}^4$ .
2. Use the vectors in (a) and (b) of Exercise 1 to verify the triangle inequality:  
 $\|x+y\| \leq \|x\| + \|y\|$
3. The Pythagorean Equality in  $\mathbb{R}^n$  says: for any orthogonal vectors  $u$  and  $v$ ,  
 $\|u+v\|^2 = \|u\|^2 + \|v\|^2$ .



Verify this equality for the following pairs of orthogonal vectors:

- (a)  $u = [-6 \ 4 \ 3 \ 7]^T$  and  $v = [3 \ -2 \ 4 \ 2]^T$
- (b)  $u = [2 \ -1 \ -1 \ 1]^T$  and  $v = [3 \ 2 \ 2 \ -2]^T$
4. Verify that the following sets of vectors are an orthogonal set:
- (a)  $u_1 = [1 \ -1 \ 2 \ -1]$ ,  $u_2 = [1 \ 2 \ 0 \ -1]^T$ ,  $u_3 = [2 \ 0 \ 0 \ 2]$ ,  $u_4 = [-2 \ 2 \ 3 \ 2]$
- (b)  $v_1 = [1/\sqrt{6} \ 1/\sqrt{6} \ 0 \ 2/\sqrt{6}]$ ,  $v_2 = [-1/\sqrt{3} \ -1/\sqrt{3} \ 0 \ 1/\sqrt{3}]$ ,  
 $v_3 = [-1/\sqrt{2} \ 1/\sqrt{2} \ 0 \ 0]$
- (c)  $w_1 = [-1.5 \ .5 \ .5 \ .5]$ ,  $w_2 = [1 \ 1 \ 1 \ 1]$ ,  $w_3 = [0 \ -2 \ 1 \ 1]$
5. For the matrix  $A$  below:
- (a) Find a basis for  $NS(A)$  and verify that the vectors are orthogonal to  $RS(A)$  by checking that they are orthogonal to a basis for  $RS(A)$ .
- (b) Find a basis for  $NS(A^T)$  and verify that the vectors are orthogonal to  $CS(A)$  by checking that they are orthogonal to a basis for  $CS(A)$ .

$$A = \begin{bmatrix} 1 & 2 & 2 & -1 & 2 & 2 \\ 0 & 0 & 1 & -1 & 1 & 0 \\ 1 & 2 & 2 & 0 & 1 & 3 \\ 2 & 4 & 3 & -2 & 4 & 2 \\ 1 & 2 & 1 & -2 & 3 & 2 \end{bmatrix}$$

## 4.2 PROJECTIONS AND LEAST SQUARES

The orthogonal projection  $P_y x$  of a vector  $x$  in  $\mathbb{R}^n$  onto another vector  $y$  is a simple, yet important, idea.

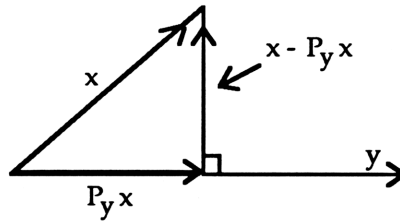


Figure 1.

As Figure 1 suggests, the projection of vector  $x$  onto vector  $y$  is a scalar multiple of  $y$ . In fact,

- (1)  $P_y x = \frac{(x \cdot y) y}{\|y\|^2}$  and
- (2)  $x - P_y x$  is orthogonal to  $y$ .

The following program, PROJ, may be used to calculate the projection vector  $P_y x$  of  $x$  onto  $y$ .

**PROJ** (Projection vector)

*Inputs:* level 2: a vector  $x$

level 1: a vector  $y$

*Effect:* Returns the projection vector  $P_y x$  to level 1

« → X Y « X Y DOT Y \* Y Y DOT / » »

Store variable PROJ in your ORTH subdirectory, next to a copy of CLEAN.

**EXAMPLE 3.** For  $x = [5 \ 15 \ 5]^T$  and  $y = [3 \ 4 \ 5]^T$  find  $P_y x$  and verify that  $x - P_y x$  is orthogonal to  $y$ .

Put two copies vector  $x$  on the stack, followed by two copies of vector  $y$ . The command 4 ROLLD will rearrange the stack to

level 4:  $y$

3:  $x$

2:  $x$

1:  $y$

Press **PROJ** to see  $P_y x$ , then **-** to see  $x - P_y x$ , then **DOT** to see  $y \cdot (x - P_y x)$ .

More generally, we may consider the orthogonal projection  $P_W b$  of a vector  $b$  onto a subspace:

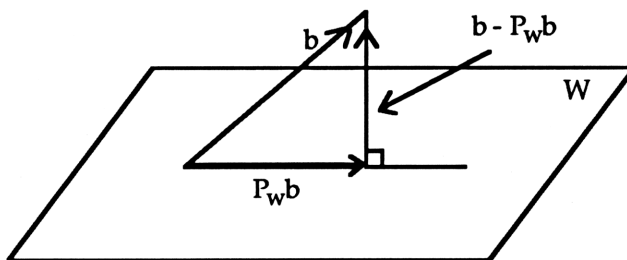


Figure 2.

We shall later define this projection vector  $P_W b$  more precisely, but for now all we need to know is what our geometric intuition tells us: that *vector  $b - P_W b$  is orthogonal to each vector in  $W$ .*

There is a subtle but important connection between orthogonal projections and orthogonal subspaces which is frequently exploited when attempting to solve *overdetermined* linear systems, systems with more equations than unknowns. Intuitively, with more equations than unknowns we are asking too much of the unknowns, and might therefore expect that no solution exists.

More precisely, given an overdetermined system, say  $Ax = b$  where  $A$  is  $m \times n$  and  $m > n$ , we have  $\text{rank } A \leq n < m$ . Thus  $\dim \text{CS}(A) = \text{rank } A < m$ , so  $\text{CS}(A)$  cannot be all of  $\mathbb{R}^m$ . Consequently, there will be vectors  $b$  in  $\mathbb{R}^m$  for which  $Ax = b$  has no solution. However, we may be willing to settle for the next best thing: find a vector  $x^*$  in  $\mathbb{R}^n$  which is "as close as possible" to being a solution to  $Ax = b$ ; that is, a vector  $x^*$  for which the distance  $\|Ax^* - b\|$  from  $Ax^*$  to  $b$  is minimal. Such an  $x^*$  is called a *least squares solution* to  $Ax = b$  because minimizing  $\|Ax^* - b\|$  is equivalent to minimizing  $\|Ax^* - b\|^2$ , which is a sum-of-squares.

We thus seek  $x^*$  so that vector  $Ax^*$ , which lies in the column space  $W$  of  $A$ , is closest to  $b$ . Looking back at Figure 2 we see that  $Ax^*$  must be the projection of vector  $b$  onto the column space  $W$ , in which case  $b - Ax^*$  is orthogonal to each vector in

$CS(A)$ . Remembering that the vectors which are orthogonal to  $CS(A)$  are precisely the vectors in  $NS(A^T)$ , we are practically forced into  $A^T[b - Ax^*] = 0$ , or equivalently

$$A^T A x^* = A^T b.$$

The linear system appearing above is referred to as the system of *normal equations*; thus, vector  $x^*$  is a least squares solution to  $Ax = b$  iff it is a solution to the system of normal equations. In general, the normal equations will have more than one solution. But in the special case that  $A$  has maximal rank, *i.e.*,  $\text{rank } A = n$ , we know that  $A^T A$  is invertible, so  $A^T A x^* = A^T b$  has a *unique* solution  $x^*$ .

**EXAMPLE 4.** Given the overdetermined system  $Ax = b$ :

$$\begin{aligned} 2x_1 - x_2 + x_3 &= 0 \\ 2x_1 + 3x_2 - x_3 &= 1 \\ 3x_1 - 3x_2 + 3x_3 &= 8 \\ 3x_1 + x_2 + x_3 &= 6 \end{aligned}$$

Put two copies of the augmented matrix  $[A|b]$  on the stack and then use **ELIM** to verify that the system has no solution, but  $\text{rank } A = 3$ . Drop the result from level 1, use **SPLIT** to split-off  $b$  and then swap with  $A$ . Use **ENTER** to put two more copies of  $A$  on the stack. Use **TRN** to get  $A^T$ , swap levels with  $A$  and calculate  $A^T A$ . Now swap with  $A$  and calculate  $A^T$ . Use **ROT** to put  $b = [0 \ 1 \ 8 \ 6]^T$  on level 1 and  $A^T$  on level 2, then calculate  $A^T b = [44 \ -15 \ 29]^T$ . Build the augmented matrix  $[A^T A | A^T b]$  with **AD.CC** and solve the normal system with **GJ.PV** to get  $x^* = [-1.6666667 \ 3.8333334 \ 7.91666686]$ . You may your result check by calculating  $x^* = (A^T A)^{-1} A^T b$ .

## FITTING CURVES TO DATA

We shall now see how least squares solutions arise in curve-fitting problems. Suppose we have  $n$  data points  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$  where all the  $x_i$ 's are distinct. Consider the problem of finding a polynomial  $P(x) = c_0 + c_1x + \dots + c_mx^m$  of degree  $\leq m$  which passes through these data points, *i.e. fits* the data. We shall require  $n \geq m+1$ . Thus our requirements are  $P(x_i) = y_i$  for  $i = 1, \dots, n$  or

$$\begin{aligned} c_0 + c_1x_1 + \dots + c_mx_1^m &= y_1 \\ c_0 + c_1x_2 + \dots + c_mx_2^m &= y_2 \\ &\vdots \\ c_0 + c_1x_n + \dots + c_mx_n^m &= y_n \end{aligned}$$

This linear system has  $n$  equations and  $(m+1)$ -unknowns (the coefficients of  $P(x)$ ).

In terms of matrices, the system is  $Ac = y$ , where

$$(*) \quad A = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ & & \vdots & & \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix}, \text{ and } c = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_m \end{bmatrix} \text{ and } y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

Since we require  $n \geq m+1$ , there are at least as many equations as unknowns, so the system will, in general, be overdetermined and we naturally seek a least squares solution. However,  $A$  has independent columns, for if  $Ac = 0$  had a non-0 solution, this would mean that there exists a non-0 polynomial  $P(x)$  of degree  $\leq m$  having  $m+1$  roots! Since  $A$  has independent columns, we know there is a *unique least squares solution*, given as the unique solution to the normal equations

$$(A^T A)c = A^T y.$$

Notice what happens in case A is square ( $n = m+1$ ): A is square with independent columns, so A is invertible and  $c = A^{-1}y$  is the unique polynomial of degree  $\leq n-1$  passing through the  $n$  data points. In this case we call  $P(x)$  the interpolating polynomial for this data, and the square matrix

$$A = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m+1} & x_{m+1}^2 & \dots & x_{m+1}^m \end{bmatrix}$$

is known as a **Vandermonde matrix**.

The following program, P.FIT, may be used to create the coefficient matrix A for fitting a polynomial of degree  $\leq m$  to  $n$  data points  $(x_i, y_i)$ ,  $i = 1, \dots, n$ . When  $n = m+1$ , A will be a Vandermonde matrix.

**P.FIT** (Polynomial Fit Matrix)

**Input:** level 2: an integer M  
level 1: a list  $\{x_1, x_2, \dots, x_N\}$

**Effect:** Returns the matrix

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^M \\ 1 & x_2 & x_2^2 & \dots & x_2^M \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \dots & x_N^M \end{bmatrix}$$

```
« DUP SIZE → M lst N « 1 N FOR J lst J GET → x « 1 1 M
FOR I x I ^ NEXT » NEXT N M 1 + 2 →LIST →ARRY » »
```



Store variable P.FIT in your ORTH subdirectory, next to PROJ.

**EXAMPLE 5.** For the data given below:

- (a) Find the least squares cubic polynomial which fits the data.
- (b) Find the interpolating polynomial for the data.

x	1	2	3	4	5
y	.6	1.2	2	2.8	4.1

Key in the number 3, then the list { 1 2 3 4 5 } of the x-coordinates of the data and press **P.FIT** to see

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 4 & 16 & 64 \\ 1 & 5 & 25 & 125 \end{bmatrix}.$$

Put two additional copies of A on the stack. Get  $A^T$  with **TRN**, swap with A and get  $A^T A$  with **\***. Swap with A and get  $A^T$ . Put  $y = [.6 \ 1.2 \ 2 \ 2.8 \ 4.1]$  on the stack and press **\*** to get  $A^T y = [10.7 \ 40.7 \ 170.7 \ 755.9]$ . Use 5 **AD.CO** to build the augmented matrix  $A^T A | A^T y$ . Use **GJ.PV** and **SPLIT** to see  $c = [-.16 \ .85 \ -.125 \ .025]$ . Thus the least squares cubic polynomial fit is  $P_3(x) = -.16 + .85t - .125t^2 + .025t^3$ .

For part (b), key in the number 4, the list { 1 2 3 4 5 } of x-coordinates of the data and press **P.FIT** to see

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \\ 1 & 5 & 25 & 125 & 625 \end{bmatrix}.$$

Since  $A$  is known to be invertible, we may obtain the solution  $c = A^{-1}y$  by applying  $\boxed{+}$  to  $A$  and  $y = [.6 \quad 1.2 \quad 2 \quad 2.8 \quad 4.1]^T$ . Enter  $y$  and press  $\boxed{\text{SWAP}} \boxed{+} 3 \boxed{\text{CLEAN}}$  to show  $c = [1.1 \quad -1.525 \quad 1.321 \quad -.325 \quad .029]$  to 3 decimal places. Thus the interpolating polynomial is approximately

$$P(x) = 1.1 - 1.525t + 1.321t^2 - .325t^3 + .029t^4.$$

Though the above example does not show it, experience has shown that the coefficient matrix  $A^T A$  of the normal system, with  $A$  given by (\*), is often *ill-conditioned*, in the sense that very small errors in the entries of  $A^T A$  may produce very large errors in the solution. Thus, in most real-world problems we seek other ways to solve the normal equations. We will return to this point later.

**COMMENT:** If you wish to plot the data points and then overlay the graph of a least squares polynomial fit, you may use the following program to plot the data.

48S VERSION

**DPLOT** (Data Plot)

*Inputs:* levels 2 and above: the x- and y-coordinates of  $N$  data points  $x_1, y_1, x_2, y_2, \dots, x_N, y_N$

level 1: the number  $N$  of data points

*Effects:* plots the data points  $(x_i, y_i)$

```
« 2 * →LIST → 1st « ERASE DRAX 1 1st SIZE FOR J 1st DUP J
GET SWAP J 1 + GET R→C PIXON 2 STEP GRAPH » »
```

Store DPLOT next to P.FIT in the ORTH subdirectory.

#### 28S VERSION:

```
« 2 * →LIST → 1st « CLLCD DRAX 1 1st SIZE FOR J 1st DUP J
GET SWAP J 1 + GET R→C PIXEL 2 STEP LCD→ 'SCR' STO DGTIZ
» »
```

Since the 28S calculator has no built-in provision for overlaying graphs, you may use the following program OVLAY.

#### FOR 28S ONLY:

**OVLAY**      (Overlay a plot)

*Input:*      the function  $f$  to be graphed

*Effect:*      draws the graph of  $f$  over the plot produced by  
DPLOT

« STEQ SCR →LCD DRAW DGTIZ »

On the 28S, store DPLOT and OVLAY in the ORTH subdirectory next to P.FIT.

**EXAMPLE 6.** To use DPLOT on the data in EXAMPLE 5, begin by entering the  $x$ - and  $y$ -coordinates of the data: 1, .6, 2, 1.2, 3, 2, 4, 2.8, 5, 4.1. Then enter 5 and press **DPLOT**. If you are using the default PPAR, on the 48S you will see only 4 plotted points, and on the 28S only 2. Thus, you need to adjust the vertical scale in order to see all 5 points. Press **ON** to remove the graph, then recall the data from the last stack with **←** **STACK** on the 48S or with **■** **UNDO** on the 28S. Change the  $y$ -range appropriately (on the 48S, load -2, 4.3 into **YRNG** on the PLOT menu; on the 28S press 3 **\*H** on the PLOT menu). Return to your ORTH subdirectory and again execute 5 **DPLOT** to see all 5 points plotted.

To overlay the plot with the least squares cubic fit polynomial

$$P_3(x) = -.16 + .85x - .125x^2 + .025x^3$$

remove the plot with **[ON]**, then enter `'-.16 + .85* X - .125 * X^2 + .025 * X^3'` onto level 1. On the 28S, simply press **[OVLAY]** to overlay the plotted data with the graph of  $P_3(x)$ . On the 48S, go to the PLOTR menu with **[F2]** **[PLOT]**, load  $P_3(x)$  with **[F1]** **[DRAW]**, then press **[DRAW]**. When you're finished, purge from ORTH the variables left there by the plotting: EQ, PPAR (and in the 28S, SCR).

## EXERCISES 4.2

1. (a) Generate a random  $4 \times 3$  matrix over  $Z_{10}$  whose columns will be called  $u$ ,  $v$  and  $w$ .

(b) Find  $P_v u$  and verify that  $u - P_v u$  is orthogonal to  $v$ .

(c) Find  $P_w u$  and verify that  $u - P_w u$  is orthogonal to  $w$ .

2. (a) Generate a random  $4 \times 3$  matrix  $A$  over  $Z_{10}$  and a random vector  $b$  in  $Z_{10}^4$ .

Apply Gaussian elimination to the normal equations  $A^T A x = A^T b$  to obtain a least squares solution to  $Ax = b$ .

- (b) Repeat using a random  $5 \times 3$  matrix  $A$  over  $Z_{10}$  and a random vector  $b$  in  $Z_{10}^4$ .

3. The following program FEVAL is useful for evaluating functions:

<b>FEVAL</b>	(Function Evaluation)
<i>Inputs:</i>	level 2: 'f(x)', where f(x) is the function to be evaluated
	level 1: a value $x_0$
<i>Effect:</i>	Returns $f(x_0)$ to level 2 and f(x) to level 1 (awaiting another $x_0$ )
« 'X' STO DUP EVAL SWAP 'X' PURGE »	

- (a) Use FEVAL to fill -in the following table of values for  $f(x) = (x+2)^2e^{-x}$  (round to 3 decimal places).

x	-2.2	-1	.5	1.5	3
$y = (x+2)^2e^{-x}$					

- (b) Use DPLOT to plot the 5 data points.
- (c) Find the least squares cubic polynomial  $P_3(x)$  fit to this data; overlay your data plot with the graph of  $P_3(x)$ .
- (d) Find the interpolating polynomial  $P_4(x)$  for this data; overlay your data plot with the graph of  $P_4(x)$ .
4. Suppose you wanted to fit a fifth degree polynomial to 20 data points  $(x_i, y_i)$ , where  $x_i = i$ . What is the coefficient matrix of the system of normal equations?

### 4.3 ORTHONORMAL BASES

We have already seen some of the advantages in having an orthonormal basis  $v_1, v_2, \dots, v_k$  for a subspace  $W$  of  $\mathbb{R}^n$ . The basis vectors are mutually orthogonal and

have length 1, so in this respect they are just like the standard basis vectors  $e_1, e_2, \dots, e_k$  for  $\mathbb{R}^k$ , where  $e_j$  is column  $j$  of the identity matrix. More importantly, orthonormal bases are valued for the ease with which they enable us to write any vector  $w$  in  $W$ :

$$w = (v_1^T w)v_1 + (v_2^T w)v_2 + \dots + (v_k^T w)v_k.$$

The coefficients in this equation are just dot products, so can be found by a simple matrix multiplication instead of the more involved process of solving a linear system:

$$\begin{bmatrix} \text{---} & v_1^T & \text{---} \\ \text{---} & v_2^T & \text{---} \\ & \vdots & \\ \text{---} & v_k^T & \text{---} \end{bmatrix} \begin{bmatrix} | \\ w \\ | \end{bmatrix} = \begin{bmatrix} v_1^T w \\ v_2^T w \\ \vdots \\ v_k^T w \end{bmatrix}.$$

Orthonormal bases are also just what we need to determine the projection  $P_W b$  of a vector  $b$  onto subspace  $W$ . If  $v_1, \dots, v_k$  is an orthonormal basis for  $W$  then  $P_W b$  is (uniquely !) given by

$$P_W b = (v_1^T b)v_1 + (v_2^T b)v_2 + \dots + (v_k^T b)v_k.$$

This projection was the key to understanding least squares solutions to a linear system  $Ax = b$ .

## THE GRAM-SCHMIDT PROCESS

The Gram-Schmidt process is a way to build an orthonormal basis  $q_1, q_2, \dots, q_k$  from a given basis  $x_1, x_2, \dots, x_k$  for  $W$ . Here's how it works.

We let  $q_1$  be the *normalized* version of  $x_1$  :  $q_1 = \frac{x_1}{\|x_1\|}$  . Then, inductively, having constructed orthonormal vectors  $q_1, \dots, q_j$ , we construct  $q_{j+1}$  as follows:

$$(*) \quad q_{j+1} = x_{j+1} - (\text{the sum of the projections of } x_{j+1} \text{ onto } q_1, q_2, \dots, q_j), \text{ normalized.}$$

Thus, before normalization,  $q_{j+1} = x_{j+1} - (\text{the projection of } x_{j+1} \text{ onto the subspace spanned by } q_1, \dots, q_j)$ .

Let's look at several steps:

Step 1:  $q_1 = x_1$ , *normalized*

Step 2:  $q_2 = x_2 - \underbrace{(x_2 \cdot q_1) q_1}_{\text{projection of } x_2 \text{ onto } q_1}, \text{ normalized}$

Step 3:  $q_3 = x_3 - \underbrace{(x_3 \cdot q_1) q_1 - (x_3 \cdot q_2) q_2}_{\text{projections of } x_3 \text{ onto } q_1 \text{ and } q_2}, \text{ normalized}$

⋮

etc.

This is the standard Gram-Schmidt process (there are variations). You should recall that, at each stage,  $\text{Span} [x_1, \dots, x_j] = \text{Span} [q_1, \dots, q_j]$ , so when we're done,  $W = \text{Span} [x_1, \dots, x_k] = \text{Span} [q_1, \dots, q_k]$  and we have an orthonormal basis for  $W$ .

Relative to using the calculator to do the calculations, our position is that it is not pedagogically sound for beginning students to use a program which "does it all" and thus hides the underlying step-by-step process. We prefer instead to prepare and

execute a simple program to carry out each step of the construction. Begin with the basis vectors stored as variables X1, X1, ..., XK in user memory.

Step 1: « X1 X1 ABS / ENTER EVAL , Q1 STO (calculates  $q_1$  and stores it as Q1)

Step 2: « X2 X2 Q1 PROJ - DUP ABS / ENTER EVAL , Q2 STO  
(calculates  $q_2$  and stores it as Q2)

Step 3: « X3 X3 Q1 PROJ - X3 Q2 PROJ - DUP ABS / ENTER EVAL  
, Q3 STO (calculates  $q_3$  and stores it as Q3)

·  
·  
·

. . . and so on.

**EXAMPLE 6.** Apply the above construction to the vectors

$$x_1 = [2 \ 1 \ 0]^T$$

$$x_2 = [0 \ 1 \ 1]^T$$

$$x_3 = [2 \ 0 \ 2]^T$$

Though you may not recognize the entries, the Q1, Q2 and Q3 you constructed are actually the calculator's approximations to

$$q_1 = \frac{1}{\sqrt{5}} [2 \ 1 \ 0]^T$$

$$q_2 = \frac{1}{3\sqrt{5}} [-2 \ 4 \ 5]$$

$$q_3 = \frac{1}{6} [2 \ -4 \ 4] \quad .$$

Once you have Q1, Q2, Q3 as stored variables, you should check to see how close they are to being orthonormal by putting Q1, Q2, Q3 on the stack (in that order),



pressing 3 ROW⇒ to create a matrix  $Q = \begin{bmatrix} \text{---} & Q1 & \text{---} \\ \text{---} & Q2 & \text{---} \\ \text{---} & Q3 & \text{---} \end{bmatrix}$ , then ENTER TRN

\* to see  $QQ^T$ . Due to the floating-point calculations you won't see  $I_3$ ; but clean-up with CLEAN and you should see  $I_3$ .

You may, of course, calculate  $Q1$ ,  $Q2$  and  $Q3$  directly on the stack without entering the algorithm each time as a simple program. But, with the programs, you may check your work before execution and thus avoid procedural errors. We have found this to be a useful teaching device.

### EXERCISES 4.3

1. (a) Generate a random  $4 \times 3$  matrix over  $Z_{10}$  whose columns will be called  $x_1$ ,  $x_2$  and  $x_3$ .  
 (b) Construct an orthonormal basis  $\{q_1, q_2\}$  for  $W = \text{Span}[x_1 \ x_2]$ .  
 (c) Find the projection vector  $P_W x_3$  of  $x_3$  onto  $W$ .  
 (d) Verify that  $x_3 - P_W x_3$  is orthogonal to  $W$  by checking that it is orthogonal to both  $x_1$  and  $x_2$ .
2. Repeat exercise 1 with a random  $5 \times 4$  matrix over  $Z_{10}$ ; let  $W = \text{Span}[x_1 \ x_2 \ x_4]$ .

### 4.4 ORTHOGONAL MATRICES AND QR-FACTORIZATIONS

We now return to a question raised at the outset of the chapter. What do we know about the change of basis matrix, call it  $Q$ , from one orthonormal basis  $B$  to another orthonormal basis  $B'$ ? Without going into the details here, it is not hard to see that  $Q^T Q = I$ , or in other words, *the columns of  $Q$  are orthonormal vectors*. Even

more is true: since  $Q$  is square,  $Q^T Q = I$  guarantees that  $Q Q^T = I$ , which says that  $Q$  *also has orthonormal rows and*  $Q^{-1} = Q^T$ .

In general, we shall call a square matrix  $Q$  **orthogonal** if  $Q^T Q = I$ . Any orthogonal  $Q$  has orthonormal columns, orthonormal rows, and  $Q^T = Q^{-1}$ . Two other properties of such matrices are worth noting:

- (i)  $Q$  preserves lengths,  $\|Qx\| = \|x\|$ , all  $x$
- (ii)  $\det Q = \pm 1$ .

If we are called upon to solve a linear system  $Qx = b$  with  $Q$  orthogonal, it is easy:  $x = Q^{-1}b = Q^T b$ . When  $Q$  is not square, but nevertheless has orthonormal columns, we still have  $Q^T Q = I$  which can be used to solve an inconsistent system  $Qx = b$  by solving the normal equations  $Q^T Q x = Q^T b$ . Here,  $x = Q^T b$  just as in the square orthogonal case.

Not only do orthogonal matrices occur when we change from one orthonormal basis to another (as for instance, when we *rotate*  $\mathbb{R}^3$ ), they lie at the very heart of the Gram-Schmidt process. In fact, just as Gaussian elimination on a matrix  $A$  amounts to an LU-factorization  $A = LU$ , the Gram-Schmidt process applied to a matrix  $A$  having independent columns amounts to a **QR-factorization**  $A = QR$  where  $Q$  has orthonormal columns and  $R$  is invertible and upper (or right) triangular.

To see this, look back at the Gram-Schmidt process, and in each step solve for the  $x$ -vector:

Step 1:  $x_1$  is a scalar multiple of  $q_1$ , say  $x_1 = r_{11}q_1$

Step 2:  $x_2$  is a linear combination of  $q_1$  and  $q_2$ , say  $x_2 = r_{12}q_1 + r_{22}q_2$

Step 3:  $x_3$  is a linear combination of  $q_1, q_2$  and  $q_3$ , say  $x_3 = r_{13}q_1 + r_{23}q_2 + r_{33}q_3$

·  
·  
·

Step j:  $x_j$  is a linear combination of  $q_1, q_2, \dots, q_j$ , say  $x_j = r_{1j}q_1 + r_{2j}q_2 + \dots + r_{jj}q_j$ .

Let  $A$  have  $x_1, x_2, \dots, x_n$  as its columns, left-to-right, and let  $Q$  have  $q_1, q_2, \dots, q_n$  as its columns, also left-to-right. Let  $R$  be the right triangular matrix

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & \cdots & r_{1n} \\ & r_{22} & r_{23} & \cdots & \cdot \\ & & r_{33} & \ddots & \cdot \\ & & & \ddots & \cdot \\ & & & & r_{nn} \end{bmatrix}.$$

In terms of the matrices  $A$ ,  $Q$  and  $R$  the above steps show that

$$\text{col 1 of } A = Q(\text{col 1 of } R)$$

$$\text{col 2 of } A = Q(\text{col 2 of } R)$$

$$\text{col 3 of } A = Q(\text{col 3 of } R)$$

·  
·  
·

$$\text{col } j \text{ of } A = Q(\text{col } j \text{ of } R)$$

or

$$A = QR.$$

Moreover, since  $q_1, q_2, \dots, q_n$  is orthonormal, we know that the  $i^{\text{th}}$  coefficient in

$$x_j = r_{1j}q_1 + r_{2j}q_2 + \dots + r_{jj}q_j$$

is  $r_{ij} = q_i \cdot x_j$ . Also,  $r_{11} \neq 0$  because  $r_{11} = \|x_1\|$ ,  $r_{22} \neq 0$  because  $r_{22} = \|x_2 - (x_2 \cdot q_1)q_1\|$ , etc., and so  $R$  is invertible.

Thus, when the Gram-Schmidt process is applied to the columns of an  $m \times n$  matrix  $A$  (whose columns are assumed independent) we get a factorization  $A = QR$ , where  $Q$  is the same size as  $A$  and has orthonormal columns, and  $R$  is the  $n \times n$  (square), invertible, right triangular matrix whose non-0 entries are given by  $r_{ij} = q_i \cdot x_j$ .

### TO OBTAIN $A = QR$ ON THE CALCULATOR

- (1) Start with  $A$  and its columns  $X_1, X_2, \dots, X_N$  in user memory.
- (2) Construct  $Q_1, Q_2, \dots, Q_N$  from  $X_1, X_2, \dots, X_N$  by the Gram-Schmidt process.
- (3) Construct matrix  $Q$ :

$Q_1 \ Q_2 \ \dots \ Q_N \ N$  ROW $\Rightarrow$  TRN ,  $Q$  STO

- (4) Construct matrix  $R$ :

Build  $R$  by putting its entries onto the stack in row order, then use  $\Rightarrow$ ARRY. Remember that in the upper triangle,  $r_{ij} = q_i \cdot x_j$  for  $i \leq j$ , so use QI XJ DOT. Enter 0's for the lower triangle. Store with ,  $R$  STO.

You may verify that  $A = QR$  as follows:

Press A Q R \* to put  $A$  on level 2 and  $Q \cdot R$  on level 1, then use CLEAN as necessary to clean-up  $Q \cdot R$ . Now press SAME. (SAME is located on the first page of the PRG TEST menu on the 48S and on the second page of the PROGRAM TEST menu on the 28S; a 1 indicates  $A = QR$ , and a 0 indicates  $A \neq QR$ . If you forget to clean-up  $QR$ , you probably won't get  $A = QR$ .)

### EXAMPLE 7.

- (1) Continuing with the vectors from Example 6:
- (2) Construct A as

$$A = \begin{bmatrix} 2 & 0 & 2 \\ 1 & 1 & 0 \\ 0 & 1 & 2 \end{bmatrix}.$$

- (3) After constructing Q you should see

$$Q = \begin{bmatrix} .894427191 & -.298142397 & .333333333334 \\ .4472135955 & .596284794 & -.666666666665 \\ 0 & .7453559925 & .666666666665 \end{bmatrix}.$$

- (4) After constructing R you should see

$$R = \begin{bmatrix} 2.2360679775 & .4472135955 & 1.788854382 \\ 0 & 1.3416407865 & .894427191 \\ 0 & 0 & 2 \end{bmatrix}.$$

Verify that  $A = QR$ :

8  returns 1. Now purge R, Q, A, Q3, Q2, Q1, X3, X2, X1 from user memory.

The factorization  $A = QR$  of a matrix A with independent columns can be used to find the least squares solution to  $Ax = b$ . The unique least squares solution is given by the unique solution to the normal equations  $A^T A x = A^T b$ ; but the coefficient matrix  $A^T A$  is often ill-conditioned, especially if it is a large Vandermonde matrix arising from a polynomial fit, so we do not want to calculate  $(A^T A)^{-1}$ . From  $A = QR$  we have  $A^T A = (QR)^T QR = R^T Q^T QR = R^T R$ , so the normal equations read

$$R^T R x = R^T Q^T b.$$

Since  $R$  is invertible, so is  $R^T$  and we may simplify the normal equations to

$$Rx = Q^T b.$$

Since  $R$  is upper-triangular, back substitution may now be used to find  $x$ , the unique least squares solution to  $Ax = b$ .

**EXAMPLE 8.** Apply the QR-factorization to obtain the least squares solution to the inconsistent linear system

$$\begin{aligned} 2x_1 + x_2 &= 1 \\ x_1 - 2x_2 &= 3 \\ 3x_1 - 4x_2 &= 2 \end{aligned}$$

Here,  $A = \begin{bmatrix} 2 & 1 \\ 1 & -2 \\ 3 & -4 \end{bmatrix}$  and has a QR-factorization where

$$Q = \begin{bmatrix} .534522483825 & .829227982898 \\ .267261241913 & -.349148624378 \\ .801783725738 & -.436435780472 \end{bmatrix} \text{ and}$$

$$R = \begin{bmatrix} 3.74165738678 & -3.20713490295 \\ 0 & 3.27326835354 \end{bmatrix}.$$

To solve  $Rx = Q^T b$ , put  $R$  on the stack and build  $Q^T b = [2.93987366104 \ -1.09108945118]^T$ . Then apply BACK to return  $x = [.5 \ -\bar{3}]^T$  as the unique least squares solution.

You should be aware that the Gram-Schmidt process, as we have presented it, is numerically unstable in floating point arithmetic. That is, round-off errors may conspire to produce vectors which are not, numerically, orthogonal. There is,

however, a variation of the Gram-Schmidt process which is more stable; but we have avoided it because it is not so obvious to a beginning student. And although the Gram-Schmidt process does produce a QR-factorization, in practice other methods are used: Householder reflections or Givens rotations. These are orthogonal matrices which can be used very effectively to obtain QR-factorizations. A *Householder matrix* is simply any matrix of the form

$$H = I - 2ww^T$$

where  $w$  is a column vector with  $\|w\|_2 = 1$ . The Gram-Schmidt algorithm generates a matrix  $Q$  of the same size as  $A$  and having independent columns, but Householder matrices produce a factorization  $A = QR$  where  $Q$  is square and orthogonal. Among the columns of  $Q$  will be an orthonormal basis for the column space of  $A$ . QR-factorizations are important in many modern computer codes for solving linear systems and for calculating eigenvalues of matrices.

#### EXERCISES 4.4

1. (a) Generate a random  $4 \times 3$  matrix  $A$  over  $Z_{10}$  and a random vector  $b$  in  $Z_{10}^4$ .  
 (b) Obtain a QR-factorization for  $A$ . Check your results.  
 (c) Use your QR-factorization to get a least squares solution to  $Ax = b$ .
2. Repeat Exercise 1 for a random  $5 \times 3$  matrix  $A$  over  $Z_{10}$  and a random vector  $b$  in  $Z_{10}^4$ .
3. (a) Normalize  $v = [1 \ -2 \ 1 \ -3 \ 1]^T$  to get a unit vector  $w$ .  
 (b) Use  $w$  to build a Householder matrix  $H = I - 2ww^T$ .  
 (c) We have previously noted that  $H$  is orthogonal. Look at  $H$ . What else is obvious?

- (d) In view of your conclusion in (c), without calculating, what will  $H^{-1}$  be?
  - (e) Verify your result in (d) with a calculation.
4. Repeat exercise 3 with a random vector  $v$  in  $Z_{10}^4$ .
  5. Refer to the vectors  $u_1, u_2, u_3, u_4$  in Exercise 4(a) of Section 4.1, where you were asked to show that they are orthogonal.
    - (a) Normalize these vectors to obtain an orthonormal basis  $B' = \{ v_1, v_2, v_3, v_4 \}$  for  $\mathbb{R}^4$ .
    - (b) Find the change-of-basis matrix  $P$  from the standard basis  $B = \{ e_1, e_2, e_3, e_4 \}$  to the  $B'$  basis.
    - (c) Verify that  $P$  is orthogonal.
  6. Let  $W$  be the subspace of  $\mathbb{R}^4$  spanned by the orthogonal vectors  $w_1, w_2, w_3$  in Exercise 4 (c) of section 4.1.
    - (a) Normalize  $w_1, w_2, w_3$  to obtain an orthonormal basis  $w_1^1, w_2^1, w_3^1$  for  $W$ .
    - (b) Show that  $x_1 = [1 \ 0 \ 0 \ 0]$ ,  $x_2 = [0 \ 1 \ 0 \ 0]$ ,  $x_3 = [0 \ 0 \ 1 \ 1]$  is an orthogonal basis for  $W$ ; normalize to an orthonormal basis  $x_1^1, x_2^1, x_3^1$ .
    - (c) Find the change-of-basis matrix  $P$  from the  $x_i^1$  - basis to the  $w_i^1$  -basis and verify that  $P$  is orthogonal.



## CHAPTER 5

### EIGENVALUES AND EIGENVECTORS

We come now to the last of the several major themes which characterize beginning courses in linear algebra: eigenvalues and eigenvectors. Eigenvalue-eigenvector considerations are of paramount importance in many real applications of linear algebra, especially those involving systems of linear differential equations.

Typically, students are called upon to work through a variety of elementary problems as they begin to meet eigenvalue-eigenvector notions to help reinforce their understanding of the concepts. But it has been the author's observations, more often than not, that the hand calculations which are required soon overwhelm all but the most able of students, and many never get to move much beyond the basics. Boggled down in calculations, they do not get a real chance to seriously consider what is often a major objective of the course: the orthogonal diagonalization of a real symmetric matrix.

The HP-48S and HP-28S calculators can help by substantially removing the computational burden associated with hand calculation of characteristic polynomials, eigenvalues and associated eigenvectors, and the construction of an orthogonal diagonalizing matrix  $Q$ . Actually, this chapter is quite short because we have already seen how to use the calculator to solve linear systems (useful for finding eigenvectors) and to construct orthonormal bases (useful in building an orthogonal diagonalizing matrix  $Q$ ). What remains is to see how we might reasonably use them to calculate eigenvalues.

## 5.1 THE CHARACTERISTIC POLYNOMIAL

Given a square,  $n \times n$  matrix  $A$ , any real or complex number  $\lambda$  for which there is a non-0 vector  $x$  such that  $Ax = \lambda x$  is called an *eigenvalue* of  $A$ , and the non-0 vector  $x$  is an associated *eigenvector*. To find such pairs  $(\lambda, x)$  we consider the equation  $Ax = \lambda x$ , which is clearly equivalent to  $(A - \lambda I)x = 0$ .<sup>1</sup> Thus  $\lambda$  is an eigenvalue, and  $x$  an eigenvector, iff  $x$  is a non-0 solution to the homogeneous linear system with coefficient matrix  $A - \lambda I$ . A non-0 solution exists iff  $A - \lambda I$  is singular, which happens precisely when  $\det(A - \lambda I) = 0$ . The left-hand side,  $\det(A - \lambda I)$ , is a polynomial of degree  $n$  in  $\lambda$ , often called the *characteristic polynomial* of matrix  $A$ . Some authors prefer to use  $\det(\lambda I - A)$  instead, but the difference is minor since these two polynomials differ only by a factor of  $(-1)^n$ . What really counts is that the eigenvalues of  $A$  are the roots of either of these polynomials, and for any such root  $\lambda$  the associated eigenvectors are the non-0 solutions to  $(A - \lambda I)x = 0$ .

Though the above is rather elegant from a purely algebraic viewpoint, it can be a computational nightmare in the real world. In the first place, the defining equation for the characteristic polynomial,  $\det(A - \lambda I)$ , is computationally impractical for all but modest sized, or highly-specialized matrices. And secondly, it is no easy task to determine the roots of a polynomial; certainly, most sophomores have trouble finding the roots of even well-behaved cubic polynomials.

Given an  $n \times n$  matrix  $A$ , the following calculator program, CHAR, calculates the coefficients of  $\det(\lambda I - A) = \lambda^n + c_{n-1}\lambda^{n-1} + \dots + c_1\lambda + c_0$ , which is the characteristic polynomial of  $A$ , or  $(-1)^n$  times the characteristic polynomial of  $A$ , depending upon your point of view. The program implements the SOURIAU-FRAME method, which uses traces of the first  $n$  powers of  $A$ . For the details and a proof, see *Matrices and Linear Transformations*, by Charles Cullen, Addison-Wesley, 1966, p.193.

**CHAR** (Characteristic polynomial)*Input:* level 1: an  $n \times n$  matrix  $A$ *Effect:* returns a list  $\{1 \ c_{n-1} \dots \ c_1 \ c_0\}$  of coefficients of  $\det(\lambda I - A) = \lambda^n + c_{n-1}\lambda^{n-1} + \dots + c_1\lambda + c_0$ 

```
« DUP SIZE 1 GET {1} → mtx n poly « mtx 1 n FOR j 0 1 n FOR k
OVER {k k} GET + NEXT j NEG / 'poly' OVER STO+ mtx DUP ROT *
SWAP ROT * + NEXT DROP poly » »
```

For HP-28S: remove the ' marks from 'poly' and replace STO+ with + 'poly' STO

For convenience, we also include a program to calculate the trace of a matrix.

**TRACE** (Trace of a matrix)*Inputs:* level 1: a square matrix*Effect:* returns the trace of the matrix on level 1

```
« DUP SIZE 1 GET → D N « 0 1 N FOR I 'D(I,I)' EVAL
+ NEXT » »
```

Store CHAR and TRACE in your EIGV subdirectory.

**EXAMPLE 1.** Enter  $\begin{bmatrix} 1 & 1 & 2 \\ 2 & 1 & 1 \\ 2 & 3 & 2 \end{bmatrix}$ . Press CHAR and see  $\{1 \ -4 \ -4 \ -5\}$ . Thus  $\det(\lambda I - A) = \lambda^3 - 4\lambda^2 - 4\lambda - 5$ . Retrieve the matrix and press TRACE to see 4 for the trace.

**EXAMPLE 2.** Enter 
$$\begin{bmatrix} 4 & -8 & 2 & 5 \\ 0 & 1 & -6 & -2 \\ -9 & 0 & 7 & 1 \\ 7 & 3 & -8 & 9 \end{bmatrix}$$
 . Press CHAR to see the coefficients list

$\{ 1 \ -21 \ 144 \ -421 \ -4623 \}$ . Thus  $\det(\lambda I - A) = \lambda^4 - 21\lambda^3 + 144\lambda^2 - 421\lambda - 4623$ . Retrieve the matrix and press TRACE to see 21 for the trace.

### EXERCISES 5.1

1. (a) Generate random  $3 \times 3$  matrices  $A$  and  $B$  over  $Z_{10}$  and store them.  
 (b) Compare trace  $A$ , trace  $B$  and trace  $(A+B)$ . What do you observe?  
 (c) Repeat (a) - (b) using random  $4 \times 4$  matrices.  
 (d) Formulate a conjecture on the basis of your observations. Prove your conjecture.
2. (a) Generate a random  $3 \times 4$  matrix  $A$  and a random  $4 \times 3$  matrix  $B$ , both over  $Z_{10}$ .  
 (b) Compare trace  $(AB)$  and trace  $(BA)$ ; what do you observe?  
 (c) Repeat (a) - (b) for random  $3 \times 5$  and  $5 \times 3$  matrices over  $Z_{10}$ .  
 (d) Formulate a conjecture on the basis of your observations. Prove your conjecture.
3. (a) Generate a random  $3 \times 3$  matrix  $A$  over  $Z_{10}$ .  
 (b) Calculate the characteristic polynomials of  $A$  and  $A^T$ . What do you observe?  
 (c) Repeat (a) - (b) for random  $4 \times 4$  and  $5 \times 5$  matrices over  $Z_{10}$ .

- (d) Formulate a conjecture based upon your observations.
- (e) Prove your conjecture.
4. Let  $A_n(1)$  denote the  $n \times n$  matrix of all 1's.
- (a) Find  $\det[\lambda I - A_n(1)]$  for  $n = 2, 3, 4, 5$ .
- (b) For arbitrary  $n$  what will  $\det[\lambda I - A_n(1)]$  be ?
- (c) What are the eigenvalues of  $A_n(1)$ ?
5. Given the polynomial  $p(\lambda) = \lambda^n + c_{n-1}\lambda^{n-1} + \dots + c_1\lambda + c_0$  its companion matrix is

$$C = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -c_0 & -c_1 & -c_2 & \dots & -c_{n-1} \end{bmatrix}.$$

- (a) For each of the following polynomials find the characteristic polynomial  $\det[\lambda I - C]$  of the companion matrix:
- (i)  $p(\lambda) = \lambda^3 + 5\lambda^2 - 3\lambda + 2$
- (ii)  $p(\lambda) = \lambda^4 - 6\lambda^3 + 2\lambda^2 - 5\lambda + 7$
- (iii)  $p(\lambda) = \lambda^5 + 5\lambda^4 + 4\lambda^3 + 3\lambda^2 + 2\lambda + 1$
- (b) What is the characteristic polynomial of the companion matrix for  $p(\lambda) = x^n + c_{n-1}\lambda^{n-1} + \dots + c_1\lambda + c_0$  ?

6. (a) Generate two random  $3 \times 3$  matrices  $A$  and  $B$  over  $Z_{10}$  and calculate the characteristic polynomials of  $AB$  and  $BA$ . What do you observe?  
(b) Repeat (a) for random  $4 \times 4$  and  $5 \times 5$  matrices over  $Z_{10}$ .  
(c) Repeat (a) - (b) for random  $3 \times 3$ ,  $4 \times 4$  and  $5 \times 5$  *complex* matrices over  $Z_{10}$ .  
(d) Formulate a conjecture based upon your observations. Discuss your conjecture and its implications with your instructor.
7. (a) Generate a random  $3 \times 3$  matrix  $A$  over  $Z_{10}$  and put two copies of  $A$  on the stack.  
(b) Find the characteristic polynomial  $p(x)$  of  $A$  and evaluate  $p(A)$ .  
(c) Repeat (a) - (b) using random  $4 \times 4$  and  $5 \times 5$  matrices over  $Z_{10}$ .  
(d) Repeat (a) - (c) using random  $3 \times 3$ ,  $4 \times 4$  and  $5 \times 5$  *complex* matrices over  $Z_{10}$ .  
(e) Formulate a conjecture based upon your observations. Discuss your conjecture with your instructor.

## 5.2 EIGENVALUE CALCULATIONS

Eigenvalue and eigenvector concepts are best learned in the context of examples, and although low order matrices having integer entries are not always typical of the matrices encountered in scientific and engineering applications, they serve us well in the learning process. But even with such matrices, finding the eigenvalues by hand as the roots of the characteristic polynomial is a difficult ... if not impossible ... task, unless the matrices are highly contrived. To avoid such contrivance, we may use polynomial root-finding programs on the calculator, and Hewlett-Packard has provided us with two such programs.

The first, due to William C. Wickes of Hewlett-Packard, is program PROOT. PROOT implements a closed-form technique for finding all roots, real or complex, of a polynomial of degree  $\leq 4$  having real or complex coefficients. You will find PROOT in Appendix 4 of this book. Although there is no closed-form formula for the roots of a general polynomial of degree 5, the calculator's solver can be used to find a real root  $x_0$ , a polynomial divide routine can be used to factor out  $x - x_0$ , and PROOT can then obtain the other four roots. Because of a result due to Perron, this process may be extended to positive matrices of higher order, though it can become unwieldy for  $n > 6$ . We recommend that HP-28S users store this program in the EIGV subdirectory, together with the required subroutines QUD, CUBIC and QUAR in Appendix 4.

For those who have the HP-48S, we have obtained permission from Hewlett-Packard to use a more powerful and robust polynomial root-finder program, also named PROOT, written in machine language and not yet available in the public domain. This PROOT uses advanced numerical methods to produce all roots of an *arbitrary* polynomial. Because the code is protected, you can only obtain this program by calculator-to-calculator infrared transmission, or by computer-to-calculator serial transmission from a diskette. The diskette is available from the publisher, Harcourt Brace Jovanovich, to instructors who intend to adopt this book for classroom use.

Appendix 4 also contains three other programs that are especially helpful for working with the characteristic polynomial of a matrix:

- PSERS (Polynomial Series)
- PVAL (Polynomial Value)
- PDIV (Polynomial Divide)

PSERS and PVAL can be used to get algebraic expressions for a polynomial having specified real or complex number coefficients, and to evaluate such polynomials at a

particular real or complex number. PDIV is used to divide one polynomial by another. You should store them in your POLY subdirectory. Be sure to put a copy of CLEAN into this subdirectory also.

For HP-28S users, the PROOT program for polynomials of degree  $\leq 4$  requires as input a list  $\{ a_n \ a_{n-1} \ \dots \ a_1 \ a_0 \}$  of coefficients of the polynomial  $a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ , with  $n \leq 4$ , and returns the roots to stack levels 1-4. For HP-48S users with access to the more powerful PROOT, the input must be in the form of a vector  $[ a_n \ a_{n-1} \ \dots \ a_1 \ a_0 ]$  and PROOT returns to level 1 a vector whose components are the roots. Thus, to more effectively use this more powerful PROOT for eigenvalue calculations, we recommend using the following auxiliary program  $\lambda$ .VALUES.

**$\lambda$ .VALUES** (Eigenvalues)

*Input:* level 1: a list of coefficients  $\{ a_n \ a_{n-1} \ \dots \ a_1 \ a_0 \}$

*Effect:* returns on levels 1 through n all roots, real or complex, of the polynomial  $a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ .

*Required program:* PROOT (machine-language version)

« OBJ→ →ARRY PROOT OBJ→ DROP »

Store  $\lambda$ .VALUES next to PROOT in your EIGV subdirectory.

EXAMPLE 4. Start with 
$$\begin{bmatrix} 5 & 1 & 4 \\ -6 & 2 & 6 \\ 0 & 0 & -1 \end{bmatrix}$$
 as your matrix A. Make another copy with **ENTER**. Press **CHAR** to see  $\{ 1 \ -6 \ 9 \ 16 \}$ . Thus the characteristic polynomial (up to a  $\pm$  sign) is  $\lambda^3 - 6\lambda^2 + 9\lambda + 16$ . To obtain the eigenvalues press  **$\lambda$ .VAL** on the 48S, or **PROOT** on the 28S. On the 28S you'll notice that one root needs a



little clean-up. The eigenvalues are  $-1$  and  $3.5 \pm i(1.9364916731)$ . In case you don't recognize it,  $1.9364916731$  is your calculator's decimal approximation to  $.5\sqrt{15}$ .

To find the eigenvectors associated with  $\lambda = -1$ , we must solve  $(A - (-1)I)x =$

$(A + I)x = 0$ . Put  $A + I = \begin{bmatrix} 6 & 1 & 4 \\ -6 & 3 & 6 \\ 0 & 0 & 0 \end{bmatrix}$  on level 1 and use  $\boxed{\text{GJ.PV}}$  to get

$\begin{bmatrix} 1 & 0 & .25 \\ 0 & 1 & 2.5 \\ 0 & 0 & 0 \end{bmatrix}$ . Thus, all eigenvectors associated with  $\lambda = -1$  are scalar multiples of  $\begin{bmatrix} -.25 & -.25 & 1 \end{bmatrix}^T$ .

EXAMPLE 5. This time we use  $\begin{bmatrix} -14 & -16 & -26 & -9 \\ 16 & 19 & 28 & 12 \\ -7 & -8 & -11 & -7 \\ 13 & 14 & 24 & 14 \end{bmatrix}$  as our matrix A. Make another copy with  $\boxed{\text{ENTER}}$ .  $\boxed{\text{CHAR}}$  returns  $\{ 1 \ -8 \ 10 \ 48 \ -99 \}$ , so the characteristic polynomial is  $\lambda^4 - 8\lambda^3 + 10\lambda^2 + 48\lambda - 99$ . Press  $\boxed{\lambda.VAL}$  (or  $\boxed{\text{PROOT}}$  on the 28S) to show (after a little clean-up) four real roots:  $\lambda = 3, 3, -2.464101615$  and  $4.464101615$ . In case you don't recognize the last two, they are decimal approximations to  $1 - 2\sqrt{3}$  and  $1 + 2\sqrt{3}$ , respectively. Drop all roots except  $1 + 2\sqrt{3}$ , put this root into real form with  $\boxed{\text{C} \rightarrow \text{R}}$ , and DROP the imaginary part. Do  $\boxed{+/-}$  ( or  $\boxed{\text{CHS}}$  )  $\boxed{4}$   $\boxed{\text{IDN}}$   $\boxed{*}$   $\boxed{+}$  to get  $(A - \lambda I)$ , where  $\lambda \approx 1 + 2\sqrt{3}$ . Now use  $\boxed{\text{GJ.PV}}$  and  $\boxed{\text{CLEAN}}$  to see a 1-dimensional eigenspace spanned by  $\begin{bmatrix} .19403439 & .57664426 & -.83880688 & 1 \end{bmatrix}^T$ .

**EXAMPLE 6.** Enter two copies of the following matrix

$$A = \begin{bmatrix} 7 & 2 & 4 & 6 \\ -6 & -1 & -4 & -4 \\ 4 & 4 & 5 & -2 \\ -16 & -12 & -14 & -3 \end{bmatrix}.$$

This time **CHAR** returns { 1 -8 22 -40 25 }, so  $p(\lambda) = \lambda^4 - 8\lambda^3 + 22\lambda^2 - 40\lambda + 25$ .  **$\lambda.VAL$**  or **PROOT** tells us that the roots are  $\lambda = 1, 5$  and  $1 \pm 2i$ . To find the eigenspace associated with  $\lambda = 1 - 2i$  we proceed as before: with (1, -2) on level 1 and A on level 2, do **+/-** or **CHS**, 4 **IDN** **\*** **+** to see the complex matrix  $[A - (1 - 2i)I]$ . Gauss-Jordan elimination with **GJ.PV** shows that  $[-1 + i \ 1 \ -i \ 1]$  spans the eigenspace.

**EXAMPLE 7.** Make two copies of this 6×6 matrix A:

$$A = \begin{bmatrix} 140 & 40 & -22 & -6 & -14 & 16 \\ -265 & -74 & 43 & 11 & 27 & -34 \\ 422 & 124 & -64 & -19 & -43 & 45 \\ 78 & 28 & -8 & -4 & -8 & -4 \\ -29 & -8 & 5 & 1 & 5 & -4 \\ 3 & 0 & -1 & 0 & 0 & 1 \end{bmatrix}.$$

**CHAR** returns { 1 -4 -9 64 -100 48 0 }. Thus

$$\begin{aligned} p(\lambda) &= \lambda^6 - 4\lambda^5 - 9\lambda^4 + 64\lambda^3 - 100\lambda^2 + 48\lambda \\ &= \lambda(\lambda^5 - 4\lambda^4 - 9\lambda^3 + 64\lambda^2 - 100\lambda + 48). \end{aligned}$$

USING THE 48S:  **$\lambda.VAL$**  returns -4, 3, 2, 2, 1 and 0 as the eigenvalues. Since  $\lambda = 2$  has multiplicity two, let's find the associated eigenspace. Apply **GJ.PV** with partial pivoting to  $(A - 2I)$  and obtain

$$\begin{bmatrix} 1 & 0 & 0 & -.5 & .5 & 0 \\ 0 & 1 & 0 & .75 & -1.25 & 0 \\ 0 & 0 & 1 & -1.5 & 1.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Thus, the eigenspace for  $\lambda = 2$  has dimension two and you may verify that the vectors  $[.5 \ -75 \ 1.5 \ 1 \ 0 \ 0]^T$  and  $[-.5 \ 1.25 \ -1.5 \ 0 \ 1 \ 0]^T$  are a basis.

**USING THE 28S:** Use EDIT to remove the 0 from  $\{1 \ -4 \ -9 \ 64 \ -100 \ 48 \ 0\}$ , press **ENTER** to make a duplicate copy of the edited list, then use X **PSERS** to generate the polynomial expression

$$p(x) = x^5 - 4x^4 - 9x^3 + 64x^2 - 100x + 48.$$

Now graph  $p(x)$  using the default plotting parameters. Inspection of the display shows a root near  $x = 1$ , so move the cursor to  $x=1$ , initialize with **INS**, and use the SOLVER to obtain the zero  $x=1$ . Build the list  $\{1 \ -1\}$  representing  $x-1$  and use **PDIV** to divide  $p(x)$  by  $x-1$ . The quotient list is  $\{1 \ -3 \ -12 \ 52 \ -48\}$  and **PROOT** returns the remaining four eigenvalues  $-4, 3, 2, 2$ . Now proceed according to the instructions given above for using the 48S to find the eigenspace associated with  $\lambda=2$ .

Program PROOT ON THE 28S does not apply to polynomials of degree 5 or higher, but the technique used in Example 7 may be used on any  $6 \times 6$  matrix. To graphically locate a real root of the characteristic polynomial with the 28S may require that you adjust the plotting parameters. You may also use this approach on *positive* matrices of higher order (although the work may become a little unwieldy for sizes beyond  $6 \times 6$ ) because of the (advanced) theorem due to Perron:

*A real, square matrix having only positive entries has a real eigenvalue  $\lambda$  which is a simple (i.e., not repeated) root of the characteristic polynomial, and all other eigenvalues have absolute value  $< |\lambda|$ .*

In view of this, for a positive  $6 \times 6$  matrix, we may graphically locate the dominant real eigenvalue  $\lambda$ , divide the characteristic polynomial by  $(x - \lambda)$  and then graphically obtain a real root  $\lambda_1$  of the quotient polynomial. After dividing the quotient by  $(x - \lambda_1)$  we may use PROOT to obtain the remaining eigenvalues. An example is in the exercises.

It should be clear that by using the calculator to find the eigenvalues as roots of the characteristic polynomial, you will avoid the tedious computations that will be required if you resort to traditional paper-and-pencil methods. Moreover, you will be able to consider a wider variety and size of matrices with the calculator. But, finding characteristic polynomials of matrices with non-integer entries is not usually done, because small errors in the coefficients may induce sizeable errors in the computation of the eigenvalues. Thus our procedure is not generally recommended for the majority of the matrices which arise in most legitimate applications of science and engineering. It really comes down to the question of how accurate you want to be in your calculations; for matrices of order  $\leq 6$ , the above procedures may be satisfactory in most instances.

The question of numerically obtaining the eigenvalues of matrices is *much* more complicated than, say, the numerical solution of linear systems, and any discussion of the appropriate procedures is certainly beyond the scope of this manual. We have, however, touched upon some of the fundamental ideas in the previous chapter: QR-factorizations and Householder matrices. You may refer to any good book which addresses numerical linear algebra for the details. In particular, you may enjoy reading the excellent expository section 7.3 in Gil Strang's *Linear Algebra and its Applications*, 3<sup>rd</sup> Edition, Harcourt, Brace Jovanovich, 1988.

## EXERCISES 5.2

1. Adding a multiple of a row to another row will not change the determinant of a square matrix  $A$ . Will this change the eigenvalues? The characteristic polynomial? Use your calculator to investigate these questions for the matrix

$$A = \begin{bmatrix} 6 & 10 & -28 \\ -2 & -3 & 4 \\ 1 & 2 & -7 \end{bmatrix}.$$

2. For each of the matrices  $A$  in Examples 4-7:
- (a) Calculate the trace of  $A$  ( $\text{tr } A$ ) and  $\det A$ ; record your results.
  - (b) Compare  $\text{tr } A$  and  $\det A$  with  $\det(\lambda I - A)$ ; what do you observe? Express your observation as a conjecture.
  - (c) Compare  $\text{tr } A$  with the sum,  $\sum_i \lambda_i$ , of the eigenvalues of  $A$ ; what do you observe? Express your observation as a conjecture.
  - (d) Compare  $\prod_i \lambda_i$ , the product of the eigenvalues of  $A$ , with  $\det A$ ; what do you observe? Express your results as a conjecture.
  - (e) Discuss your conjectures with your instructor.
3. For each of the following matrices, find:
- (a) the characteristic polynomial
  - (b) all eigenvalues
  - (c) for the eigenvalue  $\lambda$  of maximum absolute value, the associated eigenspace.

$$A = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 3 & 2 & 1 & 4 \\ 2 & 1 & 4 & 3 \\ 1 & 4 & 3 & 2 \end{bmatrix}$$

$$B = \begin{bmatrix} -2 & -8 & -1 & 6 & 5 \\ 1 & 7 & 1 & -2 & -2 \\ -11 & -16 & 0 & 10 & 5 \\ -7 & -8 & -1 & 10 & 4 \\ 7 & 8 & 1 & -6 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 8 & 6 & 8 & 5 & -3 \\ -9 & -8 & -10 & -9 & 1 \\ -3 & -2 & -3 & -2 & 1 \\ 11 & 11 & 12 & 12 & 1 \\ -8 & -9 & -8 & -8 & -1 \end{bmatrix}$$

$$D = \begin{bmatrix} 54 & 22 & -4 & -2 & -12 & -6 \\ -91 & -34 & 9 & 2 & 22 & 8 \\ 170 & 70 & -12 & -7 & -37 & -20 \\ 92 & 47 & 0 & -8 & -15 & -18 \\ -27 & -7 & 5 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

### 5.3 SIMILARITY

Having concentrated on eigenvalues in the last section, we now briefly focus on eigenvectors.

Given that we earlier made a case for having *independent* sets of vectors - sets in which no one vector depends linearly on the others - we may reasonably ask "independence questions" about the eigenvectors of  $A$ . In particular, how many independent eigenvectors may  $A$  have? Certainly no more than  $n$  because eigenvectors lie in  $\mathbb{R}^n$ , which has dimension  $n$ . And the case where  $A$  has  $n$  independent eigenvectors, say  $x_1, x_2, \dots, x_n$ , is especially nice. Then  $P^{-1}AP = D =$

$\begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}$  where  $\lambda_i$  is the eigenvalue associated with  $x_i$  and  $P$  is the matrix having

$x_1, x_2, \dots, x_n$  as its columns:  $P = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \dots & x_n \\ | & | & & | \end{bmatrix}$ .

In fact, the equation  $P^{-1}AP = D$  is equivalent to saying that  $A$  has  $n$  independent eigenvectors. This equation is just a rearrangement of  $AP = PD$  which, when read column-by-column, simply says  $Ax_i = \lambda_i x_i$ . The  $x_i$ 's are independent because they are the columns of the invertible matrix  $P$ . We are thus led to focus on the case where the  $n \times n$  matrix  $A$  has  $n$  independent eigenvectors as the desirable one, and we call any  $n \times n$  matrix  $A$  having fewer than  $n$  independent eigenvectors *defective*.

The equation  $P^{-1}AP = D$  for a non-defective  $A$  has important ramifications. For in general, we call matrices  $A$  and  $B$  *similar* provided  $P^{-1}AP = B$  for some invertible matrix  $P$ . The term "similar" probably derives from the elementary result that *similar matrices have the same characteristic polynomial, hence the same eigenvalues, determinant and trace*. When  $A$  is similar to a diagonal matrix  $D$  we say that  $A$  is a **diagonalizable** matrix, and the above discussion may be summarized as follows:

*A is diagonalizable iff A has  $n$  independent eigenvectors.*

Of fundamental help in determining if  $A$  is diagonalizable is the result that

*eigenvectors associated with distinct eigenvalues are independent.*

Consequently, if  $A$  has  $n$  distinct eigenvalues, then  $A$  has  $n$  independent eigenvectors - one associated with each eigenvalue - so  $A$  is diagonalizable. But it is also possible for  $A$  to be diagonalizable even when it has fewer than  $n$  distinct eigenvalues. There are two keys to understanding how this may happen:

- (1) For any eigenvalue  $\lambda$  of  $A$ ,  $\dim \text{NS}(A - \lambda I)$ , *i.e.*, the dimension of the eigenspace associated with  $\lambda$ , does not exceed the multiplicity of  $\lambda$  as a root of the characteristic polynomial;
- (2) If  $\lambda_1, \lambda_2, \dots, \lambda_k$  are the *distinct* eigenvalues of  $A$  and  $B_1, B_2, \dots, B_k$  are bases for the associated eigenspaces then the union of these bases is an independent set of eigenvectors of  $A$ .

Think about the characteristic polynomial of  $A$  in factored form:

$$\det(\lambda I - A) = (\lambda - \lambda_1)^{m_1} (\lambda - \lambda_2)^{m_2} \dots (\lambda - \lambda_k)^{m_k}$$

where  $\lambda_1, \dots, \lambda_k$  are the *distinct* eigenvalues and  $m_1, \dots, m_k$  are their respective multiplicities. Since  $\det(\lambda I - A)$  is a polynomial of degree  $n$ , we have  $n = m_1 + m_2 + \dots + m_k$ . According to (1), we have  $\dim \text{NS}(A - \lambda_j I) \leq m_j$ , for each  $j = 1, \dots, k$ . Thus, in the case where equality holds for every  $j$ , the bases in (2) will contain exactly  $m_j$  vectors and their union will produce  $n$  independent eigenvectors for  $A$ . But in the case where we have  $\dim \text{NS}(A - \lambda_j I) < m_j$  for even *one*  $j$ , the union of the bases in (2) will fail to produce  $n$  independent eigenvectors and  $A$  will be defective.

*A is defective iff for some eigenvalue  $\lambda$  there are not enough independent eigenvectors associated with  $\lambda$ .*

EXAMPLE 8. Program CHAR returns { 1 6 9 0 0 } for matrix

$$A = \begin{bmatrix} -9 & 2 & -6 & 0 \\ 5 & 1 & 5 & 7 \\ 6 & 0 & 3 & 1 \\ 3 & -2 & 3 & -1 \end{bmatrix}.$$

Thus the characteristic polynomial is  $\lambda^4 + 6\lambda^3 + 9\lambda^2 = \lambda^2(\lambda^2 + 6\lambda + 9) = \lambda^2(\lambda + 3)^2$  and the distinct eigenvalues are 0 and -3, each having multiplicity 2. A quick



application of GJ.PV to  $A - 0I$  and  $A + 3I$  shows that  $\text{NS}(A - 0I)$  and  $\text{NS}(A + 3I)$  each have dimension 1, so  $A$  is (*doubly*) defective.

**EXAMPLE 9.** Consider  $A = \begin{bmatrix} 0 & 3 & -2 & 0 & -4 \\ -4 & 5 & -4 & 0 & -4 \\ 4 & -3 & 6 & 0 & 4 \\ 4 & -3 & 4 & 3 & 5 \\ -6 & 3 & -6 & 0 & -2 \end{bmatrix}$ . Program CHAR returns

$\{ 1 \ -12 \ 55 \ -120 \ 124 \ -48 \}$  for the list of coefficients of the characteristic polynomial. Our earlier procedures show the eigenvalues as  $\lambda = 2, 2, 3, 4$ . Since  $\lambda = 2$  is the only repeated root, to settle the question whether  $A$  is diagonalizable or defective we must determine  $\dim \text{NS}[A - 2I]$ . Program GJ.PV shows two free variables, so  $\dim \text{NS}[A - 2I] = 2$ , the multiplicity of 2 as a root of the characteristic polynomial. Thus  $A$  is diagonalizable. In fact, a basis for the eigenspace associated with  $\lambda = 2$  consists of the vectors  $[-1 \ 0 \ 1 \ 0 \ 0]$  and  $[0 \ 1\bar{3} \ 0 \ -1 \ 1]$ . The eigenspaces associated with  $\lambda = 1, 3$  and  $4$  have  $[1 \ 1 \ -1 \ -1 \ 1]$ ,  $[0 \ 0 \ 0 \ 1 \ 0]$  and  $[-1 \ 0 \ 0 \ 1 \ 1]$  as bases, respectively. Using these basis vectors as the columns of matrix

$$P = \begin{bmatrix} [-1 & 0 & 1 & 0 & -1] \\ [0 & 1\bar{3} & 1 & 0 & 0] \\ [1 & 0 & -1 & 0 & 0] \\ [0 & -1 & -1 & 1 & 1] \\ [0 & 1 & 1 & 0 & 1] \end{bmatrix},$$

you can verify that  $P^{-1}AP = D = \begin{bmatrix} 2 & & & & \\ & 2 & & & \\ & & 1 & & \\ & & & 3 & \\ & & & & 4 \end{bmatrix}$ .

## EXERCISES 5.3

$$1. \quad (a) \quad \text{Use } A = \begin{bmatrix} 7 & 2 & 4 & 6 \\ -6 & -1 & -4 & -4 \\ 4 & 4 & 5 & -2 \\ -16 & -12 & -14 & -3 \end{bmatrix} \text{ and } P = \begin{bmatrix} 0 & -1 & -1 & -1 \\ 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \text{ to find the}$$

similar matrix  $B = P^{-1}AP$ .

(b) Use CHAR to find the characteristic polynomials of  $A$  and of  $B$ .

Verify that they are the same.

(c) Verify that  $A$  and  $B$  have the same trace and determinant.

2. Determine whether the following matrices  $A$  are diagonalizable or defective. For each one that is diagonalizable, find an invertible  $P$  and a diagonal  $D$  for which  $P^{-1}AP = D$ .

$$(a) \quad \begin{bmatrix} 0 & 1 & -2 & 0 \\ -2 & 3 & -4 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & -1 & 0 & -1 \end{bmatrix} \quad (b) \quad \begin{bmatrix} 10 & -5 & 12 & 0 \\ 9 & -4 & 12 & 0 \\ -5 & 3 & -5 & 0 \\ -7 & 2 & -9 & -2 \end{bmatrix}$$

$$(c) \quad \begin{bmatrix} 8 & -3 & 8 & 0 & 5 \\ 9 & -4 & 12 & 0 & 9 \\ -5 & 3 & -5 & 0 & -5 \\ -7 & 2 & -9 & 0 & -5 \\ 2 & -2 & 4 & 0 & 5 \end{bmatrix} \quad (d) \quad \begin{bmatrix} 1 & 0 & 3 & 0 & -3 & 3 \\ -1 & 2 & 2 & -1 & -2 & 3 \\ 0 & 0 & 0 & -2 & 2 & 2 \\ -2 & 2 & 0 & 4 & 0 & 0 \\ -2 & 2 & -3 & -1 & 5 & 2 \\ 0 & 0 & -1 & -1 & 1 & 4 \end{bmatrix}$$

## 5.4 REAL SYMMETRIC MATRICES

We shall conclude with a very brief review of an extremely interesting class of matrices, the real symmetric matrices. Such matrices play an important role in a variety of applications.

The first thing to remember about real symmetric matrices is that their *eigenvalues are real numbers, i.e., no complex eigenvalues occur*. But more importantly, *eigenvectors associated with distinct eigenvalues are orthogonal*. Being orthogonal, of course, is stronger (and better) than simply being independent. Finally, the really big result is that no real symmetric matrix is defective; that is, *every real symmetric matrix is diagonalizable*. This is because for each eigenvalue  $\lambda$ , there are "enough" eigenvectors. And since we are free to choose the basis vectors for each eigenspace in any way we wish, why not choose them to be orthonormal? Putting all this information about the eigenvectors together, we have the truly elegant result:

*for any real symmetric matrix  $A$  there is an orthogonal matrix  $Q$  such that  $Q^{-1}AQ = D$  is diagonal.*

Since  $Q$  is orthogonal,  $Q^{-1} = Q^T$ . The columns of  $Q$  are orthonormal eigenvectors of  $A$  which form a basis for  $\mathbb{R}^n$ . They correspond, in order, to the eigenvalues of  $A$  which form the diagonal of  $D$ .

The steps to follow in constructing such an orthogonal diagonalizing  $Q$  for a real symmetric  $n \times n$  matrix  $A$  should also be clear:

- Step 1: Find the eigenvalues of  $A$ .
- Step 2: For each eigenspace, construct an orthonormal basis (perhaps by using the Gram-Schmidt process).
- Step 3: The union of the orthonormal bases constructed in step 2 will be an orthonormal basis for  $\mathbb{R}^n$ ; use these basis vectors as the columns of  $Q$ .

**EXAMPLE 10.** Given the real symmetric matrix  $A = \begin{bmatrix} 2 & -1 & 0 & 1 \\ -1 & 2 & 0 & -1 \\ 0 & 0 & 2 & 0 \\ 1 & -1 & 0 & 2 \end{bmatrix}$ , CHAR

returns { 1 -8 21 -22 8 } as the coefficients of the characteristic polynomial.

λ.VAL (or PROOT on the 28S) then finds the eigenvalues as  $\lambda = 1, 1, 2, 4$ .

Applying GJ.PV to both  $(A - 2I)$  and  $(A - 4I)$  we obtain  $[0 \ 0 \ 1 \ 0]$  and  $[1 \ -1 \ 0 \ 1]$

as bases for the associated eigenspaces, respectively. Normalize  $[1 \ -1 \ 0 \ 1]$  to get  $[.577350269189 \ -.577350269189 \ 0 \ .577350269189]$ . Now apply GJ.PV to

$(A - I)$  to get the basis  $\{ [1 \ 1 \ 0 \ 0] \ [ -1 \ 0 \ 0 \ 1] \}$ . Gram-Schmidt these to get

$[.707106781188 \ .707106781188 \ 0 \ 0]$  and  $[ -.408248290463 \ .408248290466 \ 0 \ .816496580929]$ . Use ROW⇒ and TRN to build

$$Q = \begin{bmatrix} 0 & .577350269189 & .707106781188 & -.408248290463 \\ 0 & -.577350269189 & .707106781188 & .408248290466 \\ 1 & 0 & 0 & 0 \\ 0 & .577350269189 & 0 & .816496580929 \end{bmatrix}.$$

## EXERCISES 5.4

Find an orthogonal matrix  $Q$  and a diagonal matrix  $D$  such that  $Q^{-1}AQ = D$ .

(a)  $\begin{bmatrix} 4 & -1 & -1 & -1 \\ -1 & 4 & -1 & -1 \\ -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & 4 \end{bmatrix}$

(b)  $\begin{bmatrix} 3 & -1 & 1 & 0 \\ -1 & 3 & 1 & 0 \\ 1 & 1 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$

(c)  $\begin{bmatrix} 14 & 10 & 10 & 12 \\ 10 & 18 & 16 & 10 \\ 10 & 16 & 18 & 10 \\ 12 & 10 & 10 & 14 \end{bmatrix}$

(d)  $\begin{bmatrix} 1 & -1 & 1 & -1 & 1 \\ -1 & 2 & -2 & 2 & -2 \\ 1 & -2 & 3 & -3 & 3 \\ -1 & 2 & -3 & 4 & -4 \\ 1 & -2 & 3 & -4 & 5 \end{bmatrix}$

## CHAPTER 6

### ITERATIVE METHODS

Gaussian elimination can be costly for large linear systems. The number of multiplications/divisions required to solve  $Ax=b$ , where  $A$  is  $n \times n$ , is

$$\frac{n^3 - n}{3} \text{ multiplications/divisions for the elimination phase,}$$

followed by

$n^2$  additional multiplications/divisions for the back-substitution phase

for a total of

$$\frac{n^3 - n}{3} + n^2 = \frac{n^3}{3} + n^2 - \frac{n}{3} \text{ multiplications/divisions.}$$

When  $n$  is large, the  $\frac{n^3}{3}$  term dominates, so that, for example, if  $n$  is doubled the number of multiplications/divisions increases by a factor of 8.

Fortunately, many of the large linear systems which arise in practice have *sparse* coefficient matrices  $A$ , *i.e.*, all but a small fraction of the entries are 0. And there are versions of Gaussian elimination for sparse matrices which capitalize upon the sparseness. Iterative techniques also use sparseness to good advantage and generate a sequence of increasingly better approximations to a solution. By way of introduction to iterative techniques, we briefly discuss two simple approaches: the *Jacobi* and *Gauss-Seidel iterations*.

We also consider the elementary iterative process known as the *power method* for approximating the dominant eigenvalue and an associated eigenvector of certain matrices. Though severely limited in its applicability, the power method sets the stage for a subsequent study of the preferred iterative technique for finding

eigenvalues, the **QR-algorithm**. However, the QR-algorithm is beyond the scope of this course.

All calculator programs in this chapter should be stored in your **ITERA** subdirectory.

## 6.1 THE JACOBI AND GAUSS-SEIDEL METHODS

The Jacobi and Gauss-Seidel methods for solving a linear system  $Ax = b$  are based upon splitting the matrix  $A$  into two factors  $A = M - N$ , and then rewriting  $Ax = b$  as  $Mx = Nx + b$ . Starting with an initial estimate  $x_0$  for  $x$ , we generate a sequence of successive approximations  $\{x_k\}$  where

$$(1) \quad Mx_k = Nx_{k-1} + b \quad (k \geq 1).$$

With suitable choices for  $M$  and  $N$  the sequence  $\{x_k\}$  will converge to a solution  $x$ . In particular,  $M$  should be invertible in order that  $x_k$  be uniquely defined:

$$x_k = M^{-1}(Nx_{k-1} + b) = (M^{-1}N)x_{k-1} + M^{-1}b$$

The matrix  $M^{-1}N$  is called the *iteration matrix* and is the key to convergence.

- The **Jacobi** iteration takes  $M$  to be the diagonal part of  $A$ , so  $N = M - A$  is the negative of the off-diagonal part of  $A$ .
- The **Gauss-Seidel** iteration takes  $M$  to be the lower triangular part of  $A$ , so  $N = M - A$  is the negative of the strictly upper triangular part of  $A$ .

Convergence of  $\{x_k\}$  to  $x$  is usually defined in terms of the vector max norm:

$$\{x_k\} \rightarrow x \quad \text{if} \quad \lim_{k \rightarrow \infty} \|x_k - x\|_{\infty} = 0.$$

This is equivalent to requiring that each component of  $\{x_k\}$  converge to the corresponding component of  $x$ .

More advanced work shows that for arbitrary  $x_0$ ,  $\{x_k\} \rightarrow x$  iff  $|\lambda| < 1$  for each eigenvalue  $\lambda$  of the iteration matrix  $M^{-1}N$ , or equivalently, iff the spectral radius  $\rho(M^{-1}N) = \max \{ |\lambda| \} < 1$ .

To see how the Jacobi and Gauss-Seidel methods differ, write iteration equation (1) in detail (the components of  $x_k$  will be displayed as  $x_k = [x_1^{(k)} \ x_2^{(k)} \ \dots \ x_n^{(k)}]$ ). For the Jacobi iteration, we have

$$\begin{aligned} a_{11}x_1^{(k)} &= -a_{12}x_2^{(k-1)} - a_{13}x_3^{(k-1)} - \dots - a_{1n}x_n^{(k-1)} + b_1 \\ a_{22}x_2^{(k)} &= -a_{21}x_1^{(k-1)} - a_{23}x_3^{(k-1)} - \dots - a_{2n}x_n^{(k-1)} + b_2 \\ &\vdots \\ a_{nn}x_n^{(k)} &= -a_{n1}x_1^{(k-1)} - a_{n2}x_2^{(k-1)} - \dots - a_{nn-1}x_{n-1}^{(k-1)} + b_n \end{aligned}$$

Thus, to obtain the components of  $x_k$  on the left-hand side *we clearly need to have all*  $a_{ii} \neq 0$ . Moreover, it is apparent that with the Jacobi method the components of the vector  $x_{k-1}$  calculated during the  $(k-1)^{\text{st}}$  iteration (on the right-hand side) are used to obtain the components of the  $x_k$  (on the left-hand side) during the  $k^{\text{th}}$  iteration.

When equation (1) is written in detail for the Gauss-Seidel iteration and the diagonal terms are isolated on the left, we see a difference:

$$\begin{aligned} a_{11}x_1^{(k)} &= -a_{12}x_2^{(k-1)} - a_{13}x_3^{(k-1)} - \dots - a_{1n}x_n^{(k-1)} + b_1 \\ a_{22}x_2^{(k)} &= -a_{21}x_1^{(k)} - a_{23}x_3^{(k-1)} - \dots - a_{2n}x_n^{(k-1)} + b_2 \\ &\vdots \\ a_{nn}x_n^{(k)} &= -a_{n1}x_1^{(k-1)} - a_{n2}x_2^{(k)} - \dots - a_{nn-1}x_{n-1}^{(k)} + b_n \end{aligned}$$

That is, when we calculate  $x_1^{(k)}$  from the first equation we immediately use it in the second equation to calculate  $x_2^{(k)}$ , then we use both  $x_1^{(k)}$  and  $x_2^{(k)}$  in the third equation to get  $x_3^{(k)}$ , etc. Thus, in Gauss-Seidel, components from the  $k^{\text{th}}$  iteration are used as soon as they are available to build other components in that iteration. Because of this continual updating of components, the Gauss-Seidel process often (but not

always) converges faster than the Jacobi process. But there are matrices  $A$  for which *only one* of these two processes will converge. Thus, we need them both.

Earlier, we noted that a necessary and sufficient condition for convergence of either iterative method is that the spectral radius of the iteration matrix  $M^{-1}N$  be less than 1:  $\rho(M^{-1}N) < 1$ . Since for any square matrix  $B$ ,  $\rho(B) \leq \|B\|$  for any matrix norm, estimates on  $\rho(M^{-1}N)$  are usually expressed in terms of matrix norms; thus a *sufficient* condition for convergence is that  $\|M^{-1}N\| < 1$ , for any matrix norm. Because the row-sum norm  $\|\bullet\|_\infty$  and the column-sum norm  $\|\bullet\|_1$  are so easy to calculate, they are popularly used. (See Appendix 3 for a discussion of these matrix norms.)

Another popular criterion *sufficient* for the convergence of either process is that the coefficient matrix  $A$  be *strictly diagonally dominant*:

$$|a_{ii}| > \sum_{j \neq i}^n |a_{ij}| \text{ for } i = 1, 2, \dots, n.$$

The basis for this criterion is the following result:

**THEOREM.**

- (a)  $A$  is strictly diagonally dominant iff the Jacobi iteration matrix has row-sum norm  $< 1$ .
- (b) If  $A$  is strictly diagonally dominant then the Gauss-Seidel iteration matrix has row-sum norm  $< 1$ . (The converse is false; see Exercise 3.)

We shall use the row-sum and column-sum norms in two calculator programs to test the iteration matrices for convergence. But you should remember that these tests are only sufficient for convergence ... not necessary. Thus, it is possible for the tests to fail and still have convergence.



Here are several calculator programs which may be used to implement the Jacobi and Gauss-Seidel iterations. They should be stored in your ITERA subdirectory.

- **JTEST** tests the Jacobi iteration matrix to see if its row-sum norm  $\|\bullet\|_\infty$  or column-sum norm  $\|\bullet\|_1$  is less than 1.
- **JACOBI** performs the Jacobi iterative process.
- **D.DOM** tests the coefficient matrix for strict diagonal dominance.
- **STEST** tests the Gauss-Seidel iteration matrix to see if its row-sum norm  $\|\bullet\|_\infty$  or column-sum norm  $\|\bullet\|_1$  is less than 1.
- **SEIDL** performs the Gauss-Seidel iterative process

**JTEST** (Test Jacobi iteration matrix)

*Input:* level 1: an  $n \times n$  matrix A

*Effect:* tests the Jacobi iteration matrix for  $Ax=b$  to see if its row-sum norm or column-sum norm is less than 1. Returns an appropriate message.

```
« → A « A SIZE 1 GET → N « N IDN DUP 1 N FOR I 'A(I,I)'
EVAL { I I } SWAP PUT NEXT A SWAP / - → itmtrx « IF itmtrx RNRM
1 < THEN "RNRM < 1" ELSE IF itmtrx CNRM 1 < THEN "CNRM < 1"
ELSE "RNRM, CNRM ≥ 1" END END » » » »
```

**JACOBI** (Jacobi iteration)

*Inputs:* level 3: an  $n \times n$  matrix  $A$   
 level 2: an  $n$ -vector  $b$   
 level 1: an accuracy level  $\epsilon$  in the form .00005

*Effect:* returns, at timed intervals, the successive terms of the Jacobi iteration for  $Ax=b$  starting with  $x_0 = 0$ ; terminates when the components of two successive terms agree to within  $n$  decimal places, where  $n$  is the number of 0's in the accuracy level which precede the 5; display is set to  $(n+1)FIX$

```
« → A b ∈ « A SIZE 1 GET → N « N IDN 1 N FOR I 'A(I,I)' EVAL
{ I I } SWAP PUT NEXT DUP A - → M K « 0 'ct' STO ∈ XPON NEG
FIX { N } 0 CON 'XN' STO DO XN 'XO' STO K XO * b + M / 'XN'
STO XN CLLCD 3 DISP .5 WAIT 1 'ct' STO+ UNTIL XN XO - RNRM
∈ < ct 50 == OR END XN { ct XO XN } PURGE » » » »
```

**EXAMPLE 1.** Consider the linear system  $Ax=b$  where

$$A = \begin{bmatrix} 6 & 2 & -2 \\ 2 & 6 & -2 \\ -2 & -2 & 10 \end{bmatrix} \text{ and } b = \begin{bmatrix} 2 \\ 10 \\ -3 \end{bmatrix}.$$

$A$  is strictly diagonally dominant, hence invertible, so  $Ax=b$  has a unique solution. With  $A$  on level 1, JTEST returns the message "RNRM<1". To apply Jacobi iteration to determine the solution  $x$  to 3 decimal place accuracy, enter  $A$ ,  $b$  and .0005. Press **JACOB** to see the iterations converge to  $[-.2499 \quad 1.7501 \quad -.0001]^T$  after 15 iterations. **GJ.PV** applied to the augmented matrix shows  $x = [-.25 \quad 1.75 \quad 0]^T$ . Notice that with 3 decimal place accuracy we actually show the 4<sup>th</sup> decimal digit.

**D.DOM** (Diagonal dominance)*Input:* level 1: an  $n \times n$  matrix A*Effect:* tests to see if A is strictly diagonally dominant.

Returns one of the messages "DIAG DOMINANT" or "NOT DIAG DOMINANT".

```

« → A « A SIZE 1 GET → N « 1 N FOR I A I ED.IT C.ROW SWAP
DROP DUP { I } GET ABS 2 * SWAP CNRM IF ≤ THEN MAXR
→NUM 'I' STO ITERA "NOT DIAG DOMINANT" END IF I N == THEN
"DIAG DOMINANT" END NEXT ITERA » » »

```

NOTE: Program D.DOM calls upon program C.ROW which is assumed to be in your ED.IT subdirectory.

**STEST** (Test Gauss-Seidel iteration matrix)*Input:* level 1: an  $n \times n$  matrix A*Effect:* tests the Gauss-Seidel iteration matrix for  $Ax=b$  to see if its row-sum norm or column-sum norm is less than 1. Returns an appropriate message.

```

« → A « A SIZE 1 GET → N « N IDN DUP 1 N FOR I 1 I FOR J
'A(I,J)' EVAL { I J } SWAP PUT NEXT NEXT A SWAP / - → itmtrx «
IF itmtrx RNRM 1 < THEN "RNRM<1" ELSE IF itmtrx CNRM 1 <
THEN "CNRM<1" ELSE "RNRM,CNRM≥1" END END » » » »

```

**SEIDL** (Gauss-Seidel iteration)

*Inputs:* level 3: an  $n \times n$  matrix  $A$   
 level 2: an  $n$  vector  $b$   
 level 1: an accuracy level  $\epsilon$  in the form .00005.

*Effect:* returns, at timed intervals, the successive terms of the Gauss-Seidel iteration for  $Ax=b$  starting with  $x_0 = 0$ ; terminates when the components of two successive terms agree to within  $n$  decimal places, where  $n$  is the number of 0's in the accuracy level which precede the 5; display is set to  $(n+1)$ FIX

```
« → A b ∈ « A SIZE 1 GET → N « N IDN 1 N FOR I 1 I FOR J
'A(I,J)' EVAL { I J } SWAP PUT NEXT NEXT DUP A - → M K « 0
'ct' STO ∈ XPON NEG FIX { N } 0 CON 'XN' STO DO XN 'XO' STO
K XO * b + M / 'XN' STO XN CLLCD 3 DISP .5 WAIT 1 'ct' STO+
UNTIL XN XO - RNRM ∈ < ct 50 == OR END XN { ct XO XN }
PURGE » » » »
```

EXAMPLE 2. Use the linear system of EXAMPLE 1. With  $\epsilon = .0005$ , SEIDL shows the iterations converging to  $[-.2500 \ 1.7500 \ .0000]$  after only 6 iterations. As before, with 3 decimal digit accuracy, we display 4 decimal digits.

**EXERCISES 6.1**

1. Consider the linear system
 
$$\begin{aligned} 4x + 2y + z &= 11 \\ x + 3y - z &= 4 \\ 2x + 2y + 5z &= 21 \end{aligned}$$

whose coefficient matrix is strictly diagonally dominant.

- (a) Solve the system to 4 decimal place accuracy using Jacobi iteration; count the iterations.
- (b) Solve the system to 4 decimal place accuracy using Gauss-Seidel iteration; count the iterations.

2. Consider the linear system

$$\begin{array}{rclcl} -2x & + & 8y & + & 4z & = & 10 \\ 10x & + & 2y & - & 5z & = & 7 \\ -3x & - & 2y & + & 6z & = & 1 \end{array}$$

- Is the coefficient matrix strictly diagonally dominant?
- Apply **JTEST**. What is your conclusion?
- Apply **STEST**. What is your conclusion?
- Rearrange the equations to get an equivalent system with a strictly diagonally dominant coefficient matrix.
- Solve the rearranged system by Gauss-Seidel iteration, accurate to 4 decimal places.

3. (a) Use matrix  $A = \begin{bmatrix} 9 & 2 & -3 \\ -1 & 4 & 2 \\ 2 & -3 & 5 \end{bmatrix}$  to show that the converse of part (b) of the

**Theorem in Section 6.1 is false.**

- (b) Test the Jacobi iteration matrix for  $Ax=b$  for convergence.

4. Consider this tridiagonal system:

$$2x_1 + x_2 = 5$$

$$x_1 + 2x_2 + x_3 = -2$$

$$x_2 + 2x_3 + x_4 = -13$$

$$x_3 + 2x_4 = -10$$

- (a) Apply D.DOM, JTEST and STEST as tests for convergence.

- (b) On the basis of your results in (a), apply an iterative method to solve the system to 4 decimal place accuracy.

5. Solve the following tridiagonal system to 4 decimal place accuracy with an iterative process.

$$4x_1 - x_2 = 2$$

$$-x_1 + 4x_2 - x_3 = 9$$

$$-x_2 + 4x_3 - x_4 = -8$$

$$-x_3 + 4x_4 - x_5 = -7$$

$$-x_4 + 4x_5 = 6$$

6. The following tridiagonal system is diagonally dominant. Test the Gauss-Seidel iteration matrix and then solve the system with the Gauss-Seidel iterative process, accurate to 6 decimal places.

$$2x_1 - x_2 = 5$$

$$-x_1 + 3x_2 - x_3 = -11$$

$$-x_2 - 3x_3 - x_4 = 1$$

$$-x_3 + 3x_4 - x_5 = -3$$

$$-x_4 - 3x_5 - x_6 = 2$$

$$-x_5 + 2x_6 = 5$$

7. Consider the tridiagonal system

$$\begin{array}{rcl}
 x_1 - x_2 & & = -4 \\
 -x_1 + 3x_2 - x_3 & & = 11 \\
 -x_2 - 3x_3 - x_4 & & = -1 \\
 -x_3 + 3x_4 - x_5 & & = 3 \\
 -x_4 - 3x_5 - x_6 & & = -2 \\
 -x_5 + x_6 & & = -3
 \end{array}$$

- Apply all our tests for convergence. What can you conclude?
- Remember, these tests are only sufficient conditions for convergence. Thus, ignore the test results and try for an iterative solution anyway - accurate to 4 decimal places.

## 6.2 THE POWER METHOD

The power method is a simple iterative technique for finding an approximation to the dominant eigenvalue and an associated eigenvector of a matrix  $A$ . By a *dominant* eigenvalue we mean an eigenvalue  $\lambda_1$  of multiplicity  $m$  (so  $\lambda_1 = \lambda_2 = \dots = \lambda_m$ ) satisfying

$$|\lambda_1| > |\lambda_{m+1}| \geq \dots \geq |\lambda_n|$$

where  $\lambda_1, \lambda_2, \dots, \lambda_n$  are all the eigenvalues of  $A$ . Since a matrix may fail to have a dominant eigenvalue, the power method is not a general purpose technique.

However, it is the basis for other iterative methods; in particular, the powerful QR-algorithm.

The power method is based upon the following assumptions about matrix  $A$ :

- $A$  is real and diagonalizable;

(b)  $A$  has a dominant eigenvalue  $\lambda_1$ .

We start with an arbitrary vector  $y_0$  and form the sequence of unit vectors  $\{y_k\}$  as  $y_k = \frac{Ay_{k-1}}{\|Ay_{k-1}\|}$ , for  $k = 1, 2, \dots$ . Here, we are using the usual Euclidean vector norm (or length). Thus  $y_1 = \frac{Ay_0}{\|Ay_0\|}$ ,  $y_2 = \frac{Ay_1}{\|Ay_1\|}$ ,  $y_3 = \frac{Ay_2}{\|Ay_2\|}$ , etc. Under the two assumptions given above and another one which will soon be apparent, the power method asserts that

(i) the sequence  $\{y_k\}$  converges to a unit eigenvector  $v$  associated with  $\lambda_1$ ;

and

(ii) the sequence of numbers  $\{Ay_k \bullet y_k\}$  converges to the dominant eigenvalue  $\lambda_1$ .

To see why, recall that because  $A$  is assumed to be diagonalizable,  $A$  has  $n$  independent eigenvectors, say  $x_1, x_2, \dots, x_n$  where each  $x_j$  is associated with  $\lambda_j$ :  $Ax_j = \lambda_j x_j$ . Since  $\{x_1, \dots, x_n\}$  is a basis for  $\mathbb{R}^n$ ,

$$(1) \quad y_0 = a_1 x_1 + a_2 x_2 + \dots + a_n x_n$$

for some scalars  $a_j$  ( $j = 1, \dots, n$ ). Consider the sequence  $z_0 = y_0$ ,  $z_k = Az_{k-1}$  ( $k = 1, 2, \dots$ ) without normalization:

$$z_1 = Ay_0, z_2 = Az_1 = A^2 y_0, \dots, z_k = Az_{k-1} = A^k y_0.$$

Using (1),  $A^k x_j = \lambda_j^k x_j$  and  $\lambda_1 = \dots = \lambda_m$ , we have

$$\begin{aligned} z_k = A^k y_0 &= a_1 \lambda_1^k x_1 + \dots + a_n \lambda_n^k x_n \\ &= (a_1 \lambda_1^k x_1 + \dots + a_m \lambda_m^k x_m) + (a_{m+1} \lambda_{m+1}^k x_{m+1} + \dots + a_n \lambda_n^k x_n) \\ (2) \quad &= \lambda_1^k \left[ (a_1 x_1 + \dots + a_m x_m) + a_{m+1} \left( \frac{\lambda_{m+1}}{\lambda_1} \right)^k x_{m+1} + \dots + a_n \left( \frac{\lambda_n}{\lambda_1} \right)^k x_n \right]. \end{aligned}$$



$$\text{Now } y_1 = \frac{Ay_0}{\|Ay_0\|} = \frac{z_1}{\|z_1\|}, \quad y_2 = \frac{Ay_1}{\|Ay_1\|} = \frac{A\left(\frac{z_1}{\|z_1\|}\right)}{\left\|A\left(\frac{z_1}{\|z_1\|}\right)\right\|} = \frac{Az_1}{\|Az_1\|} = \frac{z_2}{\|z_2\|},$$

and in general,  $y_k = \frac{z_k}{\|z_k\|}$ . Looking back at (2), we see

$$(3) \quad y_k = \frac{\lambda_1^k \left[ (a_1 x_1 + \dots + a_m x_m) + a_{m+1} \left( \frac{\lambda_{m+1}}{\lambda_1} \right)^k x_{m+1} + \dots + a_n \left( \frac{\lambda_n}{\lambda_1} \right)^k x_n \right]}{\| \lambda_1^k \| \left\| (a_1 x_1 + \dots + a_m x_m) + a_{m+1} \left( \frac{\lambda_{m+1}}{\lambda_1} \right)^k x_{m+1} + \dots + a_n \left( \frac{\lambda_n}{\lambda_1} \right)^k x_n \right\|}.$$

Since  $|\lambda_1| > |\lambda_j|$  for  $j = m+1, \dots, n$ , we have  $\left| \frac{\lambda_j}{\lambda_1} \right| < 1$  for  $j = m+1, \dots, n$ . Thus from (3),

$$\text{as } k \rightarrow \infty, \quad y_k \rightarrow \pm \frac{(a_1 x_1 + \dots + a_m x_m)}{\|a_1 x_1 + \dots + a_m x_m\|} = v,$$

which is a unit eigenvector associated with  $\lambda_1$  in case at least one of  $a_1, \dots, a_m$  is  $\neq 0$ . (This is now our third assumption.)

Finally, since  $y_k \rightarrow v$ ,  $Ay_k \rightarrow Av = \lambda_1 v$ , so that  $Ay_k \cdot y_k \rightarrow \lambda_1 v \cdot v = \lambda_1 (v \cdot v) = \lambda_1 \|v\|^2 = \lambda_1$ . This completes the argument.

In summary, under our three assumptions, the power method gives us two sequences:

- a sequence of vectors  $\{y_k\}$  converging to an eigenvector  $v$  associated with the dominant eigenvalue  $\lambda_1$ ;
- an associated sequence of numbers  $\{Ay_k \cdot y_k\}$  converging to the dominant eigenvalue  $\lambda_1$ .

In practice, we calculate the sequences together, term-by-term; because the convergence of  $\{y_k\}$  to a vector  $v$  amounts to the convergence of the components of  $y_k$

to the corresponding components of  $v$ , we base our stopping criterion on the sequence  $\{y_k\}$ . The sequence of numbers  $\{A y_k \bullet y_k\}$  often converges more rapidly.

To implement the entire process on the calculator requires a program to calculate both sequences, term-by-term, and to apply the desired stopping criterion. Program POWER does just that.

**POWER** (Power method)

*Input:* level 3: a real  $n \times n$  matrix  $A$ , assumed to be diagonalizable and have a dominant eigenvalue  $\lambda$   
 level 2: an  $n$ -vector  $y_0$ , assumed to have a non-0 component in the direction of an eigenvector associated with  $\lambda$   
 level 1: an accuracy level  $\epsilon$  in the form .00005

*Effect:* returns, at timed intervals, the successive terms of a sequence of vectors which approach a dominant eigenvector, and a corresponding sequence of numbers which approach the dominant eigenvalue; terminates when two successive terms of the sequence of vectors agree to within  $n$  decimal places, where  $n$  is the number of 0's in the accuracy level which precede the 5; display is set to  $(n+1)$  FIX.

```
« → A y₀ ∈ « 0 'ct' STO ∈ XPON NEG FIX y₀ 'YN' STO DO YN
'YO' STO A YO * DUP ABS / 'YN' STO A YN * YN DOT CLLCD
1 DISP YN 4 DISP .5 WAIT 1 'ct' STO+ UNTIL YN YO - RNRM ∈ <
ct 60 == OR END A YN * YN DOT YN { ct YO YN } PURGE » »
```

**EXAMPLE 3.** Enter and store the matrix

$$A = \begin{bmatrix} 7 & -33 & -6 & -16 \\ 14 & -72 & -12 & -38 \\ -19 & 105 & 20 & 53 \\ -19 & 96 & 15 & 52 \end{bmatrix}.$$

Using the methods of Chapter 5, you can verify that  $A$  is diagonalizable with eigenvalues 3, 3, 2 and -1. Thus  $\lambda = 3$  is dominant eigenvalue of multiplicity two. Let  $y_0 = [1 \ 1 \ 1 \ 1]^T$  and proceed on the assumption that  $y_0$  has a non-0 component in the direction of an eigenvector associated with  $\lambda = 3$ . Using 4 decimal place accuracy specified by  $\epsilon = .00005$ , **POWER** shows a sequence of numbers converging to  $\tilde{\lambda} = 3.00024$  and a sequence of vectors converging to  $\tilde{x} = [.04091 \ -0.43558 \ .57141 \ .69433]^T$  after 19 iterations. With 4 decimal place accuracy we display 5 decimal digits. To see how close the pair  $(\tilde{\lambda}, \tilde{x})$  is to being an eigenvalue-eigenvector pair for  $A$ : duplicate  $\tilde{x}$  with **ENTER**, recall  $A$  to level 1 with **A** and press **SWAP** **\*** to see  $A\tilde{x} = [.12282 \ -1.30684 \ 1.71432 \ 2.08317]^T$ . Now do 3 **ROLLD** **\*** to see  $\tilde{\lambda}\tilde{x} = [.12273 \ -1.30684 \ 1.71436 \ 2.08314]^T$ . Compare levels 1 and 2.

**COMMENT.** As in this example, we usually do not know in advance whether the initial  $y_0$  has a non-0 component in the direction of a dominant eigenvector. But this rarely causes difficulty in practice because the round off errors which appear after a few iterations usually perturb the problem to the point where this is the case.

**EXAMPLE 4.** To see the effect of different initial vectors  $y_0$ , return to the matrix  $A$  of **EXAMPLE 3** using  $y_0 = [1 \ 0 \ -1 \ 0]^T$ , then with  $y_0 = [1 \ 2 \ 3 \ 4]^T$ . With 4 place accuracy, the first choice shows  $\tilde{\lambda} = 3.00026$  and  $\tilde{x} = [-0.03984 \ .43533 \ -.57388 \ -.69253]^T$  while the second choice shows  $\tilde{\lambda} = 3.00024$  and  $\tilde{x} = [.04456 \ -.43620 \ .56503 \ .69891]^T$ , both after 19 iterations.

Finally, since real symmetric matrices are diagonalizable and have only real eigenvalues, they are natural candidates for the power method.

## EXERCISES 6.2

1. For each of the following matrices  $A$ :

- (a) Find the eigenvalues by finding the roots of the characteristic polynomial.
- (b) Verify that  $A$  has a dominant eigenvalue  $\lambda$ .
- (c) Apply the power method via program POWER, starting with the vector of all 1's and using 5 decimal place accuracy; count the iterations.

$$(i) \quad A = \begin{bmatrix} 6 & 2 & 3 \\ 2 & 5 & 4 \\ 3 & 4 & -1 \end{bmatrix}$$

- (ii) The matrix  $A$  obtained as follows: seed the random number generator with 2 (press 2 RDZ), then go to the BILDR subdirectory and press 4 SYMM.

2. (a) Repeat exercise 1 for the  $5 \times 5$  matrix obtained as follows: seed the random number generator with 3, then press 5 SYMM in the BILDR subdirectory. Be sure to count the iterations and pay *close attention* to the sequence of vectors being generated.
- (b) How many iterations occurred? This is the maximum number allowed by program POWER.

To see why this many iterations occurred, notice that the stopping criteria in POWER is:  $\|y_{k+1} - y_k\|_{\infty} < \epsilon$  or  $ct = 60$  (where  $ct$  is the iteration counter). For this particular matrix, after 17 iterations, the components of  $y_{k+1}$  and  $y_k$  agree to

5 decimal places except for a sign :  $y_{k+1} = -y_k$ . Thus  $y_{k+1} - y_k = 2y_{k+1}$ , so  $\|y_{k+1} - y_k\|_\infty = \|2y_{k+1}\|_\infty = 2\|y_{k+1}\|_\infty$  which ceases to decrease. Thus the iterations continue to the maximum allowable.

3. Looking back at equation (2) in our derivation of the power method, you can see that the rate of convergence is governed by the factor  $\left| \frac{\lambda_{m+1}}{\lambda_1} \right|$  (remember:  $\lambda_1 = \lambda_2 = \dots = \lambda_m$  and  $|\lambda_1| > |\lambda_{m+1}| \geq \dots \geq |\lambda_n|$ ). The smaller this factor, the faster the convergence. That is, the convergence will be faster if  $\lambda_1$  *strongly* dominates the next largest eigenvalue. To see this in practice, consider the following two matrices A and B, which differ only in their (1,1)-entries.

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 1 & -1 & 0 \\ 0 & -1 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 6 & -1 & 0 & 0 \\ -1 & 1 & -1 & 0 \\ 0 & -1 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

For matrix A,  $\left| \frac{\lambda_{m+1}}{\lambda_1} \right| \approx .6946$  whereas for matrix B,  $\left| \frac{\lambda_{m+1}}{\lambda_1} \right| \approx .3796$  ... a little more than one-half the value for matrix A.

- (a) Apply the power method to matrix A using  $y_0 = [1 \ 1 \ 1 \ 1]$  with 4 decimal place accuracy; count the number of iterations required.
- (b) Now apply the power method to matrix B using  $y_0 = [1 \ 1 \ 1 \ 1]$  with 4 decimal place accuracy; count the number of iterations required and compare with the corresponding number in (a).
- (c) In light of this, if the convergence proceeds so slowly (you should watch the sequence of vectors ... not the sequence of  $\lambda_k$ 's) that you reach the maximum allowable iteration count before convergence, you should change the maximum iteration count in program POWER.

4. (a) Our presentation of the power method required a *unique* dominant eigenvalue  $\lambda_1$  (of multiplicity  $m \geq 1$ ). What happens if, say,  $\lambda$  and its negative  $-\lambda$  dominate? That is, if  $|\lambda_1| = |-\lambda_1| > |\lambda_3| \geq \dots \geq |\lambda_n|$ ? To find out, apply the power method to the following matrix  $A$ , using  $y_0 = [1 \ 1 \ 1 \ 1]$  and 5 decimal place accuracy. Watch the vector iterations *very carefully* and count the number of iterations.

$$A = \begin{bmatrix} 3 & -1 & 0 & 0 \\ -1 & -3 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & -3 \end{bmatrix}$$

- (b) Has the sequence of vectors converged to a unique approximate eigenvector before reaching the maximum allowable number of iterations? Repeat the iteration as necessary to see again what is happening.
- (c) Formulate a conjecture based upon your observations for discussion with your classmates and instructor.

## APPENDIX 1

### PROGRAM HOUSEKEEPING

The term user memory refers to that part of the calculator's memory which is accessible to a user through the VAR menu on the 48S and the USER menu on the 28S. User memory is where we store the various types of objects recognized by the calculator, e.g., real or complex numbers, arrays, programs, lists, etc. These objects are stored as *global variables* (in calculator terminology) which you may regard as the name of the object. Here we are concerned with the basic "housekeeping" procedures associated with programs. By "housekeeping", we mean the simple procedures used to enter, name and store, run, edit and purge programs. The Owner's Manuals minimally address programming; but anyone desiring to become really proficient in developing and using programs across a broad spectrum of applications is strongly advised to study the books "HP-48 Insights" and "HP-28 Insights", by William C. Wickes.

**WHAT IS AN HP-48S/28S PROGRAM?** A program is a sequence of data objects, procedures, commands and program structures - the *program body* - enclosed between program delimiters:

« *program body* » .

**ENTERING PROGRAMS.** Programs are keyed into the command line and entered onto the stack (level 1) with ENTER. You need not key in the necessary closing program delimiters because pressing ENTER will automatically insert them for you.

**NAMING AND STORING PROGRAMS.** To name and store a program which has been entered onto level 1 of the stack, press | to signify algebraic entry mode (suitable for entering names and expressions), then key in the desired name and press

**STO**. The program will be stored in user memory under its name, and pressing **VAR** on the 48S (or **USER** on the 28S) will show a user menu key with an *abbreviated* name (up to 5 characters).

**TO RUN A PROGRAM.** To run a program, simply press the white menu key beneath the program's abbreviated name; alternatively, key the *full* name into the command line and press **ENTER** **EVAL**. If the program happens to be on level 1, you may simply press **EVAL**. Of course, if the program requires input data for its proper execution then you must first provide that data in an appropriate way, either on the stack or as stored variables which are named in the program body.

**EXAMPLE.** The program « DUP SQ SWAP INV +  $\sqrt{\phantom{x}}$  INV » takes a number "a" from level 1 as input data and returns the calculated value of  $\frac{1}{\sqrt{a^2 + 1/a}}$  to level 1.

Key in the program by first pressing « » on the 48S (or « on the 28S), then **←** **ENTER** on the 48S (or the STACK menu key **DUP** on the 28S), followed by the other indicated commands. Press **ENTER** to add the closing program delimiters and copy the program to the stack.

Press **,** PGM1 **STO** to name this program PGM1 and store it in user memory under this label. Press **VAR** (or **USER**) to see the menu key **PGM1**.

Now, run the program using as input data the number 2: key in 2 and press **PGM1**. The answer, .471404520791, will be displayed on level 1. Notice that you did not have to enter the data onto the stack before pressing **PGM1**. This is typical; pressing the menu key **PGM1** automatically entered the data for you. Run the program with some more inputs.



**SYNTAX ERRORS.** When keying a program into the command line, if an object is accidentally entered in an invalid form, then pressing **ENTER** will cause the calculator to refuse to copy the program onto the stack and display a message indicating a syntax error. To remove the message from the screen so you can correct the syntax, simply press **ON** on the 48S (or **INS** on the 28S).

**EXAMPLE.** Key in : « → ARRY **ENTER** . Notice what happens. Now remove the message, delete the space after the →, and press **ENTER**.

**EDITING PROGRAMS.** To make any change in the body of an existing program you must *edit* the program.

- If the program is on stack level 1, the **EDIT** key will copy it into the command line where you can then make the required changes. Press **ENTER** to return the corrected version to level 1.
- If the program is not on stack level 1, but stored in user memory under, say, 'NAME' the keystrokes **⌈** **NAME** **RCL** will recall the program to level 1 and you can proceed as above. Alternatively (and indeed, *preferably*), **⌈** **NAME** **VISIT** will copy the program directly from user memory onto the command line for editing; **ENTER** will then replace the old stored program with the newly edited version in user memory.

**EXAMPLE.** Start this example with the program « PROGRAM MODI » stored in user memory as **TRY1**.

- Recall it to stack level 1 with **⌈** **TRY1** **RCL** .
- Copy it to the command line with **EDIT**, and change MODI to BODI.
- Copy back to level 1 with **ENTER**, then replace the old version with the new one by pressing **⌈** **TRY1** **STO** .

- (iv) Now copy this new version directly to the command line with , TRY1 VISIT , and change BODI to BODY.
- (v) Replace the earlier version by pressing ENTER .
- (vi) Finally, check your last work by recalling to level 1, examining the result, then dropping it from the stack with DROP.

### HP-48S SHORTCUTS:

- You can recall to level 1 the contents of any stored variable, say TRY1, by pressing → TRY1. Thus, *rightshift will recall*.
- Likewise, you can store (or load) an object on level 1 into any stored variable, by pressing ← , then the variable's menu key. Thus, *leftshift will load*. Try this by loading « + SQ COS » into TRY1; now recall the contents.

**PURGING.** Imagine that you have stored an object under variable PGM1 in your user memory. The object may be any one of the variety of objects recognized by the calculator: a real number, an array, a program, etc. To *purge* variable PGM1 is to remove it and its contents entirely from user memory. Purging a single variable is usually done with the keystrokes , PGM1 PURGE. The label disappears from the menu and its contents are removed from user memory. To purge several variables at the same time press { } on the 48S (or α { } on the 28S), then the menu key for each variable you wish to purge, then ENTER. Now press PURGE to purge the variables in this list.

**EXAMPLE.** Start by storing the numbers 1, 2 and 3 in variables 'X' 'Y' and 'Z' in user memory. With your VAR or USER menu active, purge 'X' by pressing X PURGE; watch X disappear. Now purge the two remaining variables at

the same time by building a list { Y Z } and pressing PURGE. Watch these variables disappear.

**EXERCISE.** The following program takes numbers  $x, y$  from the stack and returns  $(x + y)^2 \sqrt{x + y}$ .

« + DUP SQ SWAP  $\sqrt{\phantom{x}}$  \* »

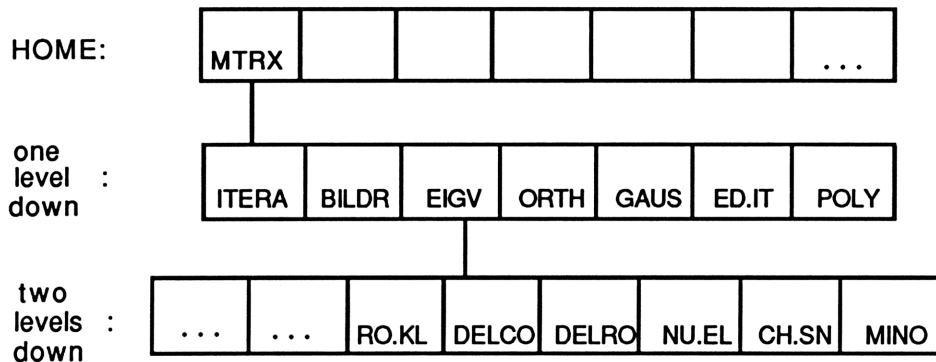
- (a) Key in this program and store it under variable "EX.1".
- (b) Run the program with inputs 9, 16.
- (c) Change the program body by replacing the \* with / and adding NEG at the end.
- (d) Run the new program with 9, 16.
- (e) In terms of  $x$  and  $y$ , what does the new program calculate?
- (f) Purge this program.
- (g) Purge programs TRY1 and PGM1 simultaneously.

## APPENDIX 2

### PROGRAM ORGANIZATION: DIRECTORIES

Just as a file cabinet organizes stored material in an office for convenient access, *directories* enable you to organize the programs (variables) you store in user memory. The original user memory is itself a directory - the HOME directory, and you can always go to HOME. And, in much the same way that certain drawers in a file cabinet are further subdivided into sections, you can create *subdirectories* within the directories. The basic idea is to group together programs associated with a particular topic or subtopic.

A convenient directory structure for the material in this book is as follows:



HOME contains various entries, one being the MTRX subdirectory, in which you may group together all the stored programs which deal with matrix topics. HOME is the *parent* directory of subdirectory MTRX, and MTRX is the parent of the subdirectories appearing in the next level: POLY, ED.IT, GAUSS, ORTH, EIGV, BILDR and ITERA. Each subdirectory groups together the programs associated with a particular topic.

The programs in this book have already been structured into a MTRX directory and loaded onto a computer disk which is available from the publisher to instructors who wish to adopt the book for classroom use. The MTRX directory may be transferred to students using the HP-48S either by calculator-to-calculator infrared transmission or by computer-to-calculator serial transmission as directed in the Owner's Manual. No such option is available for HP-28S users, and the programs must be organized into subdirectories by the individual users themselves.

- Subdirectory ED.IT: contains the matrix editing programs from sections 2.2. - 2.4 of Chapter 2.
- Subdirectory GAUSS: contains the programs relating to Gaussian elimination from Chapter 3.
- Subdirectory ORTH: contains the programs relating to orthogonality concepts from Chapter 4.
- Subdirectory EIGN: contains the eigenvalue/eigenvector programs from Chapter 5.
- Subdirectory BILDR: contains the programs from section 2.5 of Chapter 2 used to build various matrices over  $Z_{10}$ .
- Subdirectory ITERA: contains the programs dealing with the iterative methods of Chapter 6.
- Subdirectory POLY: contains some useful programs from Hewlett-Packard for working with polynomials; the programs appear in Appendix 4.

On the 28S, each subdirectory should contain a variable QUIT which may be used to return you to HOME, and each subdirectory two levels below HOME should also

contain a variable UP which may be used to return you to the parent directory. QUIT and UP are not needed in HP-48S subdirectories since HOME and UP are accessible with the shifted , key.

Here's how to create the first subdirectory, subdirectory MTRX: press , MTRX CRDIR (note that CRDIR appears on the MEMORY menu). A new label MTRX will appear in your VAR or USER menu. Pressing MTRX will send you to this new MTRX subdirectory, which is empty since you haven't stored any variables there yet.

The first variable to store in the MTRX subdirectory on the 28S, indeed in *any* 28S subdirectory, is QUIT - which will send you back to HOME. Create variable QUIT by pressing « HOME ENTER , QUIT STO. Notice that the label QUIT appears as the first label in the MTRX subdirectory. As you enter other variables QUIT will be pushed to the right and always remain as the last (right-most) variable.

You should now create the following subdirectories within MTRX: POLY, ED.IT, GAUSS, ORTH, EIGV, BILDR and ITERA. Make sure you are in the MTRX subdirectory before creating these new directories. After you have created them, you should enter each one and create the QUIT program, followed by the UP program. To create UP, press « MTRX ENTER , UP STO. Pressing UP will send you up to the MTRX directory.

Now that you have established an appropriate directory structure, you should put into ED.IT all the matrix editing programs from Sections 2.2 - 2.4 of Chapter 2. Since these programs may initially be in your HOME directory, you may need to transfer them to ED.IT. To transfer a variable from HOME, recall its contents to the stack with RCL and enter the name. Then go to the ED.IT subdirectory and press STO. Finally, purge the variable from HOME.

The contents of subdirectories GAUSS, ORTH, EIGV, BILDR, ITERA and POLY are detailed in the above bullets.

After entering the required programs into a subdirectory, you may arrange them in any order you prefer. For 28S directories, we recommend that QUIT be the last (*i.e.*, the right-most) variable in the directory, immediately preceded by UP. Program Index I contains an alphabetical listing of the programs within each subdirectory, and Program Index II is an alphabetical listing of all the programs in this book.

Let's say you have variables U, V, W and Z in a directory and want to arrange them in left-to-right order as W, Z, V, U. Simply build a list { W Z V U } containing the variables in the desired order and press ORDER, located on the MEMORY menu. When you return to your subdirectory, the variables will appear in that order.

### APPENDIX 3

## VECTOR AND MATRIX NORMS

**NORMS IN GENERAL.** When a vector  $v = (x_1, x_2, x_3)$  in  $\mathbb{R}^3$  is interpreted geometrically, its length is given by  $\|v\| = (x_1^2 + x_2^2 + x_3^2)^{1/2}$ . The well-known properties of vector lengths include:

- (1)  $\|v\| > 0$  if  $v \neq 0$ ,
- (2)  $\|\alpha v\| = |\alpha| \|v\|$  for any scalar  $\alpha$  and any vector  $v$ ,
- (3)  $\|v + w\| \leq \|v\| + \|w\|$  for any vectors  $v, w$ .

It would seem natural to adopt the corresponding notion for length, or magnitude, in  $\mathbb{R}^n$ : for  $v = (x_1, x_2, \dots, x_n)$ ,  $\|v\| = (x_1^2 + x_2^2 + \dots + x_n^2)^{1/2}$ . But there are situations where other scalar measures of the "size" of vectors in  $\mathbb{R}^n$  is more meaningful. For instance, if the components of  $v = (6, 3, 2, 5, 9)$  record the average times required to complete different components in an assembly operation, then  $(6^2 + 3^2 + 2^2 + 5^2 + 9^2)^{1/2}$  is somewhat meaningless when compared to  $6 + 3 + 2 + 5 + 9$  (the sum of the average assembly times) or  $\max \{ 6, 3, 2, 5, 9 \}$  (the largest average assembly time). Thus, in general, several different notions of length, or size, of vectors may be useful.



The term *norm* is usually applied to any generalization of Euclidean length in  $\mathbb{R}^3$ , as long as the above three conditions are met. The most commonly used norms for vectors in  $\mathbb{R}^n$  are:

- The *Euclidean vector norm*:  $\|v\|_2 = \left[ \sum_i |x_i|^2 \right]^{1/2}$
- The *vector sum norm*:  $\|v\|_1 = \sum_i |x_i|$
- The *vector max norm*:  $\|v\|_\infty = \max_i |x_i|$ .

All of these are true vector norms, in the sense that they satisfy conditions (1) - (3) above.

Analogous to vector norms are *matrix norms*,  $\|A\|$ , which are scalar measures of square matrices. To qualify as a matrix norm, the number  $\|A\|$  must satisfy:

- (1)  $\|A\| > 0$  if  $A \neq 0$ ,
- (2)  $\|\alpha A\| = |\alpha| \|A\|$ , for any scalar  $\alpha$  and any matrix  $A$ ,
- (3)  $\|A + B\| \leq \|A\| + \|B\|$ , for any matrices  $A, B$ ,
- (4)  $\|AB\| \leq \|A\| \|B\|$ , for any matrices  $A, B$ .

Conditions (1) - (3) are the same as for vector norms, but (4) is new and implies that  $\|A^n\| \leq \|A\|^n$ . Thus if  $\|A\| < 1$  then  $\|A^n\| \rightarrow 0$  as  $n \rightarrow \infty$ .

When our earlier examples of vector norms are applied to square matrices, the first two are matrix norms:

- the *Euclidean (or Frobenius) norm*:  $\|A\|_F = \left[ \sum_{i,j} |x_{ij}|^2 \right]^{1/2}$ , and
- the *sum norm*:  $\|A\| = \sum_{i,j} |a_{ij}|$ ;

but the third one,  $\|A\| = \max_{i,j} |a_{ij}|$  fails to be a matrix norm only because condition (4) need not hold.

While there are many ways to define matrix norms, it is especially useful to use a matrix norm that is connected to an existing vector norm. This may be done as follows: given a vector norm  $\|x\|$  on vectors  $x$  in  $\mathbb{R}^n$ , define a matrix norm  $\|A\|$  for square matrices by  $\|A\| = \max_{\|x\|=1} \|Ax\|$ .

This produces a true matrix norm (i.e., (1) - (4) hold) which measures the amount by which a vector  $x$  of norm 1 is "magnified" by matrix  $A$ . We call  $\|A\|$  the matrix norm *induced* by the vector norm  $\|x\|$ . The most important properties of induced matrix norms are

- (5)  $\|Ax\| \leq \|A\| \|x\|$  for all  $x$ , and
- (6)  $\|I\| = 1$ .

When the earlier three vector norms are used to induce matrix norms, it can be proved<sup>1</sup> that:

- the vector sum norm induces  $\|A\|_1 = \max_j \sum_i |a_{ij}|$ , the *column-sum norm* of  $A$
- the vector max norm induces

---

<sup>1</sup>See, e.g., Section 5.6 in Matrix Analysis, by Horn and Johnson, Cambridge University Press, 1985.

$\|A\|_{\infty} = \max_i \sum_j |a_{ij}|$ , the *row-sum norm* of  $A$

- the Euclidean norm induces

$\|A\|_2 = \max \{\sqrt{\lambda} : \lambda \text{ is an eigenvalue of } A^T A\}$ , the *spectral norm* of  $A$ .

Of these norms, the column-sum and row-sum norms are in widespread use because they are so easy to calculate. The Frobenius norm is also easy to calculate but is not induced by a vector norm. The spectral norm, on the other hand, is much more difficult to obtain and is mainly for theoretical use.

The spectral norm is not the only connection between matrix norms and eigenvalues. For any square matrix  $A$ , its *spectral radius*  $\rho(A)$  is defined by

$\rho(A) = \max \{ |\lambda| : \lambda \text{ is an eigenvalue of } A \}$ , and it can be shown that  $\rho(A) \leq \|A\|$  for any matrix norm. Thus the column-sum and row-sum norms provide easy estimates of  $\rho(A)$ .

**NORMS ON THE CALCULATOR.** Three matrix and vector norms are provided on the MTH MATR menu of the 48S and on the ARRAY menu of the 28S: the Euclidean norm, the  $\infty$ -norm and the 1-norm.

- The Euclidean (Frobenius) matrix norm:  $\|A\|_F = \boxed{\text{ABS}}$ . Since vectors on the calculator are 1-dimensional arrays sensed as column vectors,  $\boxed{\text{ABS}}$  applied to a vector  $v$  returns its vector Euclidean norm  $\|v\|_2$ .
- The row-sum, or  $\infty$ -norm:  $\|A\|_{\infty} = \boxed{\text{RNRN}}$ . For a vector  $v$ ,  $\boxed{\text{RNRN}}$  returns its vector max norm  $\|v\|_{\infty}$ .
- The column-sum, or 1-norm:  $\|A\|_1 = \boxed{\text{CNRN}}$ . For a vector  $v$ ,  $\boxed{\text{CNRN}}$  returns its vector sum norm  $\|v\|_1$ .

**EXAMPLE**

- (a) For  $A = \begin{bmatrix} 4 & -3 & 0 \\ 0 & 5 & -1 \\ 2 & 0 & -3 \end{bmatrix}$ , press **ABS** to return  $\|A\|_F = 8$ , **RNRM** to return  $\|A\|_\infty = 7$  and **CNRM** to return  $\|A\|_1 = 8$ .
- (b) For  $v = [-1 \ 2 \ 5 \ 1 \ -2 \ 1]$ , **ABS** returns  $\|v\|_2 = 6$ , **RNRM** returns  $\|v\|_\infty = 5$ , and **CNRM** returns  $\|v\|_1 = 12$ .

**EXERCISES**

1. (a) Calculate the Frobenius norm for  $I_3, I_4, I_5, I_9$  and  $I_{16}$ .
- (b) On the basis of your results in (a), formulate a conjecture about  $\|I_n\|_F$ .
- (c) Prove your conjecture.
2. (a) Compare the row-sum and column-sum norms for each of the following matrices:

$$A = \begin{bmatrix} 1 & 3 \\ 3 & -2 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 & 0 \\ 2 & -1 & 3 \\ 0 & 3 & 5 \end{bmatrix} \quad C = \begin{bmatrix} 0 & 1 & 0 & -2 \\ 1 & 3 & -4 & 2 \\ 0 & -4 & -1 & 0 \\ -2 & 2 & 0 & 5 \end{bmatrix}$$

- (b) Formulate a conjecture based on your results in (a).
- (c) Prove your conjecture.

3. Use the matrix  $A$  and vector  $x$  given below to verify that  $\|Ax\| \leq \|A\| \|x\|$  for the three matrix and vector norms provided by the calculator.

$$A = \begin{bmatrix} -3 & 0 & 0 \\ 2 & 1 & 5 \\ 1 & -4 & 4 \end{bmatrix} \quad x = \begin{bmatrix} -1 & 2 & 1 \end{bmatrix}$$

4. Consider the attempt to define a matrix norm by  $\|A\| = \max_{i,j} |a_{ij}|$ . Experiment with random  $3 \times 3$  matrices over  $Z_{10}$  to find a pair  $A, B$  for which the inequality  $\|AB\| \leq \|A\| \|B\|$  is invalid. (To calculate  $\max_{i,j} |a_{ij}|$  for matrix  $A$ , separate  $A$  into its entries with  $\boxed{\text{OBJ} \rightarrow}$  on the 48S or with  $\boxed{\text{ARRY} \rightarrow}$  on the 28S, then reassemble into a vector  $v$  with  $\boxed{\rightarrow \text{ARRY}}$ . Now apply  $\boxed{\text{RNRM}}$  to  $v$ .)

## APPENDIX 4

### POLYNOMIAL ROUTINES

In the course of calculating the eigenvalues of a matrix  $A$  as the roots of the characteristic polynomial (see Chapter 5), we have found it convenient to use several polynomial routines for the 28S prepared by William Wickes of Hewlett-Packard. These are among a host of such routines, as well as other interesting programs and examples for the 28S, which are the principal contents of *Mathematical Applications*, Copyright Hewlett-Packard Company, 1988. We strongly recommend that you purchase this booklet [call toll-free 1-800-752-0900 for the location and number of the U.S. dealer nearest you]. To make this book as self-contained as possible, Hewlett-Packard has granted permission to reproduce the several programs we need here.

The polynomial routines we require are:

- **PROOT** (finds all roots, real and complex, of a polynomial of degree  $\leq 4$  having real or complex coefficients).
- **PSERS** (returns an algebraic expression (or value) from a list of coefficients)
- **PVAL** (uses Horner's method to return an algebraic expression (or value) from a list of coefficients)
- **PDIV** (implements the division algorithm for polynomials; returns quotient and remainder)

We shall present these routines in reverse order.

**PDIV** (Polynomial Divide)*Inputs:* level 2: a dividend list {  $a_n \ a_{n-1} \ \dots \ a_1 \ a_0$  }level 1: a divisor list {  $b_m \ b_{m-1} \ \dots \ b_1 \ b_0$  }*Effect:* divides  $a(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$  by  $b(x) = b_m x^m + b_{m-1} x^{m-1} + \dots + b_1 x + b_0$  to obtain a quotient  $q(x)$  and a remainder  $r(x)$ ; the quotient, remainder and divisor polynomials are shown in list form:

level 3: { quotient }

level 2: { remainder }

level 1: { divisor }

```
« DUP 1 GET OVER SIZE → d t n « { } SWAP DUP
SIZE n - 1 + 1 SWAP START DUP 1 GET t / COLCT
ROT OVER 1 →LIST + 3 ROLLD 1 n FOR m OVER m
GET d m GET 3 PICK * - ROT m ROT PUT SWAP NEXT
DROP 2 100 SUB NEXT d » »
```

Store this program with . PDIV STO in your POLY subdirectory.

**EXAMPLE 1.** To divide  $x^5 + 2x^4 + x^3 - 3x^2 - 6x - 3$  by  $x^3 - 3$  press { 1, 2, 1, -3, -6, -3 ENTER } then { 1, 0, 0, -3 PDIV }. You will see:

level 3: { 1 2 1 }, the quotient list

level 2: { 0 0 0 }, the remainder list

level 1: { 1 0 0 -3 }, the divisor list.

**PVAL** (Polynomial value)*Inputs:* level 2: a list of coefficients {  $a_n$   $a_{n-1}$  ...  $a_1$   $a_0$  }level 1: a number, or a variable,  $x$ *Effect:* returns ( ...  $(a_n x + a_{n-1})x + a_{n-2}$  )  $x + \dots$  ) +  $a_0$ , in Horner's form; if  $x$  is a real or complex number, the result is also a real or complex number, the result is also a real or complex number; if  $x$  is symbolic, say " $x$ ", the result is the indicated symbolic Horner expression.

```
« → st x « st 1 GET 2 st SIZE FOR n x * st n GET +
NEXT » »
```

Store with P PVAL STO in your POLY subdirectory.**EXAMPLE 2.**(a) { 4, 3, 2, 1 ENTER X PVAL returns '( ( 4 \* X + 3 ) \* X + 2 \* X + 1 '(b) UNDO 2 PVAL returns 49, the value of the result in (a) when  $x = 2$ .



**PSERS** (Polynomial series)*Inputs:* level 2: a list of coefficients {  $a_n$   $a_{n-1}$  ...  $a_1$   $a_0$  }level 1: a number, or variable,  $x$ *Effect:* returns  $a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ ; if  $x$  is a real or complex number this result is a real or complex number; if  $x$  is symbolic, say " $x$ ", the result is the indicated symbolic expression.

```

« → x « LIST→ 0 SWAP 1 FOR n n 1 + ROLL x n 1 - ^
* + -1 STEP » »

```

Store with . PSERS STO in your POLY subdirectory.**EXAMPLE 3.**(a) { 4, 3, 2, 1 ENTER X PSERS returns ' 4 \* X ^ 3 + 3 \* X ^ 2 + 2 \* X + 1 '(b) UNDO 2 PSERS returns 49, the value of the result in (a) when  $x = 2$ .

**PROOT** (Polynomial root finder)

*Inputs:* Level 1: a list of coefficients {  $a_n$   $a_{n-1}$  ...  $a_1$   $a_0$  }  
 where  $n \leq 4$

*Effect:* Returns on levels 1 thru 4 all roots, real or complex,  
 of the polynomial:  $a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$

*Required programs:*

- QUD
- CUBIC
- QUAR

```
« LIST→ →ARRY DUP 1 GET / ARRY→ LIST→ DROP 1 -
  DUP 2 + ROLL DROP { NEG QUD CUBIC QUAR } SWAP
  GET EVAL »
```

Store in your POLY subdirectory with , PROOT STO.

**QUD** (Quadratic subroutine)

```
« SWAP 2 / NEG DUP SQ ROT - √ DUP2 + 3
  ROLL - »
```

Store in your POLY subdirectory with , QUD STO.

**CUBIC**

(Cubic solve subroutine)

```

« 3 PICK -3 / 3 PICK 5 PICK SQ 3 / - 5 ROLL DUP 3
^ 2 * SWAP 9 * 6 ROLL * - 27 / 4 ROLL + NEG
OVER ABS 0 IF == THEN 3 INV ^ SWAP DROP 0
SWAP ELSE 2 / DUP SQ 3 PICK 3 ^ 27 / +  $\sqrt{\quad}$  - 3
INV ^ SWAP OVER / 3 / NEG END -1 3  $\sqrt{\quad}$  R→C 2 /
4 ROLLD 3 DUPN 3 DUPN + + 8 ROLLD 7 PICK *
SWAP 7 PICK / + + 5 ROLLD 4 PICK / SWAP 4 ROLL
* + + »

```

Store in your POLY subdirectory with , CUBIC STO.

**QUAR (Quartic solver subroutine)**

*Required Subprograms:* • QUD  
• CUBIC

```
« 3 PICK NEG DUP 6 ROLL 5 PICK 4 PICK * 3 PICK
4 * - 5 ROLL 4 * 6 PICK SQ - 4 PICK * 5 PICK SQ -
CUBIC 3 →LIST 1 3 FOR n DUP n GET 2 / SQ n 2 +
PICK - ABS SWAP NEXT 4 ROLL 2 DUP2 IF < THEN
SWAP DROP 3 ELSE DROP 2 END 3 ROLL 2 IF > THEN
DROP 1 END GET 4 ROLL 2 / SWAP 2 / DUP SQ 4
ROLL - √ IF DUP ABS THEN 3 DUPN 3 ROLL 2 * 6
ROLL 2 / - SWAP / ELSE 3 PICK SQ 6 ROLL + 3 PICK
2 * + √ END 5 ROLL DROP 3 ROLL 4 DUPN + 3
ROLL + SWAP QUD 6 ROLL 6 ROLL - 3 ROLL -
SWAP QUD »
```

Store in your POLY subdirectory with . QUAR STO.

**EXAMPLE 4.** To find all the roots of  $3x^4 + 6x^2 - 24$ , press { 3, 0, 6, 0, -24 ENTER  
PROOT. Then use 9 CLEAN and DROP to see the following roots:

-1.414213562

1.44213562

2i

-2i

# PROGRAM INDEX I

(Alphabetically within subdirectories)

## ED.IT SUBDIRECTORY

<b>AD.COL</b>	Additional Column.....	23
<b>AD.ROW</b>	Additional Row .....	23
<b>A.KTH</b>	K <sup>th</sup> Power of a Matrix.....	26
<b>C.COL</b>	Extract a Matrix Column .....	17
<b>CLEAN</b>	Clean-up Round-Off.....	37
<b>C.ROW</b>	Extract a Matrix Row.....	17
<b>CH.SN</b>	Change Sign.....	16
<b>DELCOL</b>	Delete a Matrix Column.....	19
<b>DELROW</b>	Delete a Matrix Row.....	19
<b>MINOR</b>	Matrix Minor.....	20
<b>NU.EL</b>	New Matrix Element.....	15
<b>P.OF.A</b>	Polynomial Evaluation at A.....	27
<b>RO.KL</b>	Interchange Rows K and L.....	24
<b>ROW→</b>	Assemble Rows into a Matrix .....	21
<b>→ROW</b>	Separate into Rows .....	21

## GAUSS SUBDIRECTORY

<b>BACK</b>	Back Substitution .....	51
<b>CLEAN</b>	Clean-up Round-off.....	37
<b>ELIM</b>	Gaussian Elimination .....	49
<b>FWD</b>	Forward Substitution.....	62
<b>GJ.PV</b>	Gauss-Jordan Pivot.....	68
<b>LU</b>	LU-Factorization.....	57
<b>MAKL</b>	Make L and P Subroutine.....	58
<b>RO.KL</b>	Interchange rows K and L.....	24
<b>SPLIT</b>	Split-off last column.....	52

**ORTH SUBDIRECTORY**

<b>CLEAN</b>	Clean-up Round-off.....	37
<b>DPLOT</b>	Data Plot.....	95
<b>FEVAL</b>	Function Evaluation.....	98
<b>P.FIT</b>	Polynomial Fit Matrix.....	93
<b>PROJ</b>	Projection Vector.....	89
<b>OVLAY</b>	Overlay a Plot (28S only).....	96

**EIGV SUBDIRECTORY (48S VERSION)**

<b>CHAR</b>	Characteristic Polynomial.....	112
<b>CLEAN</b>	Clean-up Round-off.....	37
<b><math>\lambda</math>.VALUES</b>	Eigenvalues.....	117
<b>PROOT</b>	Polynomial Root Finder (code protected).....	disk
<b>TRACE</b>	Trace of a Matrix.....	112

**EIGV SUBDIRECTORY (28S VERSION)**

<b>CHAR</b>	Characteristic Polynomial.....	112
<b>CLEAN</b>	Clean-up Round-off.....	37
<b>CUBIC</b>	Cubic Solve Subroutine.....	168
<b>PROOT</b>	Polynomial Root Finder ( $n \leq 4$ ).....	167
<b>QUAR</b>	Quartic Solver Subroutine.....	169
<b>QUD</b>	Quadratic Subroutine.....	167
<b>TRACE</b>	Trace of a Matrix.....	112

**POLY SUBDIRECTORY**

<b>CLEAN</b>	Clean-up Round-off.....	37
<b>PDIV</b>	Polynomial Divide.....	164
<b>PSERS</b>	Polynomial Series.....	166
<b>PVAL</b>	Polynomial Value.....	165

## BILDR SUBDIRECTORY

<b>DIAG</b>	Diagonal Matrix Generator .....	42
<b>L.TRI</b>	Unit Lower Triangular Matrix Generator.....	44
<b>RAN.Z</b>	Random Matrix Generator .....	42
<b>SYMM</b>	Symmetric Matrix Generator .....	45
<b>TRIDIA</b>	Tridiagonal Matrix Generator .....	44
<b>U.TRI</b>	Upper Triangular Matrix Generator.....	43

## ITERA SUBDIRECTORY

<b>D.DOM</b>	Diagonal Dominance .....	136
<b>JACOBI</b>	Jacobi Iteration.....	135
<b>JTEST</b>	Test Jacobi Iteration Matrix.....	134
<b>POWER</b>	Power Method.....	143
<b>SEIDL</b>	Gauss-Seidel Iteration.....	137
<b>STEST</b>	Test Gauss-Seidel Iteration Matrix.....	136

## PROGRAM INDEX II

(alphabetical order)

<b>A.KTH</b> .....	<b>K<sup>th</sup> Power of a Matrix</b> .....	<b>26</b>
<b>AD.COL</b> .....	<b>Additional Column</b> .....	<b>23</b>
<b>AD.ROW</b> .....	<b>Additional Row</b> .....	<b>23</b>
<b>BACK</b> .....	<b>Back Substitution</b> .....	<b>51</b>
<b>C.COL</b> .....	<b>Extract a Matrix Column</b> .....	<b>17</b>
<b>C.ROW</b> .....	<b>Extract a Matrix Row</b> .....	<b>17</b>
<b>CH.SN</b> .....	<b>Change Sign</b> .....	<b>16</b>
<b>CHAR</b> .....	<b>Characteristic Polynomial</b> .....	<b>112</b>
<b>CLEAN</b> .....	<b>Clean-up Round-off</b> .....	<b>37</b>
<b>CUBIC</b> .....	<b>Cubic Solve Subroutine</b> .....	<b>168</b>
<b>D.DOM</b> .....	<b>Diagonal Dominance</b> .....	<b>136</b>
<b>DELCOL</b> .....	<b>Delete a Matrix Column</b> .....	<b>19</b>
<b>DELROW</b> .....	<b>Delete a Matrix Row</b> .....	<b>19</b>
<b>DIAG</b> .....	<b>Diagonal Matrix Generator</b> .....	<b>42</b>
<b>DPLOT</b> .....	<b>Data Plot</b> .....	<b>95</b>
<b>ELIM</b> .....	<b>Gaussian Elimination</b> .....	<b>49</b>
<b>FEVAL</b> .....	<b>Function Evaluation</b> .....	<b>98</b>
<b>FWD</b> .....	<b>Forward Substitution</b> .....	<b>62</b>
<b>GJ.PV</b> .....	<b>Gauss-Jordan Pivot</b> .....	<b>68</b>
<b>GRAM-SCHMIDT</b> .....	<b>Orthonormal Procedure</b> .....	<b>101</b>
<b>JACOBI</b> .....	<b>Jacobi Iteration</b> .....	<b>135</b>
<b>JTEST</b> .....	<b>Test Jacobi Iteration Matrix</b> .....	<b>134</b>
<b>L.TRI</b> .....	<b>Unit Lower Triangular Matrix Generator</b> .....	<b>44</b>
<b>λ.VALUES</b> .....	<b>Eigenvalues via PROOT (code protected)</b> .....	<b>117</b>
<b>LU</b> .....	<b>LU-Factorization</b> .....	<b>57</b>
<b>MAKL</b> .....	<b>Make L and P Subroutine</b> .....	<b>58</b>



<b>MINOR</b> .....	Matrix Minor.....	20
<b>NU.EL</b> .....	New Matrix Element.....	15
<b>P.FIT</b> .....	Polynomial Fit Matrix.....	93
<b>P.OFA</b> .....	Polynomial Evaluation at A .....	27
<b>PDIV</b> .....	Polynomial Divide .....	164
<b>POWER</b> .....	Power Method .....	143
<b>PROJ</b> .....	Projection Vector .....	89
<b>PROOT</b> .....	Polynomial Root Finder ( $n \leq 4$ ) .....	167
<b>PROOT</b> .....	Polynomial Root Finder (code protected).....	disk
<b>PSERS</b> .....	Polynomial Series.....	166
<b>PVAL</b> .....	Polynomial Value.....	165
<b>QUAR</b> .....	Quartic Solver Subroutine .....	169
<b>QUD</b> .....	Quadratic Subroutine.....	167
<b>RO.KL</b> .....	Interchange Rows K and L.....	24
<b>ROW→</b> .....	Assemble Rows into a Matrix.....	21
<b>SEIDL</b> .....	Gauss-Seidel Iteration.....	137
<b>SPLIT</b> .....	Split-off Last Column .....	52
<b>STEST</b> .....	Test Gauss-Seidel Iteration Matrix .....	136
<b>SYMM</b> .....	Symmetric Matrix Generator.....	45
<b>TRACE</b> .....	Trace of a Matrix.....	112
<b>TRIDIA</b> .....	Tridiagonal Matrix Generator.....	44
<b>U.TRI</b> .....	Upper Triangular Matrix Generator.....	43
<b>→ROW</b> .....	Separate into Rows.....	21

## S U B J E C T     I N D E X

- Back-substitution 48, 51
- Basis 76, 78
- Cauchy-Schwartz 86
- Change of basis 78
- Characteristic polynomial 111,112
- Column space 76
- Column sum norm 133, 159
- Companion matrix 114
- Conjugate transpose 18, 83
- Coordinate matrix 79
- Crout decomposition 71
- Cullen, Charles 111
- Curve fitting 92
- Defective 124
- Dependence 74
- Determinant 33
- Diagonal matrix 40, 128
- Diagonalizable matrix 128
- Dimension 76
- Dominant eigenvalue 140
- Dot product 83
- Echelon matrix 49, 66
- Eigenspace 118, 119
- Eigenvalue 111
- Eigenvector 111
- Fitting data 92
- Forward Substitution 61, 62
- Free variable 67
- Frobenius norm 159
- Gauss-Jordan reduction 66
- Gauss-Seidel iteration 131, 137
- Gaussian elimination 48
- Givens rotation 108
- Gram-Schmidt process 99
- Hermitian product 83
- Hewlett-Packard 21, 115, 163
- Householder matrix 168
- Ill-conditioned 95
- Independence 74
- Inner product 83
- Interpolating polynomial 93
- Invertible matrix 36
- Iterative methods 130
- Jacobi iteration 131, 135
- Least squares 88
- LDL<sup>T</sup>-factorization 72
- LDU-factorization 72
- Lower triangular matrix 44, 56
- LU-factorization 55
- Matrix
  - change of basis 79
  - companion 114
  - defective 124
  - diagonalizable 128
  - echelon 49
  - Householder 168
  - ill-conditioned 95
  - iteration 131
  - lower triangular 44
  - norms 158
  - orthogonal 103
  - permutation 56
  - positive 120
  - similar 124

- strictly diagonally dominant 133
- symmetric 128
- upper triangular 43
- tridiagonal 44
- Vandermonde 93
- MatrixWriter 8
- Multiplicity 140
- Multiplier 56
- Norms
  - column-sum 159
  - Euclidean 158, 159
  - Frobenius 159
  - matrix 158
  - row-sum 160
  - spectral 160
  - sum 159
  - vector 158
  - vector-max 158
  - vector-sum 158
- Normal equations 91
- Normal vector 84
- Nullspace 76
- Orthogonal 84
- Orthogonal matrix 103
- Orthonormal basis 84
- Outer Product 46
- Overdetermined 90
- Partial pivoting 53
- Permutation matrix 56
- Perron 120
- Pivot 66
- Pivot variable 67
- Polynomial fit 92
- Positive matrix 120
- Power method 140, 143
- Projection 88, 89
- QR-algorithm 140
- QR-factorization 103
- Rank 91
- Reduced row echelon matrix 66
- Row space 76
- Row echelon 49
- Row sum norm 160
- RREF 66
- Similar matrices 124
- Souriau-Frame method 111
- Spanning set 76
- Spectral radius 160
- Strang, Gil 121
- Strictly Diagonally Dominant 133
- Subspace 76
- Symmetric matrix 128
- Trace 111, 112
- Tridiagonal 44
- Unit lower triangular 44
- Upper triangular 43
- Vandermonde matrix 93
- Vector norms 158
- Wickes, William C. 1, 116, 163
- Williams, Gareth 71













**Also available from Saunders College Publishing in the  
Clemson Series:**

---

- **Calculator Enhancement for Introductory Statistics**  
By I.B. Fetta
- **Calculator Enhancement for Single Variable Calculus**  
By J.H. Nicholson
- **Calculator Enhancement for Differential Equations**  
By T.G. Proctor
- **Calculator Enhancement for a Course in Multivariable  
Calculus**  
By J.A. Reneke
- **Calculator Enhancement for Precalculus**  
By I.B. Fetta

**D.R. LaTorre, Clemson University, Consulting Editor**

ISBN 0-03-092729-3

