Instructor's MatLab® Manual

Jeremy R. Case
Taylor University

Jane Day
San Jose State University

Linear Algebra and Its Applications

Third Edition Update

David C. Lay
University of Maryland – College Park

Boston San Francisco New York
London Toronto Sydney Tokyo Singapore Madrid
Mexico City Munich Paris Cape Town Hong Kong Montreal

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.

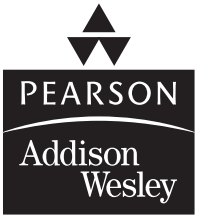
Reproduced by Pearson Addison-Wesley from electronic files supplied by the author.

Copyright © 2006 Pearson Education, Inc.
Publishing as Pearson Addison-Wesley, 75 Arlington Street, Boston, MA 02116.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America.

ISBN 0-321-28069-5

1 2 3 4 5 6 OPM 08 07 06 05



Contents

Forward

This manual is for any instructor who is using MATLAB and *Linear Algebra and Its Applications* together for the first time. It will greatly simplify your task of combining MATLAB with the text, because it is written by a colleague who has already tried out the materials with considerable success. This manual carefully describes everything you need to know about planning and conducting the course.

I am pleased with the work of the author, Professor Jeremy Case, who revised this MATLAB manual. He substantially reorganized and edited the main part, adding his own perspective based on his use of MATLAB and the text for four years. He also edited the projects to reflect changes in the new edition. The original material was written by Professor Jane Day, of San Jose State University, and revised often during more than eight years of teaching with both MATLAB and earlier versions of the text. Her work has influenced the teaching of hundreds of faculty, and I have appreciated her advice and support.

I am grateful for Professor Case's contributions to this manual, and I am confident that you and your students will appreciate his work.

David C. Lay

Introduction

I consider it a great privilege to take part in presenting this MATLAB manual to accompany the third edition of *Linear Algebra and Its Applications*. When I first began teaching Linear Algebra, I found Lay's textbook to closely match my course goals and objectives. I wanted my students to have a better conceptual understanding of linear algebra and a greater appreciation of its power and applicability than I had when I was a student. One of the benefits my students have is a tool such as MATLAB to help them see the larger picture without getting bogged down in the calculations. My students have reacted positively to the textbook and the use of MATLAB in the course, and I have been very pleased with the results

The purpose of this manual is to help you integrate computer exercises or projects into your linear algebra course. MATLAB and linear algebra work very well together, and your students will have an excellent opportunity to gain a better understanding of the material. The projects in this manual come from Professor Jane M. Day; I have made only minor changes in her work. Professor Day's projects and her earlier versions of this manual were of great help to me when I started using MATLAB. I hope you will find this manual helpful to you as well.

Jeremy Case
Department of Mathematics
Taylor University
Upland, IN 46989
jrcase@tayloru.edu

1. Getting Started

PREPARE

This manual assumes that you will be using MATLAB, but there are other excellent software packages or calculators you could use. Maple and Mathematica are examples of other mathematical software packages conducive to linear algebra. Calculators with matrix capabilities such as those made by Hewlett Packard and Texas Instruments are other possibilities. Each of these technologies listed here have manuals to accompany Lay's book and are available from Addison-Wesley. The exercises in Lay's text are written so that any appropriate software or calculator may be used.

This manual assumes that in addition to MATLAB you will use the Laydata Toolbox, which is a collection of M-files that can be downloaded from MyMathLab or accessed through **www.laylinalgebra.com**. These M-files will need to be made accessible to your students. It is also a good idea to use the *Study Guide,* a supplement to the text, to accompany this manual. The text, the *Study Guide,* and the Laydata Toolbox are highly coordinated to work together. For the students, the *Study Guide* is the primary support for the use of technology during the semester. In addition to the MATLAB boxes at the ends of many sections, the Appendix, "Getting Started with MATLAB," provides the initial information students may need. Furthermore, there are appendices for other technologies such as Maple, Mathematica, and several graphing calculators. For the instructor, your job will be much easier if you and your students have the *Study Guide* with its wealth of information.

If you have not used MATLAB extensively before, spend some time learning the basic operations before you begin class. MATLAB and linear algebra work very well together, and you can do some interesting things fairly quickly. You might work through the first project, "Getting Started with MATLAB," and then try some other projects. Exercises designated by an [M] in Lay's text are designed to be worked with the aid of technology, and you might try some of these problems. Read the MATLAB boxes in the *Study Guide* to see how MATLAB could be used on homework problems. One of the *Study Guide*'s features is that it gradually introduces MATLAB commands as they are needed. Alternatively, you could become introduced to MATLAB by working through a tutorial in a book or on the Web.

As you become more familiar with MATLAB's capabilities, attempt more involved problems such as the case studies and application projects available from the course website. These are usually found at the beginning of each chapter, and an icon in the text references the course website for these resources.

OBTAIN AN EDUCATIONAL LICENSE

Usually the most cost effective way to provide MATLAB to students is for your school to buy an educational site license. You can use any version of MATLAB for Lay's [M] exercises and most of the projects. Projects in this manual that involve plotting will work in Versions 4, 5 and 6. Graphics programs such as those from the ATLAST project [6] or other sources usually require Version 5 or 6.

ORDER STUDENT MATLAB AND STUDY GUIDES EARLY

Ask your bookstore to stock Lay's *Study Guide* with the textbook. Our institution has an educational site license for MATLAB, but if yours does not, request that the *Student Edition of MATLAB* [9] be made available as well. The latest edition of Student MATLAB is usually what they will get. Some students may have access to earlier editions, and those will work fine. Note the command **flops** is not available in Version 6, hence the project "Counting Floating Point Operations in Matrix Products" can only be done if you have an earlier version. Also, Versions 4 and 5 were available for both Windows and Macintosh platforms, but Version 6 was still in the process of being made available for Macintosh at the time of this manual's writing. Student MATLAB costs about $100 and includes a good *User's Guide*. It is identical to professional MATLAB except in a few ways that rarely affect students' use. Student Versions 5 and 6 are available on CD-ROM only.

INSTALL THE LAYDATA TOOLBOX BEFORE CLASSES START

The M-files in the Laydata Toolbox are not part of commercial MATLAB, so you must install them on the computers your students will use. If students plan to use MATLAB on their personal computers, they must also install the M-files. See Section 4 below and Section 15 of the preliminary Computer Project "Getting Started With MATLAB."

You should check the status of your software and M-files before each term begins. One year, our institution changed operating systems between semesters. What had worked one month previously no longer worked, and I spent the first week scrambling to correct it. It got the class off on the wrong foot, and it took longer for them to feel comfortable with the technology. It taught me to never take for granted what will work as computers are upgraded.

The files in the Laydata Toolbox provide the data for about 850 exercises in Lay's text, as well as the data for the individual projects printed at the back of this manual. Having the data for all numerical exercises readily available saves the tedium of typing it in and ensures that students work with the intended numbers. The Laydata Toolbox also contains some special MATLAB functions that enhance the teaching of linear algebra from Lay's text. These functions are described in Section 3 below, as well as in the *Study Guide* and in the projects as they are needed.

OBTAIN DATA FOR CASE STUDIES AND APPLICATION PROJECTS

In addition to the hard copy projects at the end of this manual, there are Case Studies and Application Projects available from the Web. The Case Studies expand topics introduced at the beginning of each chapter in Lay's textbook and use real-world data. The Application Projects either extend existing topics in the text or introduce new applications. The Data files for the Case Studies and Application Projects are contained in text files on the Web at **http://www.laylinalgebra.com**. If you decide to assign one or more of these projects that has accompanying data, then either you or your students must download the appropriate files from the Web and add them to the MATLAB path. See Section 4 below.

PREPARE STUDENT COMPUTER LAB INSTRUCTIONS

On the first day of classes, students need information about how you plan to use MATLAB in the course and how they can access the program and appropriate data. List the *Study Guide* as the "lab manual" for the course. With the *Study Guide* in hand, students rarely will need more documentation for the course other than MATLAB's **help** command and your local computer procedures. You can prepare a sheet to hand out, or put the information on a web page, or do both. Here are some facts that students may need:

- Location of campus computer lab facilities.
- Hours and days when the labs are available for student use.
- How to obtain and use computer log-on names and passwords.
- How to start MATLAB in the lab, print output, and save the work.
- Where to get help.
- How to get a personal copy of MATLAB and data for the course.

2. Planning the Course

ALLOW TIME FOR PLANNING AND ADJUSTING PLANS

It would be very good to have some release time the first time you try using a significant number of computer exercises. However, some institutions like mine cannot always provide such release time. Pressed for time, I found the projects in this manual to be very helpful the first time I taught linear algebra.

Starting out or making major changes in a course does take a lot of effort, and computers introduce another dimension for what can go wrong. I started slowly by using computer exercises as an "add on" to the traditional course. I think this is not an unwise way to begin. As you understand the technology and the students better, you can change the style and topics of your course. Students will have various interests and questions, and you should allow yourself some flexibility in modifying your course.

CONSIDER PURPOSES FOR COMPUTER ASSIGNMENTS

As you consider your students' interests and begin to appreciate the potential of computer exercises, decide what purposes are most appropriate for your class. Here are some possible ones:

1. To teach applications
2. To reinforce understanding of concepts and theory
3. To think and problem solve
4. To explore and conjecture
5. To develop some computational wisdom
6. To learn something new
7. To reduce tedious hand calculation
8. To practice routine calculations
9. To write programs to solve problems, learn algorithms, etc.

The first four purposes listed are the most important to me, but all of these reasons have merit and are addressed in various projects.

Applications provide motivation as to why one should learn the material. For many of my students, they seem more likely to forget material that they do not find interesting or that they cannot conceive of applying it in the future. You can expose your students to a variety of applications using the case studies in the book. Such examples are natural topics for computer exercises, because applications are more interesting when the data are not trivial.

The second goal is very important. People tend to misuse theory when they do not understand it, so you might stress the mastery of the big ideas such as linear independence, span, basis, dimension, eigenvalues and orthogonality. I believe students should also understand why the major theorems are true. However, many of my students do not appreciate the abstract concepts as much as I do, and it helps if they connect the concepts to ideas they already know. Practicing the ideas in concrete calculations and applications also solidifies the ideas. Furthermore, if they can be convinced that the theorems and ideas are reasonable and useful, they are more motivated. Computer exercises are an attractive device towards this end. It gets them to grapple with some theory, and some of the projects explicitly address abstract ideas. All projects ask students to explain what they have seen, and most points should be placed on those questions.

I consider the third and fourth purposes part of the broader scheme of mathematics. I will then add some questions to the projects asking students to extend the results. At my institution, linear algebra is still a senior level course while at many schools linear algebra is designed for underclassmen. Hence, while I did not modify the projects to include these questions, I encourage you to adapt the projects toward your goals.

Jane Day stresses the development of computational wisdom. Numerical issues and peculiarities come up in several of the projects, including "Reduced Echelon Form and **ref**," "Counting Floating Point Operations" and "Roundoff Error in Matrix Computations." While not undermining the faith in good software, she wants her students to learn to be wise and cautious. To her students she emphasizes the following, in this order:

1. Professionally written matrix software gives good answers to most problems, but there is almost always some error.
2. The matrix algorithms that work well on computers are more sophisticated than those presented in basic linear algebra texts. So people should employ professionally written software in their jobs.
3. Some problems are inherently difficult to solve accurately even with the best algorithms. So users should never ignore warnings from professional software.

For students who want to know more, Watkins' text [10] is a well-written contemporary introduction to numerical linear algebra, and George Forsythe's classic paper [4] is great. It is remarkable how clearly he articulated the inevitable pitfalls of floating point matrix computations, so early.

DECIDE EMPHASIS ON COMPUTER WORK

Computer projects are good vehicles for introducing simple applications, which are important for the majority of our students. One important reason for having linear algebra students work with professional software like MATLAB is that they need to know such software exists. In the workplace, they will be using professional software that will be far faster and more accurate than code based on algorithms alone. At the same time, I feel it is important for them to learn the basic algorithms of linear algebra because those enhance understanding of concepts.

Some instructors organize the computer work around weekly lists of five or more [M] exercises from the text. Other instructors include one or more [M] exercises in nearly every night's homework. Here, the point is to encourage daily use of MATLAB for most of the numerical exercises, not just those marked with [M]. Remember that the Laydata Toolbox provides data for almost all of these exercises in the text so that the time entering data on the computer is minimal.

One obvious factor in determining how much to emphasize computer work is the availability of a computer lab. Currently, my institution does not have the space available for a weekly lab so most of the computer work is done outside of class. I believe I would radically change my approach if the students could use the computer during class time rather than just me using the computer at the front of the room. However, these types of sessions would demand much more preparation on my part.

How much you will emphasize computer assignments, applications, and theory depends on your personal situation. Your teaching style, your course objectives, and the specific needs at your school are all factors that must be given consideration. The book [2] greatly helped me navigate these issues for myself. The ATLAST web page has useful information such as sample lesson plans. The address for this web page can be found below in Section 4.

DESIGN COMPUTER ASSIGNMENTS

The first time you use computer assignments, you should probably proceed with some caution. Evaluate the effectiveness and difficulty of each assignment before making another. Various details can require more attention than you might expect, especially at the beginning of the semester. For instance, you may find that access to the computer labs is inadequate, or a student who buys software has trouble installing it. Equipment has a way of breaking down when you need it most. It would be wise to assume "If it can go wrong, it will" and then be pleasantly surprised if things go smoothly.

It is important to make an easy computer assignment early and collect it, to get students started and to help you evaluate how they react to computer use. I suggest you ask them to work through "Getting Started" during the first few days. It will be good for them to know what topics are discussed there, even if they don't understand all the matrix operations at first. Then you could demonstrate the functions **replace**, **swap** and **scale** a little, and assign "Practice Row Reduction." After they succeed with that project, you might assign some [M] exercises from the first few sections of the text.

Work each computer problem yourself before assigning it. You will then know how this experience will fit with your classroom lessons, what students should watch out for, how much time to allow, and how much other homework is reasonable to assign at the same time. Most of the projects are straightforward – students have to read some, and spend some time doing the calculations, but hard thinking is required only occasionally. Emphasize that the questions that ask for *interpretation of what they calculate* are the ones that really matter, and that's where most of the points are.

In recent semesters, I have used 10 core projects, and then assigned a few more specific to the students' needs and interests. For example, the education majors were assigned the Cryptography project while those interested in business were given the project "An Economy with an Open Sector." Jane Day required about 14 projects and let students select 2 or 3 more for extra credit, which seems a reasonable way to address the variety of interests her students had.

It is a good idea to discuss each project briefly before assigning it and again when handing the papers back, to be sure everyone got the point. A very real danger in computer projects is that students push the buttons and miss the obvious points in understanding. The projects here are written like lab forms and from my experience are pretty easy to grade. Student graders, if available, can be used to grade the projects to save you time.

I encourage students to work in groups. I am almost convinced that students do better with technology when they have to figure it out for themselves rather than when it is explained to them. Furthermore, there are the MATLAB boxes and the MATLAB Appendix in the *Study Guide* for them to use as a resource. Since it is unlikely that someone will always be there to provide an explanation as newer technologies and upgrades are introduced, figuring out technology on your own seems to be an important skill to try to develop. Working in groups eases this struggle, but I still am available for consultation—particularly for the shy student who has some aversion to computers.

Sometimes students use MATLAB to check their answers to the textbook problems, but most of the time my students just use their calculators. Since one of the emphases in my class is theory, I collect homework that requires explanations or proof rather than routine problems. However, the routine problems and computer projects help build that understanding so that the theory is not developed in a "vacuum."

ANTICIPATE COMMON DIFFICULTIES

Most students today are very computer literate. However there are still a few techno-phobes in every class. Their problems are primarily not with MATLAB, but rather with the interface between it and the outside environment, such as saving, editing and printing files. Jane Day designed these projects to minimize these issues because she decided that these activities do not enhance linear algebra much. Her students would record results by hand and print a graph occasionally. The projects are much easier to grade than they used to be, when students used the **diary** command all the time and turned in huge stacks of paper. There are only two projects now that suggest students create a diary file.

If, for example, you prefer to have your students turn in their projects electronically either by email or by a web course management system, you will have to think through how you want your students to submit their solutions. Editing documents created through the **diary** command is much easier today due to the interactive capabilities of most computers. Still, your students will need guidelines so that the problems that arise are related to the material and not the word processing.

CONSIDER CLASSROOM DEMONSTRATIONS

It seems that many schools are moving towards permanent or portable demonstration units in the classroom. Such units would typically have a computer and a multimedia projector. The portable ones have the inconvenience of setting them up and down before class, but they do work. For linear algebra purposes, a color display is desirable because of the demonstrations related to graphics.

While it is very difficult at our institution to reserve a computer lab on a regular basis, I can use such a projection unit with a computer and a camera on a calculator for classroom demonstrations. This has benefited my classes so much that I resent being assigned a classroom with just a chalkboard for my other courses. If I understand the educational research correctly, students learn much more by seeing the material in a variety of contexts.

One way classroom demonstrations can be used is to check on student learning. For example, if the students appear to understand a particular problem and its expected numerical results, students can witness an instant calculation to see if that corresponds to their understanding. The class can also make conjectures, and those claims can be put to the test rather quickly using technology. An aid to this end is the collection of special MATLAB functions that comes with the Laydata Toolbox. See the *Study Guide* and Section 3 on page 8 below for further descriptions of the special functions.

If the computer is used appropriately, the material becomes more convincing and richer. The likelihood of getting bogged down in the arithmetic increases the chances that students miss the larger, fuller picture. Using MATLAB allows me to focus on the concepts and ideas that I want to stress. For example, I can ask what should be calculated to verify that the answer is correct, and then use MATLAB to do that calculation. This process can be a much better use of class time for helping students grasp the material rather than working out a tedious solution by hand. This process of *analysis*, *prediction*, *computer solution*, and *verification* is how professional scientists use computers, and these are important skills for students to practice.

DECIDE HOW YOU WILL TEST STUDENTS

How students should be assessed on exams is always a critical issue for an instructor. One major consideration is whether to allow MATLAB on exams. If MATLAB is unavailable, you will need to decide if graphing calculators will be permitted. You obviously will have to consider the profiles of your students and the test conditions at your school.

My approach is that students should primarily be tested on conceptual ideas rather than computations. There are other settings, such as homework, where computational skills can be assessed. I allow TI-85 and TI-86 graphing calculators on tests because the use of MATLAB for me is not practical. I still try to minimize the need for a calculator unless I give an untimed test or I link the data directly to their calculators.

As far as the type of questions to include on an exam, I like test questions that can be quickly done if one takes the appropriate perspective and thinks about the issues involved. I also value questions that require the synthesis of a variety of ideas and topics. When working on your exams, consider the text website which has many sample tests and review sheets for three different types of courses. You might choose to tell your students which types of questions on those exams are similar to the ones you sometimes create.

Consider including on exams a question or two based on projects that the students have completed. Such questions can be effective in reinforcing the objectives of the assignment and in determining which members of the group actually participated in the project. The following sample questions are examples where computers were used during instruction but not on the test.

1. The following matrices are row equivalent: $A = \begin{bmatrix} 1 & 2 & -3 & -2 \\ -1 & -2 & 0 & 8 \\ 2 & 4 & -5 & -6 \end{bmatrix}$, $R = \begin{bmatrix} 1 & 2 & 0 & -8 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 0 \end{bmatrix}$.

Write the general solution to $A\mathbf{x} = \mathbf{0}$. Write a particular solution to $A\mathbf{x} = \mathbf{0}$. Consider the matrix transformation $\mathbf{x} \rightarrow A\mathbf{x}$; is it 1-1? onto? Explain answers. Find a basis for the column space of A and a basis for the null space of A.

2. There is a real 3x3 matrix A for which the general solution of some system $A\mathbf{x} = \mathbf{b}$ is $\mathbf{x} = x_3 \begin{bmatrix} -3 \\ 4 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix}$.

What is the general solution of $A\mathbf{x} = \mathbf{0}$?

3. A certain population of owls feeds almost exclusively on wood rats. Letting $o(k)$ and $r(k)$ denote the number in each population in year k, a biologist estimates that $o(k+1) = .5\ o(k) + .05\ r(k)$ and $r(k+1) = -.9\ o(k) + 50\ r(k)$. Write the matrix that describes the interaction of these two populations from year k to year $k+1$.

Assume the pattern described will continue in the future. Don't calculate, but instead answer in words:

(a) What would you calculate, and how would you interpret the results, to find out the number of individuals in each population five years from now?
(b) How could you use eigenvalues and eigenvectors to describe the long term behavior of the owl and rat populations? Include any equations you need to discuss, and say what all your symbols mean.

4. Consider the vectors $\mathbf{v}_1 = (1,1,1,1)$, $\mathbf{v}_2 = (2,1,0,-3)$, and $\mathbf{v}_3 = (-1,2,0,0)$.

(a) Which pairs of these vectors are orthogonal to each other? Show work.
(b) Write the formulas for additional calculations which could be done to get an orthonornal basis for the subspace spanned by $\{ \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \}$. If you have time, complete the calculations for extra credit.

5. Let A be the nxn matrix in which each entry is 1. Justify your answers to the following questions. For (a)-(c), think, don't calculate! A very little calculation will be needed for (d).
(a) There are two distinct eigenvalues of A. What are they?
(b) What is dim(Nul(A))?
(c) What is the characteristic polynomial of A?
(d) What is a basis for each eigenspace of A?

6. Suppose A is an invertible matrix, and $\mathbf{x}$ is an eigenvector of A.

(a) Which of the following matrices also have $\mathbf{x}$ as an eigenvector? Circle the ones that do:
A^{-1} A^{T} $2A$ A^{3} $A+A^{3}$

(b) Choose one of those you circled in part (a) and justify it. For instance, if you circled $2A$, you must show that if $\mathbf{x}$ is an eigenvector of A then it is also an eigenvector of $2A$.

7. Let A be an nxn matrix. In each part below, circle the one best possible expression to complete the sentence truthfully. Only that one best choice will be counted correct:

(a) Suppose A has four distinct eigenvalues. Then A (will) (will not) (could but doesn't have to) have four independent eigenvectors.

(b) If A has zero as an eigenvalue, the eigenspace of zero (will) (will not) (could but doesn't have to) equal Nul(A).

(c) Suppose A^{2} is the nxn zero matrix. Then A (will) (will not) (could but doesn't have to) be the zero matrix.

BE CREATIVE

The very existence of powerful and accessible matrix utilities raises questions about what topics to emphasize, what skills students need to learn, and what style of teaching is best. These issues are not easily resolved. Many have been influenced by the constructivist theory that students learn best when making connections to what they already know or what they want to know. As mentioned before, I have used certain projects for certain majors and had them expand on those ideas by having them develop an education lesson plan or by writing a report after doing some independent reading. Each year I try to use more group work in my courses so that students feel more involved in the course. I encourage my students to study and to work on problems together, and I think working together lowers the frustration level with regards to computers.

I encourage you to discuss what is happening in other departments and disciplines at your institution. Also make use of instructors at other schools. My discussions with other colleagues have allowed me to understand better what other faculty are trying to accomplish and to be exposed to the educational issues and curriculum developments in other disciplines. Linear algebra has many uses, and you might gain some interesting and motivating problems from others at your school or from your community as a service-learning project. At the same time, communicate what you are doing to allow them to appreciate your subject matter.

3. Using Software for Demonstrations

Most of the M-files in the Laydata Toolbox simply contain data for exercises and projects, and this can be helpful for demonstrations as discussed on page 5. In addition, there are a few files that are called the "special functions," which do particular kinds of calculations or graphing. They were written for various exercises and projects but they can also be effective for occasional demonstrations. I encourage you to experiment with them early so you'll know about them when an occasion arises where they might be helpful. Here is a list of these special functions with a few comments:

Special Function	Description
replace, swap, scale	Single row operations
gauss, bgauss	Sweep out specified columns
nulbasis	Produces a basis for the null space
proj	Orthogonal projection of vector onto a subspace
gs	Performs Gram-Schmidt algorithm
qrbasic	Basic QR method for calculating eigenvalues
qrshift	QR method with shifts and deflation

The five functions below produce simple but effective graphics.

seesum, seeprod, seecom	Visualize vector arithmetic
drawpoly	Draw polygons
singvec	Search visually for singular vectors

The function **randomint** allows you to specify size and rank, and is very useful for generating quick, clear examples.

randomint	Create random integer matrices
randomstoc	Creates a stochastic matrix with random nonnegative entries

The simple way to find out how to use any MATLAB function is with **help**. For example, at the MATLAB prompt, type **help replace** or **help seesum**.

You also may want to investigate the many M-files that accompany the ATLAST book [6]. These are available free from the ATLAST Web page, as described in Section 4 below, and some make excellent classroom demonstration. However, if your students make regular use of them, you should ask them to buy the book. Some favorites are **powplot**, which plots the image of the unit circle under powers of a 2x2 matrix, **cogame**, a coordinate guessing game, and **eigshow** and **svdshow**, which let you search geometrically for eigenvalues and singular vectors. (The special function **singvec** mentioned above does the same thing as **svdshow** but **singvec** has less sophisticated graphics, is free, and runs in any version of MATLAB whereas **svdshow** may not.)

Although all these special functions are very nice, at some point emphasize that they were developed for educational purposes and should not be used for professional applications. If such a need arises, they should use professionally written software that employs the most sophisticated and efficient algorithms known. For example, in MATLAB one should use MATLAB's backslash to solve linear systems, not **ref**. The latter does not check for condition number, and its algorithm is not the most efficient. Usually it produces accurate answers, but when a matrix is nearly singular, **ref** can return the wrong reduced form, and the user will not be warned. See the project "Roundoff Error in Matrix Calculations" for more details. Similarly, one should use MATLAB's **qr** function, not the special function **gs** in any professional setting where an orthonormal basis is needed. The function **gs** was written to help students learn the Gram-Schmidt algorithm, and it works fine on a small set of vectors.

4. Downloading M-files from the Web

To get the Laydata Toolbox files from the Addison-Wesley web page, use Netscape or another Web browser and type the following address.

http://www.laylinalgebra.com

Follow the on-screen directions to obtain the version of Lay's files you want.

Once you have the files, you will want to decompress them and make them accessible to the working path so that MATLAB knows where to find them. To avoid having to type a possibly complicated path to the correct folder, create an empty folder name laydata inside the main MATLAB folder. For example, on a PC the file should be created in c:\matlab\ (or whatever your working MATLAB path is). With the folder already named, navigation is easier as you move through the directory tree and decompress the downloaded files into the appropriate folder. On a PC, you can use a program such as Winzip to decompress and extract the files. For Macintosh computers, Stuffit is the standard decompression program. At my school, unzip is the utility of choice for Linux and Unix.

If your access to MATLAB is through a network, ask your network administrator to install the Laydata Toolbox and any other scripts or M-files you need so that they are accessible.

If the Laydata toolbox or other M-files are saved someplace else, you can use the Set Path feature in MATLAB.

- In MATLAB 6, select **Set Path** from the **File** menu. Click on **Add with Subfolders** and find the folder name of the Laydata subdirectory. Highlight the folder and click on OK. Finally, click on **Save** and **Close**.
- In MATLAB 5, select **Set Path** from the **File** menu. Use **Browse** to find the Laydata toolbox, and click OK. Then under the **Path** menu, select **Add to Path**. Finally, choose **Save Path** from the **File** menu of the Path Browser.
- In MATLAB 4, you need to edit and save the file **matlabrc.m**. (Use your computer's Search capabilities to find the file.) Part of the way down this file you will find a section which starts "MATLABPATH = ". Add the Laydata directory to the list. See Section 15 of the preliminary project for an example.
- If you are running MATLAB on a network, you should ask the system administrator to store the Laydata toolbox to MATLAB's path.

For more details, see Section 15 of the preliminary Computer Project, "Getting Started with MATLAB" and the MATLAB appendix in the *Study Guide.*

Another recommend source of Linear Algebra materials and M-files can be found at the ATLAST site:

http://www.umassd.edu/SpecialPrograms/Atlast

As with the Laydata Toolbox, you will need to decompress the ATLAST M-files and make them accessible to MATLAB's working path.

5. Computer Projects

GENERAL INFORMATION

The projects are intended to enrich and expand the material in Lay's text. They are independent of each other. Each one begins by stating a purpose, the prerequisite sections from text, and the MATLAB functions used. On the line listing the MATLAB functions, the commands inherent to MATLAB are listed first and are followed by a semicolon. The functions and data files from the Laydata Toolbox follow the semicolon. You may copy and use the projects as written, or adapt them. Most projects should require 1-2 hours, and a few may take longer. They do not require very hard thinking (except for an occasional extra credit question, and the project "Subspaces"). The MATLAB commands are given, so the time required depends mostly on how much independent reading students must do. Their work will go faster if you lecture a little on the material before they begin a project, but any of these can be "read and do" assignments.

I allow about a week for each project. You may want to be lenient with your deadlines as equipment misbehaves, networks go down, and students have more difficulty with a project than you expected.

PARTNERS

I encourage but do not require students to do their computer work with a partner. This helps the students work through some of the computer issues together and cuts down on computer frustration. It also reduces my grading. Most of my classes do well in pairing up, but some students need help finding a partner even after the first couple of weeks.

Before assigning the lab it is a good idea to present some ground rules. For example, both people should work on the lab and understand the solutions. Their signature on their paper indicates that both did the work and that both agree to the work submitted. I suggest you follow up the computer assignment after it has been submitted to confirm the signatures and to discuss briefly the objectives. One colleague of mine picks one group to present their solution to the rest of the class.

NOTES ABOUT THE INDIVIDUAL PROJECTS

Here are a few comments about each project. The symbol **R** means the project is especially recommended because of its value. I usually grade each project out of 10 points and am generally more lenient with my grading than on other homework. Jane Day generally uses a 5 point system and her point assignments are listed here.

Getting Started With MATLAB. This is long and it is not necessary to do it all but it can be helpful for novices, and for reference.

R **Practice Row Reduction** (5). This is easy and could be assigned soon after students have learned to do row operations by hand. They will practice doing them with Lay's functions **replace**, **scale**, and **swap**.

Exchange Economy and Homogeneous Systems (5). Students seem to like economic models. Assign this one immediately after covering homogeneous systems in Section 1.6, and perhaps letting them read about Leontief models by themselves. The last two questions provoke them to think about row operations abstractly. For the extra credit question, Jane Day gives one point if a student works out a symbolic example with three rows and columns say, but no points if they give only numerical examples. A nice general argument is worth 2-3 points.

Reduced Echelon Form and ref (5). This is easy and shows students how to row-reduce a matrix using **gauss** and **ref**. They also see that roundoff error can cause **ref** to produce the wrong answer, by experimenting with different values for the tolerance in **ref**. Although the command **ref** can be introduced as early as Section 1.2, the text emphasizes echelon form rather than reduced echelon form until Section 1.5. The *Study Guide* uses **gauss**, **swap**, and **scale** for row operations until Section 4.3 (and 2.9).

Rank and Linear Independence (5). This project can be used to introduce rank earlier than the text does. The last question here is an "explore and conjecture" type. *(Section 1.7 is the prerequisite section.)*

R **Visualizing Linear Transformations of the Plane** (5). This looks long but much of it goes quickly. It uses **drawpoly** for some graphics, and helps students start thinking about a matrix as a transformation early. Most students seem to have very little geometric intuition and need all the practice they can get to develop some. *(Section 1.9)*

Population Migration (5). Students like this, especially the plotting. Before assigning this one and going over the city-suburb example in the text, you might ask your students what will happen if this pattern of migration persists. Will everyone move to the suburbs? Have them calculate $\mathbf{x}_k$ for some large values of k and report back at the next class. *(Section 1.10)*

R **Elementary Analysis of the Spotted Owl Population** (5). This one does not need as much introduction, but students should read the simple example at the start of Chapter 5. Mention that they will do a naive analysis of the population's long term behavior in this project and later will use eigenvalues and vectors to analyze it more deeply in "Using Eigenvalues to Study Spotted Owls." *(Section 1.10)*

Lower Triangular Matrices (5). This is a simple but nice exploration of matrix multiplication. Students will discover that the product of (unit) lower triangular matrices is (unit) lower triangular, and write proofs. *(Section 2.1)*

The Adjacency Matrix of a Graph (10). This is a more sophisticated look at matrix multiplication. Students examine very carefully how an entry of A^2 is calculated. They must also create a definition, which is a new kind of exercise for most of them. Before assigning this project, I recommend you discuss graphs a little and explain "contact level k."

The answers for question 4(b) are "All but W8" and "All." According to Jane Day, a good answer to 4(d) would be "Dangerous means highest level one contact, and W6, W4 and W1 are most dangerous." Occasionally an observant student of hers would see that W6 is in level two contact with everyone else, and is the only worker like that. She gives 2 extra credit points for that answer, and report their insight to the class when handing the papers back. It never fails to cause a stir, it motivates the others to pay more attention to details. Students' definitions in 4(d) are often vague, like "high contact level" and she usually takes off 1-2 points but tries to help them say what they really wanted the definition to be, based on their explanations at the end. *(Section 2.1)*

It is recommended that you use both the next project and **"Roundoff Error in Matrix Calculations"** if possible. Although neither is essential to the flow of ideas of the text, the computational realities they illustrate are important things for these students to know since many of them will do matrix calculations in scientific applications.

R **Counting Floating Point Operations in Matrix Products** (5). This project is especially easy and can be done any time after Section 2.1. It shows that the number of flops to calculate $A(BC)$ and $(AB)C$ can be dramatically different. Note: the command **flops** is available in MATLAB 4 or 5, but not in version 6.

R **Cryptography** (5-10). Many students like this one a lot---particularly the education majors. They use MATLAB's remainder function **rem** to do some arithmetic modulo 26. In the extra credit question they calculate by hand a matrix inverse modulo 26. This project was originally created by a student, Sanja Petrovic. *(Section 2.2)*

Using Backslash to Solve $A\mathbf{x}=\mathbf{b}$ (5). The purpose is to see why the backslash operator is preferable to solving matrix equations when A is invertible. Students compare solutions of equations involving the ill-conditioned Hilbert matrices using the backslash command, the matrix inverse command **inv**, and the **ref** command. *(Section 2.3)*

R **Roundoff Error in Matrix Calculations** (5). Students use backslash, **ref** and **inv** to solve 8 linear systems three different ways. They see that the different algorithms give somewhat different answers in every case, and very different answers for the poorly conditioned systems. They see definitions of floating point notation, residual vector, condition, and Hilbert matrix; learn to watch for warnings; and use **norm**. There are brief discussions of condition number and the algorithms used in the three methods. *(Section 2.3)*

Partitioned Matrices (5). This is an important topic, as partitioned matrices are used frequently in applications. Question 2 is designed to reinforce the fact that $[A \quad B]\begin{bmatrix} X \\ Y \end{bmatrix}$ equals $AX + BY$, not $XA + YB$. Invariably, one or two people make that mistake. What leads them astray is the definition of $A\mathbf{x}$ earlier in the text, where the scalars x_i are written on the left of the columns. To try to forestall this common error with partitioned matrices, when defining $A\mathbf{x}$, point out that the x_i's are scalars so they look more natural on the left of a vector, but technically a scalar could be written on the left or right. However, when we learn to multiply two matrices, we'll see that the order makes a big difference, and one has to be especially careful when multiplying partitioned matrices. *(Section 2.4)*

Schur Complement (5). This is a nice application of partitioning. Students learn three ways to calculate a Schur complement, including using row operations. The extra credit question is challenging. *(Section 2.4)*

LU Factorization (5). This explains how the LU factorization algorithm in Section 2.5 differs from MATLAB's **lu** function, and provides practice using both.

An Economy With An Open Sector (5). Students will verify Theorem 11 in Section 2.6 and then experiment to see that the result can fail if they change just one entry of the consumption matrix C enough. They will probably discover that increasing c_{11} to about .95 will cause the solution of $\mathbf{x} = C\mathbf{x} + \mathbf{d_1}$ to have some negative entries. They should give something like the following explanation: if the Chemicals sector consumes .95 of its own output then it is not surprising that the economy cannot meet the demands from other sectors, and this is what the nonsense solution says.

Matrix Inverses and Infinite Series (5). This explores the meaning of Theorem 13 in Section 2.6. Students will experiment with the series $I + S + S^2 + \ldots$, finding matrices S for which it does and does not seem to converge. They could do this project any time after Section 2.2 as a "read and do" assignment.

Homogeneous Coordinates for Computer Graphics (5). This uses **drawpoly**. Homogeneous coordinates for R^2 and how they can be manipulated with 3x3 matrices are novel ideas to most students. Computer science majors especially like this. *(Section 2.7)*

Subspaces (20). Span is a hard concept for many students, and this has proved to be the most challenging project. You can pair up the students and assign each group a different pair of matrices, A and B; A is 5x4, B is 5x5, and both have rank 4. There are 25 pairs of integer matrices in the file **submats** . The first 17 pairs do have the same column space, and the last 8 pairs do not. If you prefer to generate your own matrices, here are commands create a pair that do have the same 4-dimensional column spaces:

A = randomint(5,5,4), B = A*randomint(5,4,4)

If you want A and B that do not have the same column spaces, try this instead:

A = randomint(5,5,4), B = randomint(5,4,4)

(The command **randomint(m,n,k)** yields an mxn matrix with rank k.)

Students quickly reduce A and B and see they have the same rank. They don't feel very comfortable with sets, but after they discuss the problem with me and each other for a while, usually they figure out that they can row-reduce $[A\ B]$ and then explain in words why the result shows that each column of B is in Col A, hence Col B is a subset of Col A. An elementary way to finish would be to reduce $[B\ A]$ to see that Col A is also a subset of Col B, and then conclude that the two sets must be the same, but few students seem to think of doing that. Instead they explain that Col B is inside Col A and has the same dimension so by Theorem 15, a basis for Col B must span Col A.

This project could be assigned after Section 4.6 (or Section 2.9 if you cover that instead).

Markov Chains and Long-Range Predictions (5-10 points, depending on how much help is given ahead of time). This is fun and good motivation for eigenvalues and eigenvectors. If you ask everyone to do this project, you should first lecture on Markov processes a little. *(Section 4.9)*

R **Real and Complex Eigenvalues** (5). This is easy and you could assign it instead of lecturing on complex eigenvalues. Students calculate some complex eigenvalues by hand and then create some examples of their own, to be sure they really look at the matrix entries. Then they learn how to use MATLAB's **eig** function to find eigenvalues and eigenvectors. *(Section 5.5)*

R **Using Eigenvalues to Study Spotted Owls** (10). Before assigning this, either lecture on complex eigenvalues or assign the previous project. This is a long project but students like it. It is a lovely application of eigenvalues and diagonalizability, and includes some plotting. Most of the theory from Sections 5.1-5.3 is applied. Students use **eig** and experiment to find the critical value of t, the survival rate for juvenile→subadult ("critical" means the minimum value of t which makes the dominant eigenvalue at least 1).

The extra credit question asks users to verify theoretically what they have seen experimentally. It is challenging but many hints are given. The idea for this question is due to Andre Weideman and used with his permission. Techniques from calculus can be used to show that the stage matrix will always have its dominant eigenvalue real and positive and be diagonalizable, and then one can derive a formula for the critical value of t. *(Sections 5.5 and 5.6)*

QR Factorization (5). This is not hard and should be of interest to many students, since QR factorizations are widely used in practice. Students will learn how MATLAB's **qr** function differs from the QR factorization developed in the text and also will verify the connection beween QR factorization and the Gram Schmidt Process. *(Section 6.4)*

The QR Method for Calculating Eigenvalues (10). Some students will be very interested in this. Here they use the **qr** function to experiment with the basic QR algorithm for eigenvalues and with a shift-deflate version of it. This type of iterative process is used in modern software for calculating eigenvalues. It never fails to amaze that the processes usually work! Convergence is discussed briefly and a reference given for more information. Two functions in the Laydata Toolbox, **qrbasic** and **qrshift**, assist with the calculations. *(Sections 5.2 and 6.4)*

R **Least-Squares Solutions and Curve Fitting** (5). This is easy and can be done early in Chapter 6 to motivate the ideas. The text algorithm is used to calculate coefficients for the least-squares line, quadratic and cubic curves; **norm** is used to calculate least-squares error, and **plot** is used to graph the data and the curves. An end note describes how to use **polyval** and **polyfit**, which are MATLAB functions that can do most of the work for you.

In question 2(b), most students guess $(vel)^3$ and say the error is smaller for the cubic than for the line or quadratic, and that is acceptable to some including me. However, the correct answer is that drag depends on $(vel)^2$. Consider giving an extra credit point to anyone who sees that this is sensible guess since the quadratic curve is a dramatically better fit to the data than the line, but the cubic gives very little improvement. These things are evident both from the graph and the errors. In fact, the theoretical formula is $drag = \rho c(vel)^2 A$ where ρ is air density, c is the coefficient of drag, which varies with the angle of attack, and A is the surface area of the wing.

6. Overview of the Case Studies and Application Projects

The following case studies and application projects are available from the website which accompanies the text. An icon in the text refers the reader to these resources. The case studies amplify the opening vignette of each chapter and provide exercises based upon the topic mentioned in the vignette. The application projects highlight applications of linear algebra and direct students through a sequence of exercises on that application. Many of these resources use real world data. This data, which has been formatted for MATLAB, may be downloaded to accompany the case study or application project. Solutions for the exercises are also available from the website. These resources have been class tested and are an excellent source of out-of-class assignments.

CASE STUDIES

Chapter 1: Linear Models in Economics This case study examines Leontief's "exchange model" and shows how systems of linear equations can model an economy. Real economic data is used.

Chapter 2: Computer Graphics in Automotive Design This case study explores how a three-dimensional image is rendered effectively in two dimensions. Perspective projections, rotations, and zooming are discussed and applied to wireframe data derived from a 1983 Toyota Corolla.

Chapter 3: Determinants in Analytic Geometry This case study examines how determinants may be used to find the equations for lines, circles, conic sections, planes, spheres, and quadric surfaces.

Chapter 4: Space Flight and Control Systems This case study studies a mathematical model for engineering control systems. The notion of rank is used to determine whether a system is controllable, and a system of equations is solved to determine which inputs into the system would yield a desired output.

Chapter 5: Dynamical Systems and Spotted Owls This case study examines how eigenvalues and eigenvectors can be used to study the change in a population over time. Real data from populations of spotted owls, blue whales, and plants (speckled alders) is studied, and the notion of a sustainable harvest is introduced.

Chapter 6: Least-Squares Solutions This case study uses the method of least-squares to fit linear, polynomial, and sinusoidal curves to real data. This data includes performance in the Olympic men's 400-meter run, climatic data from Charlotte NC, and tidal data from the Cape Hatteras pier.

Chapter 7: The Singular Value Decomposition and Image Processing This case study examines how a singular value decomposition of a matrix may be used to reduce the amount of data needed to store a reasonable image of a graphical object. Two types of images are studied: three dimensional surfaces and black-and-white two-dimensional pictures.

APPLICATION PROJECTS

Section 1.2: Interpolating Polynomials This set of exercises shows how a system of linear equations may be used to fit a polynomial through a set of data points. Polynomial curves are used to fit real data taken from *Car and Driver* magazine.

Section 1.2: Splines This set of exercises shows how a system of linear equations may be used to fit a piecewise-polynomial curve through a set of data points. Cubic splines are used to fit real data taken from *Car and Driver* magazine.

Section 1.10: Diet Problems This set of exercises provides examples of vector equations that result from balancing nutrients in a diet. Real data from the UDSA website is used.

Section 1.10: Traffic Flow Problems This set of exercises shows how system of linear equations may be used to model the flow of traffic through a network. Real data from the Seattle Transportation Management Division and the Charlotte-Mecklenburg Utilities Department is used in this exploration.

Section 1.10: Loop Currents This set of exercises provides further examples of loop currents, and reinforces the text's development of this topic.

Section 2.1: Adjacency Matrices This set of exercises studies the adjacency matrix of a graph. The real route maps of various airlines help to motivate graphical questions which may be answered with adjacency matrices.

Section 2.1: Dominance Matrices This set of exercises applies matrices to questions concerning competition between individuals and groups. The problem of ordering teams within a football conference is discussed, and real data from various football conferences is used.

Section 2.1: Other Matrix Products This set of exercises introduces and explores the properties of two matrix products: the Jordan product and the commutator product.

Section 2.3: Condition Numbers This set of exercises motivates the definition of the condition number of a matrix, and explores how its value affects the accuracy of solutions to a system of linear equations.

Section 2.5: The LU and QR Factorizations: This set of exercises shows how to use an LU factorization to perform a QR factorization. The QR factorization is introduced in Exercise 24 of this section.

Section 2.5: Equilibrium Temperature Distributions This set of exercises discusses the problem of determining the equilibrium temperature of a thin plate. An appropriate system of equations is derived, and is solved both by finding a matrix inverse and by an LU factorization.

Section 2.6: The Leontief Input-Output Model This set of exercises provides three real data examples of the Leontief input-output model discussed in the text. American economic data from the 1940's and the 1990's is studied.

Section 3.3: The Jacobian and Change of Variables This set of exercises is designed for students who have experienced multivariate calculus. The Jacobian is derived and applied to a change of variables in double and triple integrals.

Section 4.1: Hill Substitution Ciphers This set of exercises studies how matrices may be used to encode and decode messages. Matrix arithmetic modulo 26 is used.

Section 4.6: Error-Detecting and Error-Correcting Codes This set of exercises studies how to construct methods for detecting and correcting errors made in the transmission of encoded messages. The United States Postal Service bar code is studied as an error-detecting code, and the error-correcting Hamming (7,4) code is also studied.

Section 5.3: The Fibonacci Sequence and Generalization This set of exercises introduces the Fibonacci sequence and Lucas sequences. Eigenvalues, eigenvectors, and diagonalization are used to derive general formulas for an arbitrary element in these sequences.

Section 5.4: Integration by Parts This set of exercises shows how the matrix of a linear transformation relative to a cleverly chosen basis may be used to find antiderivatives usually found using integration by parts.

Section 6.4: The QR Method for Finding Eigenvalues This set of exercises shows how the QR factorization of a matrix may be used to calculate its eigenvalues. Two methods for performing this action are considered and compared.

Section 6.4: Finding Roots of Polynomials with Eigenvalues This set of exercises describes how the real roots of a polynomial can be found by finding the eigenvalues of its companion matrix. The QR method is then employed to find these eigenvalues.

Section 7.2: Conic Sections and Quadric Surfaces This set of exercises shows how quadratic forms and the Principal Axes Theorem may be used to classify conic sections and quadric surfaces.

Section 7.2: Extrema for Functions of Several Variables This set of exercises is designed for students who have experienced multivariate calculus. Quadratic forms are used to investigate maximum and minimum values of functions of several variables. Results are derived in terms of the eigenvalues of the Hessian matrix.

7. References

[1] D. Carlson, C.R. Johnson, D.C. Lay, A.D. Porter, "The Linear Algebra Curriculum Study Group Recommendations for the First Course in Linear Algebra," *College Math. Journal* (24), 1993, 41-46.

[2] D. Carlson, C.R. Johnson, D.C. Lay, A.D. Porter, A. Watkins, W. Watkins, eds., *Resources for Teaching LinearAlgebra*, MAA Notes, Mathematical Association of America, 1997.

[3] D. Carlson, C.R. Johnson, D.C. Lay, A.D. Porter, eds., *Linear Algebra Gems: Assets for Undergraduate Mathematics,* MAA Notes, Mathematical Association of America, 2002.

[4] George E. Forsythe, "Pitfalls in Computation, or Why a Math Book Isn't Enough," *American Mathematical Monthly*, Nov. 1970.

[5] Marc E. Herniter, *Programming in MATLAB®,* Pacific Grove, CA: Brooks/Cole, 2001.

[6] Steven Leon, Eugene Herman and Richard Faulkenberry, eds., *ATLAST Computer Exercises for Linear Algebra*, Englewood Cliffs: Prentice-Hall, 1996.

[7] R. Pratap, *Getting Started with MATLAB*, Philadelphia: Saunders, 1996.

[8] Kermit Sigmon and Timothy Davis, *MATLAB Primer*, 6th ed., Boca Raton: CRC Press, 2001.

[9] *MATLAB Student Version Release 12,* Natick, Massachusetts: MathWorks, 2002.

[10] David Watkins, *Fundamentals of Matrix Computations*, New York: Wiley, 1992.

MATLAB PROJECTS

to accompany the text

LINEAR ALGEBRA AND ITS APPLICATIONS, 3rd ed., David C. Lay

Title of Project	Prerequisite Sections in Text
Getting Started With MATLAB	None
Practice Row Operations	1.2
Exchange Economy and Homogeneous Systems	1.6
Reduced Echelon Form and **ref**	1.2*
Rank and Linear Independence	1.7
Visualizing Linear Transformations	1.9
Population Migration	1.10
Elementary Analysis of the Spotted Owl Population	1.10
Lower Triangular Matrices	2.1
Adjacency Matrix of a Graph	2.1
Counting Floating Point Operations in Matrix Products	2.1
Cryptography	2.2
Using Backslash to Solve $A\mathbf{x} = \mathbf{b}$	2.3
Roundoff Error in Matrix Calculations	2.3
Partitioned Matrices	2.4
Schur Complements	2.4
LU Factorization	2.5
An Economy With An Open Sector	2.6
Matrix Inverses and Infinite Series	2.6
Homogeneous Coordinates for Computer Graphics	2.7
Subspaces	4.6 or 2.9
Markov Chains and Long Range Predictions	4.9
Real and Complex Eigenvalues	5.5
Using Eigenvalues to Study Spotted Owls	5.5 and 5.6
QR Factorization	6.4
QR Method for Calculating Eigenvalues	5.2 and 6.4
Least-Squares Solutions and Curve Fitting	6.6

**It is suggested that this project not be assigned until after Section 1.5 or 1.6.*

MATLAB Project: Getting Started with MATLAB Name_____

Purpose: To learn to create matrices and use various MATLAB commands. Examples here can be useful for reference later.

MATLAB functions used: **[ ] : ; + - * ^**
size, help, format, eye, zeros, ones, diag, rand, round, cos, sin, plot,
axis, grid, hold, path; and **randomint** and **startdat** from Laydata Toolbox

Introduction. This can be used as a brief tutorial and as a reference for basic operations. Use MATLAB's **help** command or see a User's Guide for more information. Some of the commands discussed here are about linear algebra topics which will not be formally introduced in your course for several weeks, so even if you go through this project early, you may want to refer back to it at various times. Write notes and questions to yourself as you work through it.

Instructions. Start MATLAB by clicking on its icon (or if you have MATLAB 3.5, type **matlab** at the DOS prompt). The MATLAB prompt is a double arrow, >>. In this project each line that begins with >> is a command line, and the bold face words following >> are MATLAB commands. Try each of these for yourself: that is, type the bold face words and then press the key that is labeled "Return" or "Enter," to cause those commands to be executed. (In the first few sections we will write **[Enter]** to mean press that key, but we will omit this "carriage return" prompt later.) After you execute each line, study the result to be sure it is what you expect, and take notes. After trying the examples in each section, do the exercises.

If you do not complete this tutorial in one session, the variables you created will be erased when you exit MATLAB. See the remark before Section 6 to find out how to get them back quickly the next time you continue work on this project.

Sections:

1. Creating matrices.

A *matrix* is a rectangular array, and in linear algebra the entries will usually be numbers. The most direct way to create a matrix is to type the numbers between square brackets, using a space or comma to separate different entries and a semicolon or [Enter] to create row breaks. Examples:

```
>> A = [1 2;3 4;5 -6]      [Enter]
A =
     1  2
     3  4
     5 -6

>> B = [1 -2 3             [Enter]
     4 5 -6]                [Enter]
B =
     1 -2  3
     4  5 -6

>> x = [4;3;2]             [Enter]
x =
     4
     3
     2
```

>> X = [1,2,3] **[Enter]**
X =
1 2 3

To see a matrix you have created, type its name followed by [Enter] . Try each of the following and make notes how the results were displayed. Notice MATLAB is case sensitive -- for example, *x* and *X* are names for different objects:
>> A **[Enter]**

>> A,B **[Enter]**

>> X,x **[Enter]**

MATLAB will not try to execute an assignment until it is complete. For example, if you forget and press [Enter] before typing the right bracket, it will just wait for the bracket. Try this:
A = [1 2;3 4;5 -6 **[Enter]**
] **[Enter]**

Exercise: If you have not done it already, create the matrices *A*, *B*, *x*, and *X* above. Then create *C*, *D*, *E*, and **vec** as shown below. (We write them with square braces as a textbook would, but MATLAB does not display braces.) For each matrix, record what you typed and be sure the MATLAB display is what you expected.

$$C = \begin{bmatrix} -4 & 8 & -1 \\ 5 & 0 & 3 \\ 6 & 2 & 10 \end{bmatrix} \qquad D = \begin{bmatrix} 2 & -1 \\ 1 & 3 \\ -2 & 1 \end{bmatrix} \qquad E = \begin{bmatrix} 2 & -1 \\ 0.1 & 3 \\ -2 & 1 \end{bmatrix} \qquad \mathbf{vec} = \begin{bmatrix} 3 \\ -5 \\ 1 \end{bmatrix}$$

Notice that since one entry in *E* is a decimal, MATLAB displays every entry as a decimal.

2. The arrow keys.

MATLAB keeps about 30 of your most recent command lines in memory and you can "arrow up" to retrieve a copy of any one of those. This can be useful when you want to correct a typing error, or execute a certain command repeatedly. Type the following line and record the error message:

>> Z = [1 2 3 4;5 0] **[Enter]**

Error message:_____

To correct such an error, you could retype the entire line. This is easier: press the up arrow key on your keyboard one time to retrieve that last line typed, use the left arrow key to move the cursor so it is between 2 and 3, type a semicolon and then press [Enter] to cause the new line to execute.

You can also use the right arrow key to move to the right through a line, and if you "arrow up" too far, use the down arrow key to back up. To erase characters, use the BackSpace or Delete keys. It does not matter where the cursor is when you press [Enter] to execute the line.

Exercise. Press the up arrow key several times to find the command line where you defined *E*. Change the 0.1 entry to 0.01 and press [Enter] to execute. Record the new version of *E*:

3. The size command. When *M* is a matrix, the command **size(M)** returns a vector with two entries which are the number of rows and the number of columns in *M*. Example:

```
>> size(A)          [Enter]
ans =
     3  2
```

Notice that **ans** is a temporary name for the last thing you calculated if you did not assign a name for that result.

Exercise. Calculate the size of each of the other matrices you have created, *B*, *X*, **x**, *C*, *D*, *E*, **vec**, *Z*.

4. The help command. The command **help** can provide immediate assistance for any command whose name you know. For example, type **help size**. Also try **help help**.

5. Accessing particular matrix entries. If you want to see a matrix which you have stored, type its name. To correct entries in a stored matrix, you must reassign them with a command. That is, MATLAB does not work like a spreadsheet or text editor – you cannot edit things visible on the screen by darkening them and typing over.

But you can see or change a particular entry, or an entire row, or an entire column, or even a block of entries. Try the following commands to view and change various things in the matrix *C* you created above. In each part type the first command line to see what the matrix and certain entries look like before you change them; then type the second command line to cause a change. Record the result of each command and compare the new version of *C* with the previous version to be sure you understand what happened each time:

a) `>> C, C(3,1)`

`>> C(3,1) = -9`

b) `>> C, C(:, 2)`

`>> C(:, 2) = [1;1;0]`

c) `>> C, C([1 3], [2 3])`

`>> C([1 3], [2 3]) = [-2 4;6 7]`

d) `>> C, C([1 3], :)`

`>> C([1 3], :) = C([3 1], :)`

e) `>> C, C(3, :)`

`>> C(3, :) = [0 1 2]`

Notice the effect of the colon in **C(:, 3)** is to say "take all rows", and its effect in **C(3, :)** and **C([1 3], :)** is to say "take all columns."

We will assume in all the following that you have created the matrices and vectors *A,B,C,D,E,X*,**x**,**vec** above so they exist in your current MATLAB workspace. If you do not complete this tutorial in one session, all variables will be erased when you exit MATLAB. If you continue this tutorial at a new MATLAB session later, you will need to type in whatever variables you need. However, if the M-files in the Laydata Toolbox have been set up for your computer, you can simply type **startdat** to get *A,B,C,D,E,X*,**x**,**vec**. Ask your instructor about these capabilities, or see Sections 15 and 16 in this project for more details.

6. Pasting blocks together. When the sizes allow it, you can create new matrices from ones that already exist in your MATLAB workspace. Using the matrices *B,C,D* created above, try typing each of the following commands and record the result of each command in the space below it. If any error messages appear, think why.

[C D] **[D C]** **[C;B]** **[B;C]** **[B C]**

7. Some special MATLAB functions for creating matrices: eye, zeros, ones, diag . Examples:

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

```
>> zeros(3)
ans =
     0     0     0
     0     0     0
     0     0     0
```

```
>> ones(size(D))
ans =
     1     1
     1     1
     1     1
```

Exercises. Type each of the following commands and record the result:

eye(4) **zeros(3,5)** **zeros(3)** **ones(2,3)** **ones(2)**

diag([4 5 6 7]) **diag([4 5 6 7], -1)** **C, diag(C), diag(diag(C))**

Type commands to create the following matrices. For each, record the command you used:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 7 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & -2 \end{bmatrix}$$

8. Using the colon to create vectors with evenly spaced entries. In linear algebra, a *vector* is an ordered n-tuple; thus one-row and one-column matrices like those we called **x**, *X* and **vec** above would be called vectors. Frequently it is useful to be able to create a vector with evenly spaced entries (for example, in loops, or to create data for plotting). This is easy to do with the colon. Examples:

```
>> v1 = 1:5

v1 =
     1  2  3  4  5
```

```
v2 = 1:0.5:3

v2 =
    1.0  1.5  2.0  2.5  3.0
```

```
v3 = 5:-1:-2

v3 =
     5  4  3  2  1  0  -1  -2
```

Exercises. Use the colon notation to create each of the following vectors. Record the command you used for each:

[-1 0 1 2 3 4]

[9 8 7 6 5 4 3]

[4 3.5 3 2.5 2 1.5 1]

The numbers from 0 to 2, spaced 0.1 apart (record the first few and the last few, and the command you used):

9. Using the semicolon to suppress printing. When you place a semicolon at the end of a command, the command will be executed but whatever it creates will not be displayed on the screen. Examples:

```
>> x = 1:0.2:3;

>> x
x =
    1.0  1.2  1.4  1.6  1.8  2.0  2.2  2.4  2.6  2.8  3.0
```

Exercises. Try these commands and describe the results ("pi" is a constant in MATLAB which approximates π):

a) **>> A;**
>> A

b) **>> w = 0:0.1:pi;**
>> w

10. The format command. This command controls how things look on the screen. Type each of the following commands and record the result carefully. Notice that "e" means exponent -- in fact, it means multiply by some power of 10 -- for example, 1.2345e002 is the number $1.2345(10^2)$.

```
>> R = 123.125

>> format long, R

>> format short e, R

>> format short, R
```

The default mode for display of numbers is **format short**. To restore the default mode at any time, type **format**.

The command **format compact** is very useful. It reduces the number of blank lines on the screen, allowing you to see more of what you have done recently. Try the following and describe the effect of each:

>> A,B

>> format compact, A,B

>> format, A,B

11. Matrix arithmetic. You will soon see the definitions of how to multiply a scalar times a matrix, and how to add matrices of the same shape. MATLAB uses * and + for these operations. Try the following examples, using matrices defined above. You should be able to figure out what the operations are doing. Type each line and record the result; if you get an error message, read it carefully and notice why the command did not work:

>> A, A+A, 2*A

>> A, D, A+D, A-D

>> 2*A - 3*D

>> x, vec, x+vec

>> A, B, A+B

>> x, X, x+X

MATLAB also uses * for multiplication of matrices, which is a somewhat more complicated operation. You will see the definition in Section 2.1 of Lay's text. The definition requires that the "inner dimensions" of the two matrices agree. Without knowing that definition, you can still investigate some of the properties of matrix multiplication (and you may even be able to figure out what the definition is). Type the following lines and record the result of each:

>> A, B

>>A*B

>> B*A

>> B, C, B*C, C*B

>> C, x, C*x

>> X, C, X*C

The symbol ^ means exponent in MATLAB; for example, **Y^2** is a way to calculate Y^2 (which can also be calculated as **Y*Y** of course). Try these:

>> C, C*C, C^2

>> Y = 2*eye(3), Y^2, Y^3

12. Creating matrices with random entries. MATLAB's function **rand** creates numbers between 0 and 1 which look very random. They are not truly random because there is a formula which generates them, but they are very close to being truly random.; such numbers are often called "pseudorandom." Similarly, the function **randomint** in the Laydata Toolbox creates matrices with pseudorandom integer entries.

Type the commands below and describe the result of each. Arrow up to execute the first two lines several times, to see that the numbers change each time **rand** or **randomint** is called.

>> P = rand(2), Q = rand (2,3)

>> format long, P, Q

>> format, P, Q

>> randomint(3)

>> randomint(3,4)

Remarks. You can scale and shift to get random entries from some interval other than (0,1). For example, **6*rand(2)** yields a 2x2 matrix with entries between 0 and 6; and **-3 + 6*rand(2)** yields a 2x2 matrix with entries between -3 and 3. It is also easy to create random integer matrices without **randomint**. For example, the command **round(-4.5 + 9*rand(2))** produces a 2x2 matrix with every entry chosen fairly randomly from the set of integers {-5, -4, .. 4, 5}.

13. Plotting. The **plot** command does 2-D plotting, and when you use it, a graph will appear in a Figure window. If the Figure window obscures most of the Command window and you want to see both windows at once, use the mouse to resize and move them. If you cannot see a particular window at all, pull down the menu Windows and select the one you want. It is not possible to see both windows at once in MATLAB 3.5.

As the examples below show, you can specify a color and a symbol or line type when you use **plot**. To learn more, use **help plot** and the MATLAB boxes in Lay's Study Guide. Try the following examples and make a sketch or write notes to describe what happened each time. Notice we use semicolons when creating the vectors here because each vector is quite long, and there is no reason to look at them:

```
>> x = 0:0.1:2*pi;  si = sin(x);  co = cos(x);
```

```
>> plot(x, si)
```

```
>> plot(x, si,'r')
```

```
>> plot(x, si,'-.')
```

```
>> plot(x, si,'*')
```

```
>> plot(x, si,'b*')
```

Here is one way to get more than one graph on the same axis system. Describe the result of each command:

```
>> plot(x, si, 'r*', x, co, 'b+')
```

```
>> P = [si; co];  plot(x, P)
```

Another way to get different graphs on the same axes is to use the **hold on** command. This causes the current graphics screen to be frozen, so the next plot command draws on the same axis system. The command stays in effect until you release it by typing **hold off**. Try the following commands, and describe the result of each:

>> plot(x, co, 'g--'), hold on

plot(x, si, 'ro')
hold off

It can be helpful to have grid lines displayed, and to set your own limits for the axes. Try the following. (If using MATLAB 3.5, type the **plot** commands last instead of first in each of the following lines -- e.g., **grid, plot(x, si)** .)

>> plot(x, si), grid

>> plot(x, si), axis([-8 8 -2 2])

We defined the vector **x** on the top of page 8. Change the vector **x** and use the last MATLAB command above to get the entire graph from –8 to 8.

14. Creating your own M-files.

General information:

An M-file allows you to place MATLAB commands in a text file so that you do not need to reenter the same information at the MATLAB prompt again and again. It is a good idea to have MATLAB running while you edit an M-file. This will allow you to quickly switch back and forth between the Edit screen and the MATLAB screen so you can try running your file, editing it again, running it again, etc., until it works the way you want.

An M-file must be saved with the extension .m, not .txt or .doc. This extension will be added automatically if you use the File Menu in MATLAB to open a new or existing M-file. Also, you must save an M-file in some directory which is in the MATLAB path, or else MATLAB will not be able to find the file and execute it. For example, on many computers the directory C:\matlab is always in the path. See Section 16 below for some details about these matters.

Using the editor inside MATLAB:

1. Click File on the upper left corner of the MATLAB screen. Choose New if you want to create a new M-File, or if you want to edit one that exists already, choose Open and then browse to find the file you want. Double click on the file name to open it in an Edit window. If this doesn't work inside MATLAB, use a text editor like Notepad as described below.

You can search for an existing file by using Explore or Find. On the ribbon below your Windows screen, right click on Start. Then choose Explore and look in various folders for the file you want, or choose Find, type the name of the file you want (or part of the name), and click Find Now. Once you locate the file, double click on its name to open it for editing.

2. Type the commands you want in the Edit window – any valid MATLAB commands are ok. Then pull down the File menu to "Save As." Henceforth we will write **File|Save As** for this kind of menu/mouse use. In the box labeled "Save In," type the name of the folder where you want to store this file. If unsure, type C:\matlab. In the box labeled "File Name," type a name for your file. For example suppose you type **play** for the file name. Then click "Save." The editor will add the extension .m .

Don't close the Edit window yet; instead click on the MATLAB Command window and type **play** to execute the file. If you want to edit the file more, click on the Notebook window, make changes and save it again. Repeat this procedure until your file works satisfactorily, then close the file by clicking on the box in the upper right corner of the Edit window or by using **File|Exit**.

Using Notepad outside MATLAB:

Notepad is a simple editor that comes with Windows that you will find in C:\Windows. Click on the Notepad icon to open an edit screen. You can type the commands for a new M-file here, or if you want to edit a file that already exists, choose **File|Open,** type the address of the file you want, and press [Enter]. To save a new M-file, pull down the File menu to **Save As** and type the name you want for the file. For example, you could, and type play.m as the name. Notice you must type the extension .m yourself (because Notepad doesn't know to add it).

15. Ways to get Laydata Toolbox and ATLAST M-files

The M-files in Laydata Toolbox can be downloaded from the Lay web site http://www.laylinalgebra.com. Follow the on-screen directions to obtain the version of Lay's files you want.

You will need to have Netscape 4.0 or higher or Internet Explorer 4.0 or higher installed on your computer to access these files. There are versions for various platforms. Click on the version you want, and save the files in a folder on your desktop. They will be in a compressed form. Refer to the README file for information on downloading and decompressing the files.

The ATLAST website at http://www.umassd.edu/SpecialPrograms/Atlast contains resources to facilitate the use of software in linear algebra courses. Some useful M-files can be found under the link "Library of ATLAST M-files for MATLAB."

16. Installing M-files into the MATLAB path

Whenever you want to use M-files that are not part of commercial MATLAB, such as those in the Laydata Toolbox, you must tell MATLAB where to look for them. For example, suppose the Laydata M-files are stored on your C: drive, in a folder called laydata. The following procedures will work for installing any M-files, except the names of the folders may be different.

A. For Macintosh: Drag the icon for the laydata folder into the MATLAB folder on your hard drive. Start MATLAB. From the File menu, select Open. Select one of the M-files in the laydata folder (to open the file as if for editing), then close the file. You are done now – after this, MATLAB will always know to look in that laydata folder.

B. For Windows: Open your Explore window and drag the folder called laydata into your MATLAB folder. Its address is now c:\matlab\laydata . The MATLAB command **path** outputs a long string that contains the addresses of all the folders where MATLAB looks for M-files, and you need to adjoin the address of your new folder to that string. (If you have started MATLAB, you can see the present contents of that string at any time by typing **path**.) Here are some ways to do this, for various versions of MATLAB and various operating systems.

1. If you have Student MATLAB 5.3, start it and then use **File|Set Path** to open a small screen called "Path Browser." Click the Browse button, then locate and click on laydata. Use **Path|Add to Path**. Another small screen will pop up and ask you to confirm this. Click OK. Next use **File|Save Path**. Finally, use **File|Exit Path Browser**. The method is similar in MATLAB 6.

From now on, the new path will be in effect. That is, whenever you use MATLAB, it will look for M-files in c:\matlab\laydata as well as in all the other folders which were in the path originally.

2. If instead you have Student MATLAB 5, start it and click on the Path Browser icon at the top of the screen. (This icon looks like two folders.) This opens a small screen called "MATLAB Path." Click **Add to Path**. Click the button **Browse** (which may be a button with ". . ." on it). Then locate and highlight the folder laydata. Click the button "OK," which returns you to the MATLAB Path window. Use **File|Save Path**, then click the button **Close** to close the MATLAB Path window.

3. If you have Student MATLAB 4, open Windows, then use **Start|Find|Files or Folders** and locate the file matlabrc.m . Double click on its name to open it for editing. Scroll down about 75 lines until you see lines that look like

```
';C:\MATLAB\toolbox\sigsys',...
]);
```

Insert a new line between these two, so the lines now look like

```
';C:\MATLAB\toolbox\sigsys',...
';C:\MATLAB\laydata',...
]);
```

Save the file and then close the Edit screen.

If MATLAB is currently running, type **matlabrc** at this time, to establish the new path for the current session. From now on, whenever you start MATLAB, it will look for M-files in c:\matlab\laydata as well as in all the other folders which were in the path originally.

C. For MATLAB on a network: Ask the system administrator to store your folders and adjoin their addresses to MATLAB's path. The method for doing that will depend on what version of MATLAB the network is running.

D. For MS-DOS: Open for editing the file MATLAB.BAT, which is in the MATLAB\MATLAB subdirectory. Locate the line that begins:

```
SET MATLABPATH = \MATLAB\MATLAB;\MATLAB\DEMO
```

Edit the end of this line, so it looks like:

```
SET MATLABPATH = \MATLAB\MATLAB;\MATLAB\DEMO;\MATLAB\LAYDATA
```

Save and close the file.

E. For any operating system: Strange things happen occasionally on computers. If for some reason the instructions above do not work on your system, the following will always work, but this method has to be repeated each time you start MATLAB. Start MATLAB and type the appropriate one of the following commands:

In MATLAB 6, type **addpath('c:\matlab\laydata')**
In MATLAB 5, type **addpath 'c:\matlab\laydata'**
In MATLAB 4, type **path(path, 'c:\matlab\laydata')**
In MATLAB 3.5, type **matlabpath([matlabpath, 'c:\matlab\laydata'])**

MATLAB Project: Practice Row Operations Name_____

Purpose: To practice calculating the reduced echelon form with individual row operations
Prerequisite: Section 1.2
MATLAB functions used: **-** , **/** ; and **replace**, **swap**, and **scale** from Laydata Toolbox

Background: Read about elementary row operations and reduced echelon form in Section 1.2.

1. (hand) For each matrix, calculate its reduced echelon form by hand. The last three matrices are exercises 9, 10, and 11 from Section 1.2, and it will be beneficial to you to keep a record of what was done in each step. Again, be sure to reduce all the way to reduced echelon form and to show each step:

$$\begin{bmatrix} 0 & 3 & 6 & 9 \\ -1 & 1 & -2 & -1 \end{bmatrix}$$

9. $\begin{bmatrix} 0 & 1 & -6 & 5 \\ 1 & -2 & 7 & -6 \end{bmatrix} \sim$

10. $\begin{bmatrix} 1 & -2 & -1 & 3 \\ 3 & -6 & -2 & 2 \end{bmatrix} \sim$

11. $\begin{bmatrix} 3 & -4 & 2 & 0 \\ -9 & 12 & -6 & 0 \\ -6 & 8 & -4 & 0 \end{bmatrix} \sim$

2. (MATLAB) Now use the functions **swap, replace** and **scale** to do the same row operations on the same matrices as in question 1. One possible solution for the first matrix is shown to illustrate how the functions work. Try the example: type each command line shown, press [Enter], and verify the result. The first line enters the matrix and the second line makes a copy of it. It's a good idea to work on a copy so if you decide to start over, the original matrix is still in your workspace.

M=[0 3 6 9; -1 1 –2 -1]

A = M

$$\begin{matrix} 0 & 3 & 6 & 9 \\ -1 & 1 & -2 & -1 \end{matrix}$$

A = swap(A, 1, 2)

$$\begin{matrix} -1 & 1 & -2 & -1 \\ 0 & 3 & 6 & 9 \end{matrix}$$

A = scale(A, 2, 1/3)

$$\begin{matrix} -1 & 1 & -2 & -1 \\ 0 & 1 & 2 & 3 \end{matrix}$$

A = replace(A, 1, -1, 2)

$$\begin{matrix} -1 & 0 & -4 & -4 \\ 0 & 1 & 2 & 3 \end{matrix}$$

A = scale(A, 1, -1)

$$\begin{matrix} 1 & 0 & 4 & 4 \\ 0 & 1 & 2 & 3 \end{matrix}$$

Now you use these functions to reduce the matrices in exercises 9, 10 and 11. To get the matrices from Lay's Toolbox, type the commands in bold below and press [Enter] after each line.

c1s2 (Chapter 1, section 2)
9 (Problem 9)
A=M (It's a good idea to work on a copy.)

As you reduce each matrix using MATLAB, record each line you type, and the resulting matrix. Attach an extra sheet or use the back. To learn more about the functions, see Lay's Student Study Guide, or type **help swap, help replace, help scale**.

MATLAB Project: Exchange Economy and Homogeneous Systems Name_____

Purpose: To solve a homogeneous system to find equilibrium prices for an exchange model economy.

Prerequisite: Section 1.6

MATLAB functions used: **-**, **/**, **eye**, **sum**; and **econdat** and **ref** from Laydata Toolbox

1. (hand) Let $T = \begin{bmatrix} .20 & .17 & .25 & .20 & .10 \\ .25 & .20 & .10 & .30 & 0 \\ .05 & .20 & .10 & .15 & .10 \\ .10 & .28 & .40 & .20 & 0 \\ .40 & .15 & .15 & .15 & .80 \end{bmatrix}$, $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$ and consider the system of linear equations $T\mathbf{x} = \mathbf{x}$.

(a) Write out the five equations in this system:

(b) Collect terms in your equations to get a homogenous linear system, and write out the five new equations:

2. Let $B\mathbf{x} = \mathbf{0}$ denote the homogenous system you obtained in 1(b), and calculate the reduced echelon form of $[B\ \mathbf{0}]$. Record the reduced form below. These lines will get the matrix and do the calculation:

econdat	(get the matrix B)
ref([B zeros(5,1)])	(calculate the reduced echelon form of $[B\ \mathbf{0}]$)

3. (hand) First read about Leontief Economic Models in Section 1.6 of the text. Now consider an exchange model economy which has five sectors, Chemicals, Metals, Fuels, Power and Agriculture. Assume the matrix T in question 1 above gives an exchange table for this economy as follows:

$$T = \begin{array}{c} \begin{matrix} C & M & F & P & A \end{matrix} \\ \begin{bmatrix} .20 & .17 & .25 & .20 & .10 \\ .25 & .20 & .10 & .30 & 0 \\ .05 & .20 & .10 & .15 & .10 \\ .10 & .28 & .40 & .20 & 0 \\ .40 & .15 & .15 & .15 & .80 \end{bmatrix} \end{array} \begin{matrix} C \\ M \\ F \\ P \\ A \end{matrix}$$

Notice that each column of T sums to one, indicating that all output of each sector is distributed among the five sectors, as should be the case in an exchange economy. The system of equations $T\mathbf{x} = \mathbf{x}$ must be satisfied for the economy to be in equilibrium. As you saw above, this is equivalent to the system $B\mathbf{x} = \mathbf{0}$.

(a) Let $\mathbf{x_C}$ represent the value of the output of Chemicals, $\mathbf{x_M}$ the value of the output of Metals, etc. Using the reduced echelon form of $[B \ \mathbf{0}]$ from question 2, write the general solution for $T\mathbf{x} = \mathbf{x}$:

$$\begin{bmatrix} \mathbf{x_C} \\ \mathbf{x_M} \\ \mathbf{x_F} \\ \mathbf{x_P} \\ \mathbf{x_A} \end{bmatrix} =$$

(b) Suppose that the economy described above is in equilibrium and $\mathbf{x_A} = 100$ million dollars. Calculate the values of the outputs of the other sectors and record this particular solution for the system $T\mathbf{x} = \mathbf{x}$:

$$\begin{bmatrix} \mathbf{x_C} \\ \mathbf{x_M} \\ \mathbf{x_F} \\ \mathbf{x_P} \\ \mathbf{x_A} \end{bmatrix} =$$

4. (hand) Consider the matrices T and B created above. As already observed, each column of T sums to one. Consider how you obtained B from T and explain why each column of B must sum to zero.

5. (Extra credit; attach paper) Let B be any matrix of any shape, with the property that each column of B sums to zero. Explain why the reduced echelon form of B must have a row of zeros.

MATLAB Project: Reduced Echelon Form and ref Name_____

Purpose: To calculate the reduced echelon form by hand and with **ref**, and to see some effects of roundoff error.
Prerequisite: Section 1.2.
MATLAB functions used: **-**, **/**; **format**; and **ref** and **rowdat** from Laydata Toolbox

Background: Read about elementary row operations and reduced echelon form in Section 1.2. To learn about the functions from the Laydata Toolbox, use **help** or see his Student Study Guide.

1. Type **rowdat** to get the two matrices A and B below. For each of them, first calculate the reduced echelon form by hand, then use the function **ref** from the Laydata Toolbox to calculate the reduced echelon form again

(a) (hand) The matrix A below is Example 2 in Section 1.2, where an echelon form is calculated. Repeat here the row operations done in the text, and then finish calculating by hand its reduced echelon form. Show all steps:

$$A = \begin{bmatrix} 0 & -3 & -6 & 4 & 9 \\ -1 & -2 & -1 & 3 & 1 \\ -2 & -3 & 0 & 3 & -1 \\ 1 & 4 & 5 & -9 & -7 \end{bmatrix} \sim$$

(b) (MATLAB) Type **format rat** and then **ref(A)** . Is the output identical to what you obtained above? _____
(If not, redo hand calculations.)

Remarks. MATLAB has a built-in command **rref** for finding the row reduced echelon form of a matrix. In general, the Laydata Toolbox command **ref** is faster because it does not check for rational entries as **rref** does. Except for this rational number issue, the code for **ref** is the same as **rref**.

2. Let $B = \begin{bmatrix} -0.1 & 0.1 & 2 \\ 0.3 & 0.2 & 0.7 \\ 0 & 0.5 & 6.7 \end{bmatrix}$. (a) (hand) Calculate the reduced echelon form of B. Show all steps. Hint: do all scaling at the end.

(b) (MATLAB) Type **ref(B)** . Is the output identical to what you obtained above? _____ (If not, redo hand calculations.)

Remarks. For the majority of small matrices like those used in linear algebra courses, **ref** will return a very accurate result, as it does in the two examples above. One of the reasons **ref** is accurate is that it does not pivot on a position where the value is extremely small. Such a number is often inaccurate in many digits as a result of roundoff error during previous row operations, and it is even possible that theoretically it ought to be a true zero. The usual algorithm for Gaussian elimination chooses the next pivot by looking "to the right and down" for the first nonzero entry in the first nonzero column. If that entry happens to be a very inaccurate number, pivoting on it can lead to a very wrong final result.

Thus it is wise when doing row reduction with a computer or calculator to check the size of potential pivots and not to use one that is extremely small. (The question of what is "extremely small" is a matter of judgment and depends largely on how many digits your computer arithmetic keeps.) The functions **ref** and **rref** have a tolerance variable called **tol** for this purpose. A user can specify a value for **tol**, but if a value is not specified, then **ref** uses a default value. If the absolute value of a number is less than **tol**, then **ref** will not pivot on that position.

3. (MATLAB) Use the same matrix B as in question 2. Here you will force **ref(B)** to return the wrong answer by making the value of **tol** too small, and you will experiment to figure out a good estimate for what is the default value of **tol**.

(a) Type each of the following commands and record the result:

ref(B, 1e-15)

ref(B, 1e-16)

Now you can be certain that the default value for **tol** is smaller than 10^{-15} and not smaller than 10^{-16}. Explain why:

(b) Experiment to find more precise bounds for the default value of **tol** and record the ones you find: _____

Hint: Type **ref(B, 9e-16)** , then **ref(B, e-15)**. Is **tol** between **9e-16** and **1e-15** ? Etc.

MATLAB Project: Rank and Linear Independence Name_____

Purpose: To define rank and learn its connection with linear independence.
Prerequisite: Section 1.7
MATLAB functions used: ' , **rank**; and **indat**, **randomint**, and **ref** from Laydata Toolbox

Definition. The *rank* of a matrix A is defined to be the number of pivot columns in A. Notice this is well defined since the reduced echelon form of a matrix is unique.

One way to find rank is to calculate the reduced echelon form and then count the number of pivot columns. Another quicker way is to use MATLAB's **rank** function. To obtain the data for these exercises, type **indat**.

Example 1. Let $A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 5 & 6 & 7 \\ 6 & 9 & 12 & 15 \\ 1 & 1 & 1 & 1 \end{bmatrix}$. Type **A, ref(A)** to see A and ans = $\begin{bmatrix} 1 & 0 & -1 & -2 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$. There are two pivot columns in the reduced matrix so the rank of A is 2. Type **rank(A)** to see ans = 2 at once.

1. Use both of the above methods to find the rank of each of the following four matrices. For example, type **B**, **ref(B)**, **rank(B)** and fill in the blanks below.

$$B = \begin{bmatrix} 1 & 2 & 4 \\ 1 & -1 & 1 \\ 0 & 1 & 2 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 2 & -3 & 0 \\ 1 & 2 & 1 & 1 \\ 3 & 6 & -5 & 1 \end{bmatrix} \quad D = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 5 & 7 & 9 \end{bmatrix} \quad E = \begin{bmatrix} 1 & 1 & 1 & 1 & 4 & 1 \\ 1 & 2 & 0 & 4 & 7 & 6 \\ 1 & 3 & -1 & 10 & 13 & 21 \\ 1 & 4 & -2 & 20 & 23 & 56 \\ 1 & 5 & -3 & 35 & 38 & 126 \\ 1 & 6 & -4 & 56 & 59 & 252 \end{bmatrix}$$

Record the reduced echelon form of each matrix, circle each pivot column and record the rank:

Rank: _____ _____ _____ _____

2. (hand) Read the definition of linear independence in Section 1.7. Let $M = [\mathbf{v_1}\ \mathbf{v_2} \ldots \mathbf{v_k}]$ be a matrix whose columns are $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}$. Use this definition and the definition of rank above to explain why the following are logically equivalent (i.e., why each implies the other):

(a) The set of vectors $\{\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}\}$ is linearly independent.

(b) The rank of M is k.

3. Use the method of question 2 to answer the questions below. In each case, write the appropriate matrix, use MATLAB to calculate its rank, and record the rank. (To learn how to store a matrix, see Section 1 of the computer project "Getting Started With MATLAB.")

Example 2: (a) The set $\left\{ \begin{bmatrix}1\\2\\3\\0\end{bmatrix}, \begin{bmatrix}2\\3\\4\\1\end{bmatrix}, \begin{bmatrix}3\\4\\5\\1\end{bmatrix}, \begin{bmatrix}4\\5\\6\\1\end{bmatrix} \right\}$ is not linearly independent. Verify this by typing **F, ref(F)**.

(b) Type **G, ref(G)** to find out if the set $\left\{ \begin{bmatrix}1\\2\\3\\0\end{bmatrix}, \begin{bmatrix}2\\3\\4\\1\end{bmatrix}, \begin{bmatrix}3\\4\\5\\1\end{bmatrix}, \begin{bmatrix}3\\5\\6\\1\end{bmatrix} \right\}$ is linearly independent. Is it? _____

(c) Examine the matrices B, C, D, and E in question 1. For which of these matrices is the set of its columns a linearly independent set? _____

(d) Let $\mathbf{v1} = (2,3,5,1)$, $\mathbf{v2} = (1,1,-2,9)$, and $\mathbf{v3} = (3,4,0,0)$. Is the set $\{\mathbf{v1},\mathbf{v2},\mathbf{v3}\}$ linearly independent? _____

Record the matrix you used and its rank:

5. (MATLAB and hand; use your paper) The *transpose* of a matrix X is defined to be the matrix X^{T} whose columns are formed from the corresponding rows of X. For example, if $X = \begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}$, then $X^{\mathrm{T}} = \begin{bmatrix}1&4\\2&5\\3&6\end{bmatrix}$.

Do at least 20 experiments to compare the rank of a matrix and the rank of its transpose. Use many different size matrices. In MATLAB, the single quote creates transpose -- for example, typing **X'** will create the transpose of **X**.

Start by calculating **rank(A), rank(A')**, etc. for all the matrices used above; then do the same for some matrices you create yourself. For example, you could execute lines like

X = randomint(3,7), rank(X), rank(X')

What do you think might always be true about rank(X) and rank(X^{T}), based on your experiments?

6. (Extra Credit) Prove the conjecture you stated in question 5. Use your paper and attach.

MATLAB Project: Visualizing Linear Transformations of the Plane Name_____

Purpose: To understand the standard matrix of a linear transformation. In particular, to see the geometric effect of how a 2x2 matrix transforms R^2; and conversely, to learn how to write 2x2 matrices that will transform R^2 in specific ways.

Prerequisites: Section 1.9. A theorem from Section 1.4 is needed for the extra credit problem.

MATLAB functions used: **visdat** and **drawpoly** from Laydata Toolbox

Background. As shown in Theorem 10 in Section 1.9, when a linear transformation $T: R^n \to R^n$ is given, it can be identified with a matrix, and this is an easy way to get a formula for the function. Let $T: R^n \to R^n$ be a linear transformation and let $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$ denote the columns of the nxn identity matrix. Figure out what each $T(\mathbf{e}_i)$ should be and write each $T(\mathbf{e}_i)$ as a column vector. If you then define the matrix $A = [\, T(\mathbf{e}_1) \;\; T(\mathbf{e}_2) \;\; .. \;\; T(\mathbf{e}_n) \,]$, then it will be true that $T(\mathbf{x}) = A\mathbf{x}$ for all $\mathbf{x}$, and $A\mathbf{x}$ gives is a formula for the function. In other words, given a linear transformation $T: R^n \to R^n$, if you know its values at just the n independent vectors $\mathbf{e}_1, \mathbf{e}_2, .., \mathbf{e}_n$, then its value at every point $\mathbf{x}$ is determined!

1. Example. The 2x2 linear transformation that maps $\mathbf{e}_1$ to $\mathbf{e}_1 + \mathbf{e}_2$ and $\mathbf{e}_2$ to $\mathbf{e}_1 - \mathbf{e}_2$ is $\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$.

The 3x3 matrix transformation that maps $\mathbf{e}_1$ to $\begin{bmatrix} 3 \\ -2 \\ 1 \end{bmatrix}$, $\mathbf{e}_2$ to $\begin{bmatrix} 6 \\ 0 \\ 7 \end{bmatrix}$ and $\mathbf{e}_3$ to $\begin{bmatrix} 5 \\ 4 \\ -1 \end{bmatrix}$ is $\begin{bmatrix} 3 & 6 & 5 \\ -2 & 0 & 4 \\ 1 & 7 & -1 \end{bmatrix}$.

2. Example. The function that reflects R^2 across the line $y = -x$ is a linear transformation. Notice it must map $\mathbf{e}_1$ to $-\mathbf{e}_2$ and $\mathbf{e}_2$ to $-\mathbf{e}_1$, so its matrix is $\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$. See the sketch in Table 1 of Section 1.9.

3. Exercise. (hand) (a) Write a 2x2 matrix that maps $\mathbf{e}_1$ to $4\mathbf{e}_2$ and $\mathbf{e}_2$ to $-\mathbf{e}_1$:

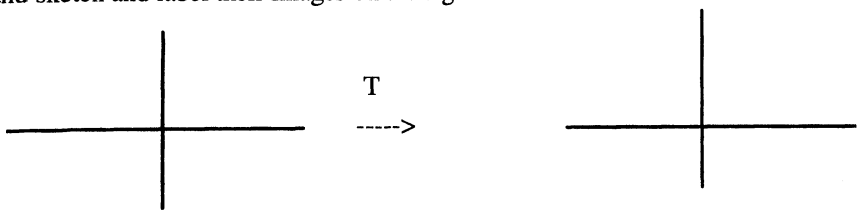
(b) Write a 2x2 matrix that reflects R^2 across the line $y = x$:

More background. A matrix transformation always maps a line onto a line or a point, and maps parallel lines onto parallel lines or onto points. (See exercises 25-28 in Section 1.8.) In the following question, you will verify these things for a particular matrix.

4. Exercise. (hand) Let $M = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$.

(a) Explain why the function $T(\mathbf{x}) = M\mathbf{x}$ maps the x-axis onto the line $y = x$, and why it maps the line $y = 2$ onto the line $y = x+2$. (Hints: A general point on the x-axis is of the form $\begin{bmatrix} t \\ 0 \end{bmatrix}$; calculate $M\begin{bmatrix} t \\ 0 \end{bmatrix}$ and interpret where those image points lie. Similarly, calculate $M\begin{bmatrix} t \\ 2 \end{bmatrix}$ and interpret.)

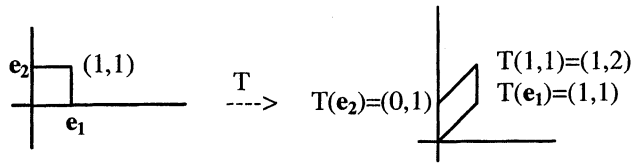
(b) Sketch here what you showed algebraically in 4(a). That is, sketch the x-axis and the line $y = 2$ on the left axes below, and sketch and label their images on the right:



Still more background. Because a matrix transformation maps parallel lines to parallel lines, it will map any parallelogram to another parallelogram (which could be degenerate – one line segment or a single point). So when a linear transformation and parallelogram are given, the easy way to draw the image of the parallelogram is to plot the images of its four vertices and connect those points to make a parallelogram.

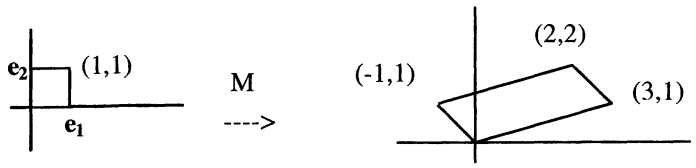
Define the *standard unit square* to be the square in R^2 whose vertices are (0,0), (1,0), (1,1) and (0,1). When you want to visualize what a 2x2 matrix transformation does geometrically, it is particularly useful to sketch the image of this standard square. Seeing how this square gets moved or distorted shows what the transformation does to the x-axis and y-axis and thus gives a good idea what the transformation does geometrically to the whole plane. Recall that any linear transformation maps the origin to itself (why?), so you only need to figure out where the transformation maps the other three vertices.

5. Example. Continue to use $T(\mathbf{x}) = M\mathbf{x}$ where $M = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$. Sketch the image of the standard unit square:



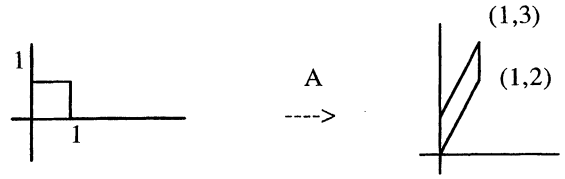
So you can see that the y-axis stays fixed and the x-axis is mapped onto the line y=x, causing a vertical shear of the plane. See more examples like this in Table 3 in Section 1.9.

6. Example. Find a matrix M which maps the standard unit square to the parallelogram with vertices (0,0), (3,1), (2,2), (-1,1). To do this, sketch the parallelogram and recognize that $M\mathbf{e}_1$ and $M\mathbf{e}_2$ must be (3,1) and (-1,1), or vice versa. (Why?) So either of the matrices $\begin{bmatrix} 3 & -1 \\ 1 & 1 \end{bmatrix}$ or $\begin{bmatrix} -1 & 3 \\ 1 & 1 \end{bmatrix}$ will work. Be sure you understand why these are the only two matrices that will work here. Calculate the image of (1,1) under each of these matrices, to verify that it is (2,2).



7. Exercise. (hand) Each of the seven matrices below is one of the special, simple types described in Sections 1.8 and 1.9. Each determines a linear transformation of R^2. For each, sketch the image of the standard unit square, label the vertices of the image, and describe how the matrix is transforming the plane. To get you started, answers are given for the first matrix.

$A = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}$ Description: a vertical shear. It leaves the y-axis fixed and increases the slope of all other lines through origin.



$B = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$ Description: _____

$C = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ Description: _____

$D = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$ Description: _____

$E = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$ Description: _____

$F = \begin{bmatrix} \cos(\pi/4) & -\sin(\pi/4) \\ \sin(\pi/4) & \cos(\pi/4) \end{bmatrix}$ Description: _____

$G = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ Description: _____

8. Exercise. (MATLAB) In computer graphics, transformations are usually accomplished by applying a succession of simple matrix transformations – dilations, shears, reflections, rotations and projections. Here you will do a variety of such.

To begin, type **visdat** to get the matrices A, B, ..., G used above. You will also get a matrix called **box** whose columns contain the coordinates of the vertices of the standard unit square.

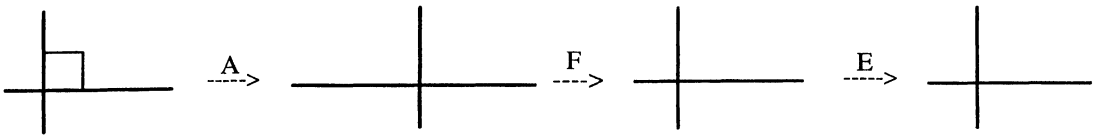
Type **A, box** and **A*box** and record these below. Notice the command **A*box** causes MATLAB to multiply A times each column of box – i.e., the columns of **A*box** are the images under A of the vertices of the standard unit square.

A = box = A*box =

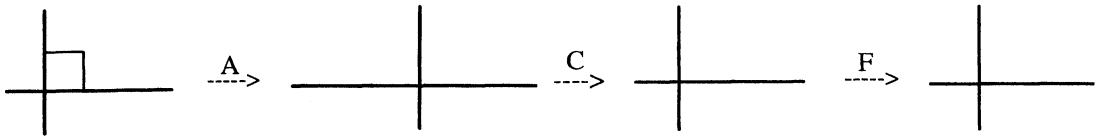
When X is a matrix with two rows, each column represents a point in R^2, and the command **drawpoly(X)** will plot those points and draw a line segment from each to the next one. Try this: type **drawpoly(box)** to see the standard unit square. Then type **drawpoly(A*box)** to see the image of that square under A. To see both figures on the same axes, type **drawpoly(box, A*box)** -- first you will see the square, then press [Enter] and you will also see its image under A.

(a) The commands below perform several successive transformations of the standard unit square. The first one does a shear using A; the second does the shear followed by rotation of the plane through $\pi/4$ using F; the third does the shear followed by the rotation and then reflects the plane across the line y = -x using E. Type these lines, and watch carefully to see the result of each successive transformation. Sketch the new figure obtained after each step:

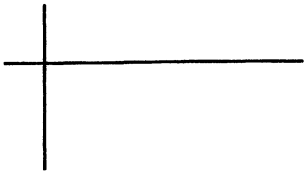
drawpoly(A*box)
drawpoly(F*(A*box))
drawpoly(E*(F*(A*box)))



(b) First shear using A, then reflect across the x-axis using C, then rotate through $\pi/4$ using F. Use **drawpoly** to sketch the result of each successive transformation, and sketch:

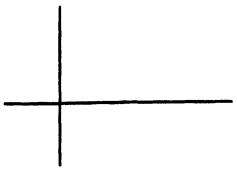


9. Exercise. (hand) Sketch the parallelogram with vertices (0,0), (4,2), (0,-4), (4,-2) and write two different 2x2 matrices X and Y which would transform the standard unit square into this parallelogram. Use **drawpoly** to verify that your matrices work:

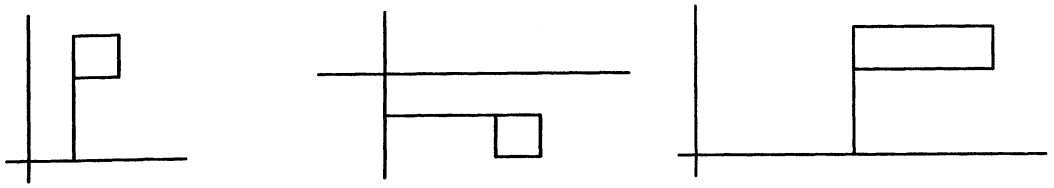


X = Y =

10. Exercise. (hand) Sketch the parallelogram with vertices (1,1), (1,2), (3,1), (3,2). Explain why no 2x2 matrix transformation could map the standard unit square onto this figure.



11. Example. In the figure below called flag1, the flagpole is from (1,0) to (1,3), and the vertices of the flag itself are (1,2), (1,3), (2,3), and (2,2). Label these points. We will find matrices which transform flag1 into the other figures, flagA and flagB.



flag1 flagA flagB

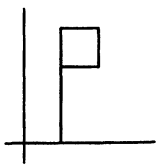
First consider flagA. One way to figure out the matrix would be to simply "see" that rotating the plane through $-\pi/2$ clearly takes flag1 into flagA, so the matrix must be $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$. Alternatively, inspect flag1 and flagA to see where $\mathbf{e}_1 = (1,0)$ and $\mathbf{e}_2 = (0,1)$ must be mapped, so as to find the columns of the desired matrix. Clearly (1,0) maps to (0,-1) – so the matrix we seek must look like $\begin{bmatrix} 0 & a \\ -1 & b \end{bmatrix}$. It is also easy to see by inspection that (1,1) must map to (1,-1), so solve the equation $\begin{bmatrix} 0 & a \\ -1 & b \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ for a and b, and you will find this yields the same matrix $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$.

Now consider flagB. Clearly (1,0) must be mapped to (3,0). Since there appears to be no skewing or dilation in the vertical direction, you could guess that (0,1) maps to itself, hence the matrix is $\begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$. Or you could proceed as above, noticing that (1,1) must map to (3,1) and using that to solve for the entries of the second column of the matrix. This would also yield $\begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$.

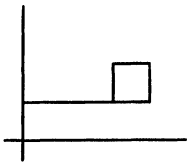
Whatever method you use to calculate the matrix, you should check that your matrix really does what you want. An easy way would be to use **drawpoly** to sketch flag1 and its image under A. The vertices of flag1 are already stored in the matrix **flag1**, so store your matrices and then use **drawpoly**. Try these commands to do that:

MA = [0 1; -1 0], drawpoly(flag1, MA*flag1)
MB = [3 0; 0 1], drawpoly(flag1, MB*flag1)

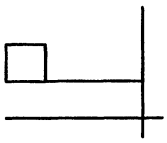
12. Exercise. The figure flag1 is shown again below. One at a time, consider each of the other three flags sketched below and find a 2x2 matrix which maps **flag1** onto it. Use **drawpoly** to verify that each of your matrices does what you want, and record each matrix below the appropriate figure.



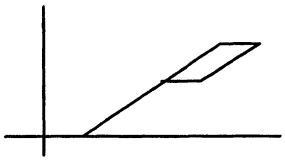
flag1



flag2



flag3



flag4

Matrices: _____ _____ _____

13. Exercise. (Extra credit, use your paper and attach.) Let T: $R^n \to R^n$ be a linear transformation. As pointed out in "Background" on page 1, T is completely determined by its values on the special vectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$.

Prove that T is completely determined by its values on any n independent vectors.

Here is an outline. Assume T: $R^n \to R^n$ is a linear transformation, let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ be independent vectors in R^n and suppose $T(\mathbf{v}_1), T(\mathbf{v}_2), \ldots, T(\mathbf{v}_n)$ are known. Let $\mathbf{x}$ be any element of R^n. Explain why $\text{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\} = R^n$, so $\mathbf{x}$ equals some linear combination of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$, then show $T(\mathbf{x})$ can be calculated using $T(\mathbf{v}_1), T(\mathbf{v}_2), \ldots, T(\mathbf{v}_n)$. (For the spanning fact, let $A = [\mathbf{v}_1\ \mathbf{v}_2\ \ldots\ \mathbf{v}_n]$, explain why A has a pivot in each column, then why a pivot in each row, and apply Theorem 4 in Section 1.4.)

MATLAB Project: Population Migration Name_____

Purpose: To study the population movement described in Exercise 11, Section 1.10 in more detail.
Prerequisite: Section 1.10
MATLAB functions used: ***, /, :, for, end, plot, print**; and Laydata Toolbox

ATTACH YOUR PLOTS AND TURN IN WITH THIS PAPER.

1. Read Exercise 11, Sec. 1.10. Notice this is a simple migration model, which assumes people just move around and the total population of the US remains constant. Hence, if **x** is a vector whose components are the number of people in each area this year, then $M\mathbf{x}$ is the number in each area next year.

(a) To obtain the data from Lay's Toolbox for this exercise, type the lines

```
c1s10
11
```

You will get $M = \begin{bmatrix} 0.9828 & 0.0026 \\ 0.0172 & 0.9974 \end{bmatrix}$ and $\mathbf{x0} = \begin{bmatrix} 29716000 \\ 218994000 \end{bmatrix}$.

(b) (hand) Describe the calculations needed to produce the entries in M, based on the information in the text.

2. Continue to consider the migration model described above.

(a) Calculate the population in CA and in the rest of the US for the years 1990 - 2000 and store that data as the columns of a matrix P.

To do this, type the lines below. The first line converts the population data to millions. The second line builds P one column at a time: the loop "**for i = 1:10 ... end**" causes MATLAB to perform the commands **x = M*x; P = [P x];** ten times; each iteration calculates a new **x** and adjoins that new column to the columns already in **P.** To learn more, type **help for** or see the MATLAB boxes in the Study Guide. The semicolons suppress printing during the calculations. The third line causes **P** to be displayed after the calculations are finished.

```
x = x0/1e6
P = x; for i = 1:10, x = M*x; P = [P  x]; end
P
```

Record the data from **P** in the table below. Round each number to 4 digits.

Population in millions, assuming no external migration

Year	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000
CA											
Rest of US											

(b) Plot the population in CA, and the population in the rest of the US, versus years, on the same graph. The following lines will do this:

yr = 1990:2000 (create the vector [1990 1991 ... 2000])
plot(yr, P), axis([1990 2000 20 240])

(c) Print your graph, title it, and label each curve "CA" and "US." You can do labeling by hand after printing, or use the commands **title**, **xlabel**, **ylabel**, **gtext** before printing. Attach the plot to this paper.

3. Suppose instead that the population is actually increasing each year because of immigration from outside the U.S., say 0.1 million people immigrate to CA and 2 million to the rest of the US each year. Then if data is expressed in millions and **x** is the population vector this year, $M\mathbf{x} + \begin{bmatrix} .1 \\ 2 \end{bmatrix}$ will be the population vector next year.

(a) Calculate the new population predictions for 1990-2000; the following lines will do this:

x = x0/1e6, d = [.1; 2]
P = x; for i = 1:10, x = M*x + d; P = [P x]; end, P

Record the data from **P** in the following table. Round each number to 4 or 5 digits.

Population in millions, assuming external migration

Year	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000
CA											
Rest of US											

(b) Arrow up to execute the **plot** command you typed before. This will produce a graph of the new data; print and label this graph as instructed above, and attach to this paper.

Notes about printing graphics:

In MATLAB 4 or 5, you can print the current graphics screen by typing the command **print**. Or you can display the graphics screen and choose Print in the pulldown File Menu on the Graphics screen. Type **help print** for more information.

MATLAB Project: Elementary Analysis of the Spotted Owl Population Name_____

Purpose: To study the owl population with several different survival rates for juveniles.
Prerequisite: Section 1.10 and the example at the beginning of Chapter 5.
MATLAB functions used: **+, * , ' , :, sum, for, end, plot, print** or **prtsc**; and **owldat** from Laydata Toolbox.

Background. Read the description of the spotted owl population example at the beginning of Chapter 5. To summarize, these owls have three distinct life stages: juvenile (first year), subadult (second year) and adult (third year and older). Let $\mathbf{x_k} = \begin{bmatrix} j_k \\ s_k \\ a_k \end{bmatrix}$ and $A = \begin{bmatrix} 0 & 0 & .33 \\ t & 0 & 0 \\ 0 & .71 & .94 \end{bmatrix}$ where j_k, s_k and a_k denote the number of owls in each stage in year k, t is the survival rate for juvenile→subadult, and $\mathbf{x_{k+1}} = A\mathbf{x_k}$. The example in the text reports that the population will eventually die out if t = .18 but not if t = .30. You will verify these facts here.

1. Let t = .18 and suppose there are 100 owls in each life stage in 1997. Type **owldat** to get the matrix A (with t = .18) and the vector $\mathbf{x_0} = \begin{bmatrix} 100 \\ 100 \\ 100 \end{bmatrix}$. Use the following MATLAB code to calculate the population in each stage and the total population in 1998:

```
x = x0;
x = A*x, total = sum(x)
```

Use the up arrow key to retrieve the last line and execute it two more times. Round the numbers to integers and record the results so far in the table below.

2. (a) It is tedious to do the calculation year by year, as in question 1. Type the following line to calculate the population vectors through year 2020 and store these as the columns of a matrix P. (See Remarks below about the **for** command.)

```
x=x0; P=x; for i = 1998:2020, x=A*x; P = [P x]; end
P
```

Record the data for 2010, 2020 in the table below. Round numbers to integers. An easy way to determine this data is to enter:

```
M = P(:, [14 24]), sum(M)
```

(since column 1 contains 1997 data, column 14 contains data for 2010, etc.; **sum(M)** sums the columns when **M** is a matrix)

Population when the juvenile→subadult survival rate is .18

Year	1997	1998	1999	2000	2010	2020
Juvenile	100					
Subadult	100					
Adult	100					
Total	300					

(b) Type the following line to plot curves for the data in each of the three age groups. Because **yr** is a row vector the same shape as each row of **P**, the command **plot(yr, P)** will plot row one of **P** against **yr**, then row 2 against **yr**, etc. Using **plot** this way causes the three graphs to appear in different colors and line types on the same axes.

yr = 1997:2020; plot(yr, P)

(c) Label your curves ("Adult," "Subadult," "Juvenile"). Print your graph. (See Remarks below.)

(d) Does it appear that the owls will die out? _____

3. Now let t = .30 and repeat the instructions of question 2 to verify that the population of owls will not die out if the survival rate of juveniles to subadults is .30 instead of .18 . To begin, type **A(2,1) = .30** and then use ↑ to retrieve the line with the **for** loop, so you do not have to type it again.

Plot the new data, label curves, print graph, and record data in the table below. Round numbers to integers.

Population when juvenile→subadult survival rate is .30

Year	1997	1998	1999	2000	2010	2020
Juvenile	100					
Subadult	100					
Adult	100					
Total	300					

4. Repeat the calculations and plotting in question 3 for these additional values of t: .20, .24, .26, .28.

You do not have to record data or print graphs for these, but just list all the values of t for which your calculations suggest that the population of owls seems to survive: _____

Remarks:

The command **for i = 1998:2020 ... end** is a loop; it causes MATLAB to perform the intermediate commands for each value of i, so 23 times. At each step, the command **x = A*x** calculates the new **x** and then the command **P = [P x]** adjoins that new **x** to the columns already in *P*. Notice the final *P* has 24 columns (why?).

To learn more about **for** and **sum**, see the MATLAB boxes in the Study Guide, or type **help for** and **help sum**.

In MATLAB 4, 5, or 6, you can print the current graphics screen by typing the command **print**. Or you can click on a graphics screen to display it, and then choose Print in the pulldown File Menu on that screen.

There are other ways to print the file such as exporting the file or by copying the screen to the computer's clipboard. On some computers, for example, simultaneously pressing the **ALT** button and the **Print Scrn** button will copy the active window's contents to the computer's clipboard. The contents can then be edited in a program such as Paint or directly pasted into a word processing file.

If necessary, consult your instructor or lab assistant, or just copy the graph by hand.

MATLAB Project: Lower Triangular Matrices Name_____

Purpose: To investigate properties of products of lower triangular matrices.
Prerequisite: Section 2.1
MATLAB functions used: **+ , * , tril, rand, eye**

1. Create several lower triangular nxn matrices, calculate their products in pairs, and see what appears to be true. For example you could do this for two different 2x2 matrices whose lower triangle entries are random numbers by typing:

n = 2
L1 = tril(rand(n)), L2 = tril(rand(n)), L1*L2, L2*L1

Execute the second line several times and inspect the result each time. Repeat with n = 3,4,5. The easy way is to press the up arrow on your keyboard to retrieve the second line, and then press [Enter] to execute it again. (Each time you execute **rand(n)** it produces a new nxn matrix with random number entries; the function **tril** puts zeros in the upper triangle, so **tril(rand(n))** produces a random lower triangular matrix.)

(a) (hand) For each pair L_1 and L_2, what entries of L_1L_2 and L_2L_1 appear to be the same? Is that true for non-triangular matrices? (Try the following calculation several times: **A=rand(2), B=rand(2), A*B, B*A** .)

(b) (hand, use back or attach an extra sheet) Prove that the product of any two nxn lower triangular matrices is lower triangular. Here is one way to begin: "Let $L_1 = \begin{bmatrix} a_{11} & 0 & .. & 0 \\ a_{21} & a_{22} & .. & 0 \\ . & . & .. & 0 \\ a_{n1} & a_{n2} & .. & a_{nn} \end{bmatrix}$ and $L_2 = \begin{bmatrix} b_{11} & 0 & .. & 0 \\ b_{21} & b_{22} & .. & 0 \\ . & . & .. & 0 \\ b_{n1} & b_{n2} & .. & b_{nn} \end{bmatrix}$." Then write out what the i,j entry of L_1L_2 looks like and explain why it must be zero when $i < j$.

2. Now investigate products of lower triangular matrices which have all diagonal entries equal to 1. Such a matrix is called a *unit* lower triangular matrix. For example you could type

n = 2
L1 = tril(rand(n), -1) + eye(n), L2 = tril(rand(n), -1) + eye(n), L1*L2, L2*L1

Execute this line several times and inspect the result each time. Repeat with n = 3,4,5. (The second input -1 for **tril** causes zeros to be put above the first subdiagonal, that is, on the main diagonal as well as in the upper triangle. So adding the identity matrix **eye(n)** produces a random lower triangular matrix with 1's on the diagonal.)

(a) (hand) For each pair L_1 and L_2, what entries of L_1L_2 and L_2L_1 appear to be the same?

(b) (hand, use back or attach an extra sheet) Prove that the product of any two nxn unit lower triangular matrices will be a unit lower triangular matrix. Notice all you need to prove here is that each diagonal entry of the product is 1 (why?). Begin the same way as in 1(b), but this time let each diagonal entry be 1; then write out what the i,i entry of L_1L_2 looks like, and explain why it must be 1.

3. (Extra Credit) Suppose L is an nxn lower triangular matrix with each diagonal entry nonzero. Create A = [L I], where I denotes the nxn identity matrix. Explain why the reduced echelon form of A must be of the form [I K], where K is another nxn lower triangular matrix with nonzero diagonal entries.

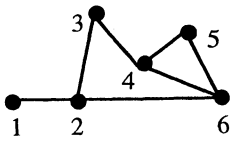
MATLAB Project: The Adjacency Matrix of a Graph Name_____

Purpose: To learn about graph and adjacency matrix, to see how the powers of the adjacency matrix provide information about the graph and vice versa, and to apply these ideas.
Prerequisite: Section 2.1
MATLAB functions used: +, ^; and **adjdat** from Laydata Toolbox

Definitions. A *graph* is a finite set of objects called nodes, together with some paths between some of the nodes, as illustrated below. A *path of length one* is a path that directly connects one node to another. A *path of length k* is a path made up of k consecutive paths of length one. The same length one path can appear more than once in a longer path; for example, 1--2--1 is a path of length two from node 1 to itself in the example below.

When the nodes have been numbered from 1 to n, the *adjacency matrix A* of the graph is defined by letting $a_{ij} = 1$ if there is a path of length one between vertices i and j and $a_{ij} = 0$ otherwise.

Example. Verify that the graph below has the matrix *A* shown as its adjacency matrix.



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

direct path from node 1 to node 2
direct path from node 2 to 1; 2 to 3; 2 to 6
.
.
.
direct path from node 6 to 2; 6 to 4; 6 to 5

Theorem (Interpretation of the powers of an adjacency matrix). If *A* is the adjacency matrix of a graph, then the (i,j) entry of A^k is a nonnegative integer which is the number of paths of length k from node i to node j in the graph.

1. (hand) To understand what the theorem says for the example above, let's carefully examine the (6,3) entry of A^2. Using the "Row-Column Rule," the (6,3) entry of A^2 looks like $a_{61}a_{13} + a_{62}a_{23} + a_{63}a_{33} + a_{64}a_{43} + a_{65}a_{53} + a_{66}a_{63}$.

Evaluate each term in this expression (you finish): (0)(0) + (1)(1) + _____ = ____.

Explain what each term in the sum above tells about paths of length 2 from node 6 to node 3. (For example, $a_{62}a_{23}$ = (1)(1) = 1; this says that the length one paths 6----2 and 2----3 appear in the graph, and together they give one path from node 6 to node 3, of length 2.)

2. Type **adjdat** to get the matrix *A* above, then type **A^2, A^3** and record results:

A^2 = A^3 =

(b) (hand) Notice that the (1,2) entry of A^2 is zero, so there are no paths of length two from node 1 to node 2. Verify this by studying the graph. Similarly, notice that the (6,6) entry of A^3 is two, so there are two paths of length three from node 6 to itself; study the graph to see that they are 6--4--5--6 and 6--5--4--6.

In the same way, study the matrices and the graph and answer:

What are the paths of length two from node 2 to itself? _____

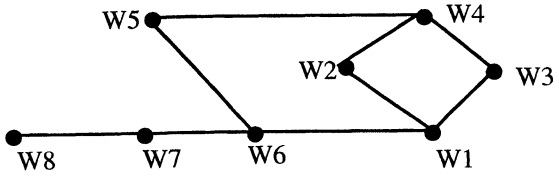
What are the paths of length three from node 3 to node 4? _____

Definition. When we have a graph, we will say that there is a *contact level k* between node i and node j if there is a path of length less than or equal to k from node i to node j.

3. (hand) Suppose A is the adjacency matrix of a graph. Explain why you must calculate the sum $A + A^2 + .. + A^k$ in order to decide which nodes have contact level k with each other:

4. Eight workers, denoted W1, .., W8, handle a potentially dangerous substance. Safety precautions are taken but accidents do happen occasionally. It is known that if a worker becomes contaminated, s/he could spread this through contact with another worker. The following graph shows the level one contacts between the workers.

(a) (hand) Write the adjacency matrix A for the following graph:



A =

(b) Store A, type **A + A^2 + A^3** and record result: $A + A^2 + A^3 =$

Use this to answer the following questions. Which workers have contact level 3 with W3? _____

Which workers have contact level 3 with W7? _____

(c) Define what you mean by a worker being *dangerous*. Be very specific so anyone could decide whether a worker is "dangerous" according to your definition:

(d) Which workers are the most dangerous if contaminated? _____

Which are least dangerous? _____

Use your definition, and explain your answers. This part is important, but whatever you say is okay as long as it agrees with your definition. Use the back or attach an extra sheet.

MATLAB Project: Counting Floating Point Operations Name_____

Purpose: To count the number of arithmetic operations done in certain matrix products and see that the placing of parentheses can make a difference in some products.

Prerequisites: Section 2.1 and 2.7

MATLAB functions used: *** , rand, flops**

NOTE: A version of MATLAB before Version 6 is needed for this project. Version 6 does not have **flops.**

Definition. A *floating point operation*, or "flop" for short, is one addition, subtraction, multiplication or division of two real numbers, done with floating point arithmetic. It is customary to say "addition" to mean addition or subtraction and "multiplication" to mean multiplication or division.

In all versions of MATLAB before Version 6, there is a function called **flops** which counts how many flops are performed in a session. You can see its value at any time by typing **flops**. It can be used another way: you can set its value to zero by typing **flops(0)**, and then it counts the number of floating point operations done after that.

1. (hand) Let **x** and **y** denote nx1 matrices. Explain why, in the (row)(column) product $\mathbf{x}^T\mathbf{y}$, the number of additions is about the same as the number of multiplications. Use this to explain why the number of additions is about the same as the number of multiplications in the product XY of any two matrices (where X and Y denote matrices of any sizes that can be multiplied).

2. This is similar to Exercise 9, Section 2.7. You will see that the number of flops done to calculate ABD can be greatly different, depending on whether you calculate AB first, or BD. Record the number of flops for each choice of D in the table below.

The following lines will get you started. We use semicolons after the matrices involving D because they are quite large. If you want to see those matrices, type commas instead of semicolons:

```
A = rand(2, 2), B = rand(2, 2), D = rand(2, 200);
flops(0), (A*B)*D; flops
flops(0), A*(B*D); flops
```

Repeat the commands using **D = rand(2, 300)** and then **D = rand(2, 400)**. You can avoid some retyping by using the up arrow key.

Count of Flops when D is 2xm

Size of D	2x200	2x300	2x400
A*(B*D)			
(A*B)*D			

3. (hand) Compare your counts above, when m = 200, with the answer to Exercise 9, Section 2.8. Read the problem's instructions, and explain why your counts here are twice as big as the ones given there:

4. Modify the lines typed in question 2 to do the same calculations for square matrices, of sizes 2x2, 5x5 and 10x10. Record your results:

Count of flops when *A*, *B* and *D* are all the same shape

Size of matrices	2x2	5x5	10x10
A*(B*D)			
(A*B)*D			

5. (hand) Explain why rearranging parentheses made a difference in question 2 but no difference in question 4.

MATLAB Project: Cryptography Name_____

Purpose: To see a method for using matrix operations to encode and decode messages.
Prerequisite: Section 2.2 of Lay's text and elementary modular arithmetic
MATLAB functions used: **rem**, * ; and **crypdat** from Laydata Toolbox

Background. Before beginning this project, read the "Notes About Arithmetic Modulo 26" at the end.

A disadvantage of a simple encoding system where each letter of the alphabet is replaced by one other symbol is that it preserves the frequencies of individual letters. For example, the letter **e** is the most common letter in English and, therefore, the letter most commonly appearing in the encoded message will probably be the substitute for **e**. That makes the code breakable by using simple statistical methods. A somewhat more sophisticated method is to divide the uncoded text into groups of letters and replace each group with another group of letters. In this project we will use groups of two letters. The first step in the method described here is to assign each letter a corresponding number between 0 and 25, as shown in the following table. This will help us transfer the problem of letter transformations to a problem involving number transformations.[1]

Substitution Table

A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	0

1. Consider the message HURRY UP

(a) (hand) To encode this message, first divide the message into groups of two letters, adding a "dummy" letter to fill out the last pair since the message has an odd number of letters:

HU RR YU PP

Using the Substitution Table, find the column vector for each pair of letters. The first is given; you fill in the rest:

HU --> $\begin{bmatrix} 8 \\ 21 \end{bmatrix}$, RR --> YU --> PP --->

(b) (MATLAB) Type **crypdat** to get the matrices for this project. Store the vectors above as **a**, **b**, **c** and **d**. Transform each vector by multiplying it by the matrix $A = \begin{bmatrix} 1 & 1 \\ 0 & 3 \end{bmatrix}$, and use arithmetic modulo 26 to reduce each number you see to a new number between 0 and 25. Call the new vectors **a1**, **b1**, **c1** and **d1**. The following line will get you started:

a = [8; 21], A*a, a1 = rem(A*a, 26)

You should see **a** = $\begin{bmatrix} 8 \\ 21 \end{bmatrix}$, *A****a** = $\begin{bmatrix} 29 \\ 63 \end{bmatrix}$ and **a1** = $\begin{bmatrix} 3 \\ 11 \end{bmatrix}$.

Transform **b**, **c** and **d** the same way and record your results below.

*A****b** = **b1** = *A****c** = **c1** = *A****d** = **d1** =

[1]The method used here is still not too safe because statistical analysis using tables of letter-pair frequencies can be used to break it. However, this method can be used in combination with other encoding systems to produce a more secure system. For more about this, see *Elementary Linear Algebra with Applications*, H. Anton and C. Rorres, Wiley, 1987.

(c) (hand) For each of the new vectors find the corresponding letter pair using the Substitution Table. The first one is shown:

a1 --> CK **b1** --> **c1** --> **d1** -->

Finally, write the encoded message:

C K _ _ _ _ _ _ _ _

2. (hand and MATLAB) You have received the following message from a friend:

XORLBSWOSSBUPX

Suppose that you know this message was encoded using the matrix A from question 1. Then to decode it you must create a vector for each block of two letters and multiply these by a matrix B, which is the inverse of A modulo 26. This will not be the usual inverse of A, but instead BA and AB equal I modulo 26. The shorthand for this is $BA \equiv AB \equiv I \bmod 26$. Check the following arithmetic to verify for yourself that $B = \begin{bmatrix} 1 & 17 \\ 0 & 9 \end{bmatrix}$ does satisfy these equations:

$$BA = \begin{bmatrix} 1 & 17 \\ 0 & 9 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 52 \\ 0 & 27 \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \bmod 26 \text{ and } AB = \begin{bmatrix} 1 & 1 \\ 0 & 3 \end{bmatrix}\begin{bmatrix} 1 & 17 \\ 0 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 26 \\ 0 & 27 \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \bmod 26$$

(a) Divide the coded message sent by your friend into blocks of two letters:

_ _ _ _ _ _ _ _ _ _ _ _ _ _

(b) Write the vector for each block:

a1 = **b1** = **c1** = **d1** = **e1** = **f1** = **g1** =

(c) Store your vectors **a1**, .., **g1** and multiply each vector by B. After reducing each entry modulo 26, record these:

a = **b** = **c** = **d** = **e** = **f** = **g** =

(d) Use the Substitution Table to retrieve the letter pairs corresponding to the vectors for the decoded message:

a --> **b** --> **c** --> **d** --> **e** --> **f** --> **g** -->

(d) Write the decoded message:

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

3. (hand and MATLAB) Using the matrix A above, encode the message STAY ON THE PATH. Record the vectors for the original message, the vectors for the coded message, and then the coded message.

4. The following message was encoded using B (instead of A): WDPSYOUFADEVACCK . Decode it! (Notice: for decoding here you will need the inverse of B modulo 26. What is it?) Record the vectors for the coded message, the vectors for the decoded message, and then the decoded message:

5. (hand) Let $C = \begin{bmatrix} 1 & 1 \\ 0 & 5 \end{bmatrix}$. Calculate the inverse of C modulo 26. Show calculations. Call the new matrix D and also show work to verify that $DC \equiv I \bmod 26$ and $CD \equiv I \bmod 26$ are true. First read the Notes below.

Notes About Arithmetic Modulo 26:

(i) Two numbers r and s are called *congruent modulo* 26 if their difference is an integer multiple of 26. When that is true, we write $r \equiv s \bmod 26$. For example, $63 \equiv 11 \bmod 26$ and $-3 \equiv 23 \bmod 26$ are both true.

To *reduce a number modulo* 26 means to subtract or add multiples of 26 to get an integer between 0 and 25. E.g., to reduce 63, subtract 52 to see that $63 \equiv 11 \bmod 26$; to reduce -3, add 26 to see that $-3 \equiv 23 \bmod 26$.

If the original number is positive, you can reduce it modulo 26 by dividing it by 26 and using the remainder. MATLAB's function **rem** does this, and it works on a single number or on a matrix. For example, typing **rem(29, 26)** reduces the number 29 and returns 3; if **x** is the matrix **[29; 63]**, then typing **rem(x, 26)** reduces each number in the matrix, and returns $\begin{bmatrix} 3 \\ 11 \end{bmatrix}$. If the original number is negative, you can still use **rem** but you must add 26 to what **rem** outputs. For example, typing **rem(-30, 26)** returns -4; add 26 to see that $-4 \equiv 22 \bmod 26$. Try these.

(ii) A number r is said to have an *inverse modulo* 26 if there is another number s such that $rs \equiv 1 \bmod 26$. For example, (3)(9) = 27, which is congruent to 1 modulo 26, so 3 and 9 are inverses modulo 26. However, 2 has no inverse modulo 26. To see this, calculate 2 times each number between 0 and 25 and check that none of these products are congruent to 1 modulo 26.

(iii) If C and D are integer matrices and reducing each number in C modulo 26 yields D, we write $C \equiv D \bmod 26$. A square integer matrix C is said to have an *inverse modulo* 26 if there is another integer matrix D such that $CD \equiv I \bmod 26$ and $DC \equiv I \bmod 26$. It is true that such D will exist if and only if $\det(C)$ is a number which has an inverse modulo 26. For example, the matrix A used in questions 1-4 has determinant 3, so A should have an inverse modulo 26, as of course it does.

To calculate the inverse of C modulo 26, calculate the reduced echelon form of $[C\ I]$ by doing row operations as usual, except use only integer multipliers and reduce each number modulo 26. Important: to scale a row, do not divide – instead, multiply and then reduce modulo 26. Here is an example.

Calculate the inverse of $\begin{bmatrix} 1 & 0 \\ 2 & 9 \end{bmatrix}$: $\begin{bmatrix} 1 & 0 & 1 & 0 \\ 2 & 9 & 0 & 1 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 9 & -2 & 1 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 27 & -6 & 3 \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 20 & 3 \end{bmatrix} \pmod{26}$.

Check: $\begin{bmatrix} 1 & 0 \\ 2 & 9 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 20 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 182 & 27 \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \pmod{26}$; and $\begin{bmatrix} 1 & 0 \\ 20 & 3 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 2 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 26 & 27 \end{bmatrix} \equiv \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \pmod{26}$.

MATLAB Project: Using Backslash to Solve Ax = b Name_____

Purpose: To learn about backslash and why it is the preferred method for solving systems Ax = b when A is invertible.

Prerequisites: Section 2.2 and the discussion of condition number in Section 2.3

MATLAB functions used: \, **inv**, **rref, format**, **rand**, **norm**, **diary**, **hilb**; and **ref** from Laydata Toolbox

Background: This project is about square invertible matrices only. Suppose A is such a matrix and you want to solve $A\mathbf{x} = \mathbf{b}$. By Theorem 5 in Section 2.2, there is a unique solution to this system. MATLAB has a special operator called backslash for solving this type of system, and it usually gives excellent results. It is the method people use in professional settings. To use it, store A and **b** and type **A\b** .

You may know two other methods in MATLAB for solving $A\mathbf{x} = \mathbf{b}$: you could type **ref([A b])** to get the reduced echelon form of the augmented matrix, and then read the solution from its last column; or, because A is assumed invertible here, you could type **inv(A)*b** , which by Theorem 5 must give the unique solution.

Backslash is the best of these methods. It uses an algorithm that is fast and minimizes roundoff error. It also checks the condition number of the coefficient matrix. If the condition number is large, it will be hard to get an accurate answer using any numerical method. Fortunately such matrices occur very rarely in real world problems. But if backslash does detect a very large condition number, it will warn you by printing a message "Matrix is close to singular or badly scaled. Results may be inaccurate." Do not ignore such a warning if you ever see it, for it means the solution is probably not correct to very many digits. If you need more accuracy, consult a numerical analyst.

The **inv** function also checks condition number, but calculating A^{-1} requires a lot more arithmetic than backslash.

It is definitely not wise to use **ref** to solve real world problems. That function was written just to help students learn linear algebra, so its algorithm is not optimal, and **ref** will not warn you if your linear system is one of those rare ones for which it is hard to get an accurate solution.

1. (MATLAB) Here you will use the square matrices in exercises 29, 31, 39 and 41 in Section 2.2. For each of these, you will create a linear system $A\mathbf{x} = \mathbf{b}$ and solve it using all three methods described above. You will not see any warnings, so these are "good" problems. You will also see that the solutions are almost identical as expected.

(a) To get started, install a diskette in your a: drive and type:

diary a:\ solve	(open a file called "solve" on your diskette)
format compact	(this causes fewer blank lines to be printed, so more results fit on the screen)
format long e	(tell MATLAB to display numbers in exponent format with 15 digit mantissas)

Type the following lines to use the matrix in exercise 29 for the problem here:

c2s2	(Opens Chapter 2 Section 2 Problems from Lay's Toolbox)
29	(Loads the matrix for problem #29)
[n,n] = size(A);	
b = rand(n,1)	(create a 2x1 column with random number entries)
x1 = A\b	(Method 1: solve $A\mathbf{x} = \mathbf{b}$ using backslash)
x2 = inv(A)*b	(Method 2: solve $A\mathbf{x} = \mathbf{b}$ using **inv**)
R = ref([A b]); x3 = R(:, n+1);	(Method 3: solve $A\mathbf{x} = \mathbf{b}$ using **ref**)

Type **[norm(x1–x2) norm(x1–x3) norm(x2–x3)]** to calculate the lengths of the differences of the three solution vectors, and view them side by side. Record the norm of each difference vector, in the table below. You can round each mantissa to an integer.

(b) Repeat the instructions in (a) for the matrices in exercises 31, 39 and 41 in Section 2.2. Note that in exercises 39 and 41 the matrix is called D, not A, so modify your commands with **D\b**, **inv(D)*b**, **ref([D b])**.

Norms of the difference vectors

	Exercise 29	Exercise 31	Exercise 39	Exercise 41
x1-x2				
x1-x3				
x2-x3				

2. Fortunately, most matrices that show up in real world problems behave well in numerical calculations, like those in question 1. However, it is worthwhile to see some with large condition numbers, so you know they do exist!

One of the classic types of matrices for which none of the methods above tends to yield a very accurate solution is the type called *Hilbert* matrices. There is a Hilbert matrix of every size, and MATLAB has a special command **hilb** for creating them, since they are frequently used as examples and test cases for new software.

You will understand the pattern in their entries best if you display each decimal as a fraction. Type the following lines to see the 4x4 and 5x5 Hilbert matrices in rational format:

format rat, hilb(4), hilb(5)

Study these to see the pattern of entries. Think what the first and last row of the 20x20 Hilbert matrix will look like.

3. (MATLAB) Now solve $A\mathbf{x} = \mathbf{b}$ when A is the 20x20 Hilbert matrix, using the three methods above. You already know what this big matrix looks like, so use semicolons as indicated to avoid printing the matrix and the large vectors:

```
A = hilb(20);  b = rand(20,1);        (create the 20x20 Hilbert matrix and a random 20x1 column)
x1 = A\b;
x2 = inv(A)*b;
R = ref([A  b]); x3 = R(:, 21);
```

If you did not see warnings from the calculations **A\b** and **inv(A)*b** , repeat the calculations using larger Hilbert matrices until you do see warnings -- e.g., try **A = hilb(21); b=rand(21,1); A = hilb(22); b=rand(22,1);** etc. You may be working on a newer computer that keeps more significant digits in its floating point calculations.

Record the size which finally produces warnings: _____.

After solving $A\mathbf{x} = \mathbf{b}$ with a matrix that causes warnings to appear, type the following line in order to view the three solutions you have created, in a format that makes it fairly easy to compare them:

format long e, [x1 x2 x3]

Notice the mantissas of the entries in **x1** and **x2** do agree in some digits. However, this is quite misleading. To see how different all these vectors really are, type **[norm(x1–x2) norm(x1–x3) norm(x2–x3)]** to calculate the norms of the difference vectors, and record those. Round to integers as before:

Norms of the difference vectors for the Hilbert matrix problem

x1–x2: _____ **x1–x3**: _____ **x2–x3**_____

Clearly the warnings are justified! Even the difference **x1-x2** is very large, and **x1** and **x2** were calculated with the two professionally coded methods. These large differences in answers calculated by different algorithms reinforce the warnings. It would be very unwise to assume any of these solutions is very accurate.

4. To finish, type **diary off** , which will close the file called "solve" on your diskette. Exit MATLAB and open this file with your favorite text editor. If it contains more than 4 pages, try to reduce its length before printing. For example, erase unnecessary blank lines or big matrices you may have created, and perhaps reduce the font size. Print the file and attach that printout to this project.

Caution: This project is about square invertible coefficient matrices only. Do not use backslash when you want to solve a system in which the coefficient matrix is not square or may not be invertible. The reason is, the command **A\b** will give you an answer but it won't be what you expect! If you want the "general solution" of such a system, use **ref([A b])** or **rref([A b])** and then write the general solution as you learned to do in Chapter 1.

When A is not invertible, the answer produced by **A\b** is a type of approximate numerical solution called a "least-squares solution." (See Chapter 6.) This type of approximate solution is important for a lot of applications, which is why MATLAB has an easy way to calculate them.

MATLAB Project: Roundoff Error in Matrix Calculations Name_____

Purpose: To see some of the effects of roundoff error by solving a variety of linear systems, using three different algorithms. Some of these problems involve poorly conditioned matrices.

Prerequisites: Section 2.2, discussion of condition number in Section 2.3; and diskette for storing your data file.

MATLAB functions used: *, -, \, **format, diary, inv, rand, norm, hilb, cond**; and **ref** from Laydata Toolbox

Background. Most computer calculations use *floating point arithmetic*, not exact arithmetic. This means each number is stored in "mantissa-exponent" format, with only the first few significant digits kept in the mantissa and all trailing digits simply lost, or "chopped." In base 10 a floating point number looks like $\pm d_0.d_1d_2... d_n\ (10^x)$, where each d_i is an integer between 0 and 9 and d_0 is not zero unless the number is exactly zero. Because of chopping, there will almost always be some roundoff error when numbers are stored and when calculations are done.

Computers usually use base 16 arithmetic, but display numbers in base 10 format. Also, they have more digits stored for a number than they usually display. You can decide how many digits you want to see, whether exponents are shown, etc. Changing the display format does not change the number that is stored, only the way it gets printed out.

1. (MATLAB) Type the following commands to see some of the different ways MATLAB can display numbers. It starts out in the default mode, which is called "format short." For each command, record the result on the right:

x = 12.123456789123456789

format long, x

format long e, x

format short e, x

format short, x

y = .001212345678912345678 9

format long, y

format long e, y

format short e, y

Inspect the result for each format, for both **x** and **y**, to be sure you understand the different formats.

More background. It is important to be able to estimate the accuracy of a computer solution, and to be able to detect when a solution is likely to be inaccurate. Here is one way to check accuracy. When $\mathbf{x}_1$ is a calculated solution to $A\mathbf{x} = \mathbf{b}$, the vector $\mathbf{r}_1 = \mathbf{b} - A\mathbf{x}_1$ is called the *residual vector*, or just the *residual*. Notice that theoretically $\mathbf{r}_1$ should be identically zero, but usually it is not, because of roundoff error. However, if $\mathbf{r}_1$ is quite small, that is a good sign that the calculated solution $\mathbf{x}_1$ is probably reasonably accurate.

It is helpful to define the *norm*, or *length* of a vector $\mathbf{x}$ as $\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + .. + x_n^2}$. For example, if $\mathbf{x} = [-4\ \ 2\ \ -2\ \ 1]$, then $\|\mathbf{x}\| = 5$.

The norm of a residual vector is what people care about, not the particular numbers in its entries. In fact, what really matters is the power of 10 in the norm of the residual vector, not the particular digits in the mantissas of its coordinates. For example, if $\mathbf{x}_1$ is a calculated solution for $A\mathbf{x} = \mathbf{b}$, and if $\|\mathbf{b} - A\mathbf{x}_1\|$ is about 10^{-14}, this suggests that $\mathbf{x}_1$ is a pretty good solution. But if $\|\mathbf{b} - A\mathbf{x}_1\|$ is about 10^{-4}, $\mathbf{x}_1$ is probably not very accurate.

A remarkable fact about professionally written matrix software today is that the algorithms used do give very accurate answers for the vast majority of real world problems. The algorithms in such software are also efficient, meaning they do a minimum amount of calculation and run quite fast, even on very big problems.

However, there are some matrices for which computer calculations are likely to be inaccurate, no matter how good the algorithm. Professional software checks and when it detects such a matrix, it warns the user that some matrix involved in the calculation is *poorly conditioned*. For now, think of A being poorly conditioned as meaning the entries of A and A^{-1} differ dramatically in size. The opposite of being poorly conditioned is being *well conditioned*. There is discussion about condition number in Section 2.3 of Lay's text, and more on page 4 below.

In the questions below, you will calculate solutions to both well conditioned and poorly conditioned systems $A\mathbf{x} = \mathbf{b}$, by three different methods. Each method uses somewhat different calculations. You will see that the solutions can be a little different even when A is well conditioned, and they vary dramatically when A is poorly conditioned.

One more definition will be useful. A *Hilbert matrix* is one of the form

$$\begin{bmatrix} 1 & 1/2 & \cdots & 1/n \\ 1/2 & 1/3 & \cdots & 1/(n+1) \\ \vdots & \vdots & \cdots & \vdots \\ 1/n & 1/(n+1) & \cdots & 1/(2n-1) \end{bmatrix}$$

Hilbert matrices are poorly conditioned, in fact the larger they are, the more poorly conditioned they are. They are used as examples and to test software, which is why MATLAB has a command for generating them. To see some, type **hilb(3), hilb(4)** .

Instructions: To begin, type **format short, format compact** . That second command will suppress some of the blank lines that usually appear between MATLAB calculations, so more information will fit on the screen at the same time.

Insert a diskette into your a: drive and type **diary a:roundoff** . This opens a text file called roundoff on your diskette. Everything that appears on the screen from now on will be stored there, until either you close that file or exit MATLAB. Thus the command **diary** provides a way to record your calculations so you can study them later. Now do questions 2-7. You will be asked to record some information in Tables 1 and 2 below.

2. (MATLAB) Type the following lines to create 8 matrices and 4 vectors. The command **rand(n,m)** creates an nxm matrix with random number entries. For the larger matrices, use semicolons as indicated so you do not have to look at so much data:

```
A1 = rand(5,5), A2 = rand(10,10); A3 = rand(20,20); A4 = rand(30,30);
H1 = hilb(5), H2 = hilb(10); H3 = hilb(20); H4 = hilb(30);
b1 = rand(5,1), b2 = rand(10,1); b3 = rand(20,1); b4 = rand(30,1);
```

3. (MATLAB) Solve the equation $A_1\mathbf{x} = \mathbf{b}_1$, using three different methods as indicated below. Each method is theoretically correct but you will see that the solutions differ some, especially for the larger size matrices. The three methods are discussed some on page 4 below.

(a) For simplicity, rename the matrix and vector and store the size by typing

```
A = A1;  b = b1;  n = 5;
```

Then type the following three lines to solve the equation $A\mathbf{x} = \mathbf{b}$ three different ways and to calculate the residual vector for each method. Notice semicolons are used to suppress printing the vectors at first, then the solutions are printed side by side, and the residual vectors are printed side by side. For the larger vectors you will appreciate the economy of space and how easy it is to compare them when displayed this way.

```
x1 = A\b;  res1 = b - A*x1;
x2 = inv(A)*b;  res2 = b - A*x2;
P  = ref([A  b]);  x3 = P(:, n+1);  res3 = b - A*x3;
  [x1  x2  x3]
[res1  res2  res3]
```

(b) Type the following line and inspect to find the power of 10 that appears in these norms. In the **A1** column of Table 1 below, record these powers of 10.

```
[ norm(res1)  norm(res2)  norm(res3) ]
```

(c) Repeat the calculations above for $H_1\mathbf{x} = \mathbf{b}_1$. First type **A = H1**. To avoid having to retype all the other MATLAB commands, just press the up arrow key until you find the line you want next, and press [Enter] to execute it again.

4. (MATLAB) Repeat the instructions in question 3 for $A_2\mathbf{x} = \mathbf{b_2}$ and $H_2\mathbf{x} = \mathbf{b_2}$, $A_3\mathbf{x} = \mathbf{b_3}$ and $H_3\mathbf{x} = \mathbf{b_3}$, $A_4\mathbf{x} = \mathbf{b_4}$ and $H_4\mathbf{x} = \mathbf{b_4}$. For example, A_2 has a different shape than A_1, so you need to first type

A = A2; b = b2; n = 10;

and then do the calculations. Again you can use the up arrow key to avoid having to retype the other commands.

Table 1. Power of 10 in the norm of each residual vector

	A1	A2	A3	A4	H1	H2	H3	H4
norm(res1)								
norm(res2)								
norm(res3)								

Table 2. Warnings about poor conditioning?

	A1	A2	A3	A4	H1	H2	H3	H4
x1								
x2								

Instructions continued. This completes the numerical calculations for this project. To answer the rest of the questions, you will need to inspect your diary file.

First type **diary off** to close the file called roundoff on your diskette. Use your favorite text editor to read it and answer questions 5 - 7. (If you prefer, you could print the file instead of just reading it on the screen. However it will be fairly long.)

5. Find the calculation of **x1** and **x2** for each different matrix. Look carefully to find which matrices yielded a warning about poor conditioning after the calculation of **x1** or **x2**. List those matrices:

Also, record all the warnings about poor conditioning that appeared, by putting "yes" in the appropriate places in Table 2 above. See remarks below about why **ref** (i.e., the calculation of **x3**) will never print a warning.

6. Examine the residual vectors **res1** and **res2** for each coefficient matrix. Which coefficient matrices yielded smaller residual vectors, and which ones yielded larger residuals? Give your two lists:

7. Discuss the matrices you listed in questions 5 and 6. Are there some matrices that you think should have triggered warnings about poor conditioning but did not? Which ones were they and why do you think that?

Notes about the three solution methods. The following notes compare the three methods used above. Notice we are discussing only the solution of systems $A\mathbf{x} = \mathbf{b}$ when A is square and invertible.

MATLAB's backslash function \ uses the LU factorization algorithm with partial pivoting (see Section 2.5 of Lay's text). This is the method people usually choose for professional scientific computing because it does the fewest arithmetic operations, hence is efficient, and because partial pivoting can greatly minimize roundoff error. Backslash gives quite accurate results for most problems.

The method using **inv** does more arithmetic than backslash: first A^{-1} is calculated, by solving $A\mathbf{x} = \mathbf{e}_i$ for each column $\mathbf{e}_i$ of I, then the product $A^{-1}\mathbf{b}$ is calculated. So it takes more time than backslash, and does more arithmetic, which is likely to create more roundoff error.

The **ref** function was written to help students learn linear algebra, not for professional use. It is pretty accurate for smaller matrices most of the time, but it is naïve because it does not check the condition of the coefficient matrix. Also, it can be slow for larger matrices.

Notes about Hilbert matrices.

The nxn Hilbert matrix is $H = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n-1} \end{bmatrix}$. Theoretically, every Hilbert matrix is invertible, but they are notoriously bad for floating point calculations. The bigger Hilbert matrix H you use, the less accurate will be the solution to $H\mathbf{x} = \mathbf{b}$ when calculated with floating point arithmetic. Some algorithms may appear to give somewhat more accurate results than others, but none will be very good for a big Hilbert matrix. The problem is in the matrix itself. Hilbert matrices are classic examples of poorly conditioned matrices which is why people use them as test cases for algorithms and why MATLAB has the command **hilb** for generating them.

MATLAB actually has a function **invhilb** for calculating an exact inverse for a Hilbert matrix! You might enjoy comparing **invhilb(n)** and **inv(hilb(n))** for several values of n. The **inv** function does floating point calculations, so it does not produce an exact inverse – in fact, when n is around 12 or larger, you will see that **inv(hilb(n))** produces a matrix which has very few entries even close to what they should be. Type **help hilb** and **help invhilb** for more information.

Notes about condition number.

The formal definition of *condition number* of a matrix A is $\|A\|\,\|A^{-1}\|$. (See Section 7.4 of Lay's text. The notation $\|B\|$ denotes the norm of the matrix B, and it is defined by $\|B\|$ = maximum of all $\|B\mathbf{x}\|$, taken over all $\|\mathbf{x}\| = 1$. So the value of $\|B\|$ can be thought of as a measure of how much the mapping B distorts lengths in $\mathbf{R}^n$.) We say a matrix is *poorly conditioned* if the condition number is quite large. In practice, what people consider "large" is a matter of judgment and depends on the size of the matrix and other things.

In MATLAB you can estimate the condition of a matrix by typing **cond(A)**. Instead of actually calculating A^{-1} and the product of the norms, a clever algorithm is used to get a quick estimate. In fact, MATLAB actually estimates the reciprocal of the condition of A and it reports "RCOND." So when RCOND is very small, that means A is poorly conditioned.

The most important thing for you to remember about solving linear systems is this: any time professional software prints a warning about poor conditioning, be skeptical of the accuracy of the calculated results. Consult a numerical analyst or references to see how the answer might be improved in such cases. Sometimes you can get a better answer by increasing the number of digits used in the calculations. A better idea, if it works, is to reformulate the problem somehow to get a better conditioned coefficient matrix.

MATLAB Project: Partitioned Matrices Name_____

Purpose: To investigate multiplication of partitioned matrices.
Prerequisite: Section 2.4
MATLAB functions used: **+ , - , * , eye, inv** and **partdat** from Laydata Toolbox

1. (a) (MATLAB) To get the matrices below, type **partdat**. (They will be named **A11, A12,** etc.)

$$A_{11}=\begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix},\ A_{12}=\begin{bmatrix} 4 & 1 & -1 \\ -2 & 0 & 3 \end{bmatrix},\ A_{13}=\begin{bmatrix} -2 \\ 1 \end{bmatrix},\ A_{21}=\begin{bmatrix} 2 & 0 \\ -1 & 1 \\ 5 & 3 \end{bmatrix},\ A_{22}=\begin{bmatrix} 1 & 1 & 2 \\ 1 & 1 & 3 \\ 1 & 2 & 3 \end{bmatrix},\ A_{23}=\begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix},$$

$$A_{31}=[\ 9\ \ -2],\ A_{32}=[\ -5\ \ 3\ \ 4],\ A_{33}=[\ 1],\ C_{11}=\begin{bmatrix} -2 & 1 \\ 1 & 2 \end{bmatrix},\ C_{21}=\begin{bmatrix} 5 & 0 \\ -1 & 3 \\ 4 & -2 \end{bmatrix},\ C_{31}=[\ 4\ \ 7]$$

Then create $A=\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$ and $C=\begin{bmatrix} C_{11} \\ C_{21} \\ C_{31} \end{bmatrix}$ by typing the following lines:

A = [A11 A12 A13; A21 A22 A23; A31 A32 A33]
C = [C11; C21; C31]

Record A and C and put in dotted lines to <u>mark the partitions</u>:

$A =$ $C =$

(b) Calculate AC using partitioned multiplication. The following lines calculate each block; record the result of each:

A11*C11 + A12*C21 + A13*C31 Result =

A21*C11 + A22*C21 + A23*C31 Result =

A31*C11 + A32*C21 + A33*C31 Result =

Now write the matrix AC and <u>mark the partitions</u> corresponding to the blocks of A and C.

$AC =$

2. (hand) Using the matrices in question 1:

(a) Can you calculate AC as $\begin{bmatrix} C_{11}A_{11} + C_{21}A_{12} + C_{31}A_{13} \\ C_{11}A_{21} + C_{21}A_{22} + C_{31}A_{23} \\ C_{11}A_{31} + C_{21}A_{32} + C_{31}A_{33} \end{bmatrix}$? Why or why not?

(b) Can you calculate AC as $\begin{bmatrix} A_{11} \\ A_{21} \\ A_{31} \end{bmatrix} C_{11} + \begin{bmatrix} A_{12} \\ A_{22} \\ A_{32} \end{bmatrix} C_{21} + \begin{bmatrix} A_{13} \\ A_{23} \\ A_{33} \end{bmatrix} C_{31}$? Why or why not?

3. (hand) Now let $A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$ denote any partitioned matrix in which A_{11} and A_{22} are square and invertible. We want formulas for X and Y so that $\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} I & X & O \\ O & I & Y \\ O & O & I \end{bmatrix} = \begin{bmatrix} B_{11} & O & B_{13} \\ B_{21} & B_{22} & O \\ B_{31} & B_{32} & B_{33} \end{bmatrix}$ is true. Assume X and Y are sizes for which $A_{11}X$ and $A_{12}Y$ make sense, and I and O denote identity and zero matrices of the appropriate sizes for where they appear.

(a) Find X and Y in terms of the A_{ij}. Do the minimal amount of calculation and show work.

(b) Find an additional condition on the blocks A_{ij} so that $B_{13} = O$. (Be careful -- don't write A_{12}^{-1}. Why not?) Show calculations:

4. (MATLAB) Apply what you found in question 3 to the particular partitioned matrix A given in question 1. That is, use your formulas from question 3(a) to calculate X and Y, and then create $D = \begin{bmatrix} I & X & O \\ O & I & Y \\ O & O & I \end{bmatrix}$. We show a way to calculate X and D; you write a command to calculate Y:

X = -inv(A11)*A12
Y =
D = eye(6), D([1 2], [3 4 5]) = X, D([3 4 5], 6) = Y

Record X and Y. Type **A*D** to calculate AD and mark the partitions in AD. Are the appropriate blocks zero?

$X =$ $Y =$ $AD =$

MATLAB Project: Schur Complements Name_____

Purpose: To learn what Schur complements are and their connection with row reduction.
Prerequisite: Section 2.4
MATLAB functions used: **inv, eye, zeros, - , * ;** and **schurdat** from Laydata Toolbox

Background. This is based on Exercise 15 in Section 2.4. Let $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ be a partitioned matrix in which A_{11} is square and invertible. Define the Schur complement of A_{11} in A to be $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$. Here you will see three ways to calculate the matrix $\begin{bmatrix} A_{11} & A_{12} \\ O & S \end{bmatrix}$, where O is the zero matrix having the same shape as A_{21}.

1. (MATLAB) Type **schurdat** to get $A_{11} = \begin{bmatrix} 0 & -1 \\ 1 & 3 \end{bmatrix}$, $A_{12} = \begin{bmatrix} 4 & 1 & -1 \\ -2 & 0 & 3 \end{bmatrix}$, $A_{21} = \begin{bmatrix} 2 & 0 \\ -1 & 1 \end{bmatrix}$, and $A_{22} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$. In MATLAB they will be stored as **A11, A12, A21, A22.**

(a) Type **A = [A11 A12; A21 A22]** to create A. Inspect to see that this does look like $\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$, and use dotted lines to mark the partitions in A:

$A =$

(b) One way to get the Schur complement S for the matrix here would be to just calculate it directly, using the definition above. Type the following line to do that, and record the result:

S = A22 - A21*inv(A11)*A12

$S =$

2. (hand) Assume now that $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ is any partitioned matrix in which A_{11} is square and invertible. Let L, I and O be of appropriate sizes so that $\begin{bmatrix} I & O \\ L & I \end{bmatrix}\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ is defined. Find a formula for L, in terms of the A_{ij}'s, so that $\begin{bmatrix} I & O \\ L & I \end{bmatrix}\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ O & S \end{bmatrix}$ is true; show work:

3. (MATLAB) A second way to get the Schur complement S of A_{11} in A to would be to use your formula from question 2 to calculate L, then create $C = \begin{bmatrix} I & O \\ L & I \end{bmatrix}$, and then calculate CA. Do this for the matrix in question 1. You figure out a command to calculate L and record your command below. A way is shown to create C and CA:

L =
C = [eye(2) zeros(2,2); L eye(2)]
C*A

Inspect CA to verify that it does look like $\begin{bmatrix} A_{11} & A_{12} \\ O & S \end{bmatrix}$, where S is the same matrix as you got in questions 1 and 2. Record results.

$L =$ $C =$ $CA =$

4. (hand) A third way to calculate the Schur complement S of A_{11} in A is to use row operations in a special way. This method actually does the least arithmetic so is the most efficient method. The idea is: don't change anything in $[A_{11}\ A_{12}]$ but just add multiples of appropriate rows of $[A_{11}\ A_{12}]$ to rows of $[A_{21}\ A_{22}]$ so as to create a block of zeros below A_{11}. Notice this is not the usual Row Reduction Algorithm because nothing will change in the top block $[A_{11}\ A_{12}]$.

Verify that this method works for the matrix A from question 1. The first step is shown; you finish. Record each matrix you create and inspect your final matrix to be sure it does look like $\begin{bmatrix} A_{11} & A_{12} \\ O & S \end{bmatrix}$:

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

5. (hand) Assume now that $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ is any partitioned matrix in which A_{11} is invertible. Prove that the method described in question 5 will always work, to get a block of zeros below A_{11}. That is, if A_{11} is invertible, you can always do row operations to A as described in question 5, to get the form $\begin{bmatrix} A_{11} & A_{12} \\ O & W \end{bmatrix}$. Also explain why the block W obtained this way must be the Schur complement of A_{11} in A. Attach an extra sheet. (Hint: you must use the invertibility of A_{11} somehow!)

MATLAB Project: LU Factorization Name_____

Purpose: To practice Lay's LU Factorization Algorithm and see how it is related to MATLAB's **lu** function.

Prerequisite: Section 2.5

MATLAB functions used: *, **lu**; and **ludat** and **gauss** from Laydata Toolbox

Background. In Section 2.5, read about Lay's algorithm for calculating an LU factorization; especially study Example 2. It is imperative you understand the algorithm for calculating the matrix L before starting. In this project you will perform his algorithm on the matrices below, and see the connection between his algorithm and the one used by MATLAB's **lu** function.

$$A = \begin{bmatrix} -5 & 3 & 4 \\ 10 & -8 & -9 \\ 15 & 1 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 15 & 1 & 2 \\ 10 & -8 & -9 \\ -5 & 3 & 4 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 3 & -5 & -3 & 0 \\ 0 & -2 & 3 & 1 & -1 \\ 0 & -10 & 15 & 5 & -5 \\ 0 & 2 & -3 & -1 & 1 \\ 1 & 1 & -2 & -2 & -1 \end{bmatrix} \qquad D = \begin{bmatrix} 2 & -4 & -2 & 4 \\ 6 & -9 & -3 & 7 \\ -1 & -4 & 0 & 8 \end{bmatrix}$$

$$E = \begin{bmatrix} 2 & 6 & -1 \\ -4 & -9 & -4 \\ -2 & -3 & 0 \\ 4 & 7 & 8 \end{bmatrix} \qquad F = \begin{bmatrix} 1 & 3 & 0 \\ 4 & 4 & 8 \\ 1 & 2 & 3 \end{bmatrix}$$

To begin, type **ludat** to get the matrices above. For each matrix, use **gauss** to reduce the matrix to echelon form; **gauss** does only "add a multiple of one row to another" type operations -- no scaling or row exchanges. When used as shown below, **gauss** zeros out the entries directly below each successive pivot position. Type **help gauss** to learn more about how this function can be used.

1. (MATLAB) For each matrix above, use **gauss** and the algorithm in Section 2.5 to calculate an LU factorization. Record the matrix gotten from each **gauss** step; inspect those to write *L*; and finally, verify that *LU* does equal the original matrix (where *U* is the final matrix in your reduction).

(a) Here is the solution for *A*; notice we store each intermediate matrix as *U*, but the final *U* is the one we really want:

U = A	(copy *A* so you can keep the original matrix and work with the copy)
U = gauss(U,1)	(pivot on the first nonzero entry in row one)
U = gauss(U,2)	(pivot on the first nonzero entry in row two)

The matrices produced by the commands above:

$$A = \begin{bmatrix} -5 & 3 & 4 \\ 10 & -8 & -9 \\ 15 & 1 & 2 \end{bmatrix} \sim \begin{bmatrix} -5 & 3 & 4 \\ 0 & -2 & -1 \\ 0 & 10 & 14 \end{bmatrix} \sim \begin{bmatrix} -5 & 3 & 4 \\ 0 & -2 & -1 \\ 0 & 0 & 9 \end{bmatrix} = U$$

Inspecting the matrices on the left gives *L*:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & -5 & 1 \end{bmatrix}$$

The following lines will store *L* and allow you to verify that *LU* does look like *A*:

```
L = [1 0 0;-2 1 0;-3 -5 1]
L*U, A
```

Modify the commands above for the other matrices, and record the same type of information for each:

(b) $B = \begin{bmatrix} 15 & 1 & 2 \\ 10 & -8 & -9 \\ -5 & 3 & 4 \end{bmatrix} \sim$

(c) After two **gauss** steps on C you will see some zero rows. This simply means all multipliers are zero after that, so put zeros in the positions that remain unfilled in L. Remember that L should have 1's on its diagonal.

$$C = \begin{bmatrix} 1 & 3 & -5 & -3 & 0 \\ 0 & -2 & 3 & 1 & -1 \\ 0 & -10 & 15 & 5 & -5 \\ 0 & 2 & -3 & -1 & 1 \\ 1 & 1 & -2 & -2 & -1 \end{bmatrix} \sim$$

(d) $D = \begin{bmatrix} 2 & -4 & -2 & 4 \\ 6 & -9 & -3 & 7 \\ -1 & -4 & 0 & 8 \end{bmatrix} \sim$

(e) $E = \begin{bmatrix} 2 & 6 & -1 \\ -4 & -9 & -4 \\ -2 & -3 & 0 \\ 4 & 7 & 8 \end{bmatrix} \sim$

(f) $F = \begin{bmatrix} 1 & 3 & 0 \\ 4 & 4 & 8 \\ 1 & 2 & 3 \end{bmatrix} \sim$

2. (hand) Let A be the matrix from part 1 of this project and let $\mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$. Solve $A\mathbf{x} = \mathbf{b}$ with the method described in the beginning of section 2.5 of the text. The idea is that the following equations are equivalent: $A\mathbf{x} = \mathbf{b}$ <==> $LU\mathbf{x} = \mathbf{b}$ <==> $L\mathbf{y} = \mathbf{b}$ and $\mathbf{y} = U\mathbf{x}$. Thus, if you first solve $L\mathbf{y} = \mathbf{b}$ for y and then $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$, you will have the solution to $A\mathbf{x} = \mathbf{b}$. Do the calculations by hand, and show work:

Step 1. Solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$ by forward substitution:

Step 2. Using the vector $\mathbf{y}$ from Step 1, solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$ by back substitution:

Background on a new topic: MATLAB's **lu** function. There are two ways to use **lu**, and we will illustrate these with the matrix F above. Either way the result will not be an LU Factorization of the original matrix but rather of a different matrix PF, where P is a permutation matrix. These matters are discussed a little more in the Remarks at the bottom of page 4.

If you type **[L U P] = lu(F)**, this will not produce the factorization you got in 1(f). Instead, **lu** will first create

$PF = \begin{bmatrix} 4 & 4 & 8 \\ 1 & 3 & 0 \\ 1 & 2 & 3 \end{bmatrix}$, where $P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. It will then factor PF, obtaining $L = \begin{bmatrix} 1 & 0 & 0 \\ .25 & 1 & 0 \\ .25 & .5 & 1 \end{bmatrix}$ and $U = \begin{bmatrix} 4 & 4 & 8 \\ 0 & 2 & -2 \\ 0 & 0 & 2 \end{bmatrix}$.

The product LU will equal PF, not F.

Try this for yourself: type **[L U P] = lu(F)** to see these matrices, and then type **P*F, L*U** to verify for yourself that $PF = LU$ is true. Compare the L and U obtained here with those you got in 1(f).

If instead you type **[L1 U1] = lu(F)**, U_1 will be the same U as before but the matrix L_1 will be P^TL. This time L_1 is not lower triangular, but the product L_1U_1 will equal the original matrix F. Try this for yourself: type **[L1 U1] = lu(F)** and observe that L_1 equals P^TL, not L; then type **L1*U1** and verify that L_1U_1 does equal F.

3. Use the **lu** function as just described with the matrices A, B, and C discussed earlier, and record results. If necessary, recall these matrices by typing **ludat**. We do the first one as an example:

(a) For $A = \begin{bmatrix} -5 & 3 & 4 \\ 10 & -8 & -9 \\ 15 & 1 & 2 \end{bmatrix}$, first type **[L U P] = lu(A), L*U, P*A** and record results:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ .6667 & 1 & 0 \\ -.3333 & -.3846 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 15 & 1 & 2 \\ 0 & -8.6667 & -10.3333 \\ 0 & 0 & .6923 \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}; \quad LU = PA = \begin{bmatrix} 15 & 1 & 2 \\ 10 & -8 & -9 \\ -5 & 3 & 4 \end{bmatrix}$$

Then type **[L1 U1] = lu(A), L1*U1, A** and record results: $L_1 = \begin{bmatrix} -.3333 & -.3846 & 1 \\ .6667 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$, $U_1 = U$, $L_1 U_1 = A$

(b) $B = \begin{bmatrix} 15 & 1 & 2 \\ 10 & -8 & -9 \\ -5 & 3 & 4 \end{bmatrix}$

(c) $C = \begin{bmatrix} 1 & 3 & -5 & -3 & 0 \\ 0 & -2 & 3 & 1 & -1 \\ 0 & -10 & 15 & 5 & -5 \\ 0 & 2 & -3 & -1 & 1 \\ 1 & 1 & -2 & -2 & -1 \end{bmatrix}$

Remarks. A permutation matrix P is one obtained by rearranging the rows of an identity matrix. The effect of calculating PA is to produce that same rearrangement of the rows of A.

The algorithm used in **lu** is called "partial pivoting," and this is what causes the creation of PA. Using partial pivoting typically improves the accuracy of row operations, so all professional software for solving linear systems, like MATLAB's **lu** and backslash functions, does partial pivoting.

MATLAB Project: An Economy With an Open Sector Name_____

Purpose: To study a linear system model of an open sector economy.
Prerequisite: Section 2.6
MATLAB functions used: **-** , **sum, eye**; and **ref** from Laydata Toolbox

Background. This is based on Exercise 13 in Section 2.6, where an economy with 7 production sectors is described. Each sector produces goods and each uses some of the output of the other sectors. There is also an open sector, i.e., a sector which only consumes. The consumption matrix will be called C, and **d** will denote the demand vector for the open sector. When C and **d** are given, a solution **x** to the equation $\mathbf{x} = C\mathbf{x} + \mathbf{d}$ is called a production vector. If there is a solution **x** with all entries nonnegative, then this economy is possible and could exist. Notice the equation $\mathbf{x} = C\mathbf{x} + \mathbf{d}$ can be rewritten as $(I - C)\mathbf{x} = \mathbf{d}$.

When the matrix C has each entry nonnegative and each column sum less than one, Theorem 11 guarantees that $I - C$ will be invertible and that the economy is possible for any nonnegative demand vector **d**—that is, the unique vector **x** which satisfies $(I - C)\mathbf{x} = \mathbf{d}$ will also be nonnegative. (The proof of Theorem 11 shows why: for such C, all entries of the matrix $(I - C)^{-1}$ are nonnegative, so when **d** has nonnegative entries, $\mathbf{x} = (I - C)^{-1}\mathbf{d}$ must also have nonnegative entries.)

1. (a) Type the following lines to get the data for C and **d** and to calculate the column sums of C. Inspect to be sure each entry of C and **d** is nonnegative and that each column sum of C is less than one:

c2s6
13
sum(C) (**sum(C)** yields a row vector containing the sum of each column)

Type the following lines to create $I - C$ and to solve the equation $(I - C)\mathbf{x} = \mathbf{d}$.

M = eye(7) - C (**eye(7)** creates a 7x7 identity matrix)
R = ref([M d]), x = R(:, 8) (**R(:, 8)** is column 8 of the matrix R)

Record **d** and **x** below. Notice the display of **x** has the expression **1.0e+005 *** above a column of numbers. Don't ignore that! It means each number in the column is multiplied by 10^5.

Choose two more nonnegative demand vectors **d1** and **d2** and solve for the production vector for each. Record all these vectors.

d = **x =** **d1 =** **x1 =** **d2 =** **x2 =**

(b) Using Section 2.6 Problem 13, interpret what the (3,1) entry of **x** means.

(c) Discuss: why does it matter that the entries of the solution **x** should be nonnegative, in an economic model like the above?

2. (a) Experiment to increase the (1,1) entry of C, until you find a new consumption matrix that gives a solution with some negative entries – that is, until solving $(I - C)\mathbf{x} = \mathbf{d}$ yields some negative entries in $\mathbf{x}$.

Remember that to change the (1,1) entry of C to, say, .16, type **C(1,1) = . 16**. Then use the arrow up key to repeat the commands for **M** and **R** to find **x**.

Record the results of your experimentation below.

New value of the (1,1) entry of C: _____ **x** =

(b) Discuss: What do the numbers in your new matrix C say about the economy? That is, why does it seem reasonable that your new matrix is not a valid consumption matrix?

MATLAB Project: Matrix Inverses and Infinite Series Name_____

Purpose: To see examples for which the matrix series $I + C + C^2 + C^3 + \ldots$ does converge to $(I - C)^{-1}$ and examples for which it does not.

Prerequisite: Section 2.6

MATLAB functions used: *, +, :, **eye**, **for**, **end**, **format**; and Laydata Toolbox

Background. By Theorem 11 in Section 2.6, the series $I + C + C^2 + C^3 + \ldots$ does converge to $(I - C)^{-1}$ when each entry of C has nonnegative entries and each column sum is less than one.

It is also true that the series will not converge for some matrices. You will check out these facts here with some examples.

1. Use the matrix C which is defined in Exercise 13, Section 2.6. Type the following lines to get C and to calculate $I + C + C^2 + \ldots + C^k$ for several values of k.

```
c2s6
13
I = eye(7); S = I;
S = I + C*S
```

Use the up arrow key to execute the sum line repeatedly. Keep count as to how many times the sum line is repeated and watch to see that this series does seem to converge. (The first time you execute this line, you get $S = I + C$; the second time you get $I + C + C^2$; etc.) How many times must you repeat it until the matrix S seems to stop changing, at least as far as what you see on the screen? _____

Definitions. The *norm of a vector* $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is defined to be $\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$. For example, if $\mathbf{x} = [-4\ 2\ -2\ 1]$, then $\|\mathbf{x}\| = 5$. Clearly, norm is a way to measure the size of a vector, and it is reasonable to call a vector small if its norm is small. Notice that when a vector has only 2 or 3 entries, this is the same definition of length you saw in analytic geometry.

The *norm of a matrix* A, often written norm(A), is defined to be the maximum of $\|A\mathbf{x}\|$, taken over all $\mathbf{x}$ such that $\|\mathbf{x}\| = 1$.

In MATLAB, to calculate the norm of a vector stored as **x**, type **norm(x)**. To calculate the norm of a matrix stored as **A**, type **norm(A)**.

2. One way to check whether S is close to the inverse of $I - C$ is to see whether the norm of $(I - C)S - I$ is small. Experiment with different values of k to find how many terms of the series you must use in $S = I + C + \ldots + C^k$ in order to get the norm of $(I - C)S - I$ to be 10^{-10} or smaller. The following lines do this for k = 10. We assume that I is still in your workspace.

```
format short e
k = 10;
S = I; for i = 1:k, S = I + C*S; end
diff = (I - C)*S – I
norm(diff)
```

Do the same calculations for k = 20 and k = 30. Record the norm of $(I - C)S - I$ for each, in the following table. Try larger values of k until you find one for which the norm of $(I - C)S - I$ is less than 10^{-10}, and record that data also:

k (number of terms used)	10	20	30			
Norm of *(I-C)S-I*						

3. Find a 2x2 matrix C with nonnegative entries and some column sum greater than one, but for which it still appears that the series $I + C + C^2 + C^3 + \ldots$ converges to $(I - C)^{-1}$. Look for a simple 2x2 example!

(a) Record your new matrix: $C =$

(b) Show what $I + C + C^2 + \ldots + C^n$ looks like for n = 10 and n = 20, for your C:

4. Find a matrix C with nonnegative entries for which $I + C + C^2 + \ldots$ definitely does not converge to $(I - C)^{-1}$. Again, look for a simple 2x2 example!

(a) Record your new matrix: $C =$

(b) Why are you certain that $I + C + C^2 + \ldots$ does not converge to to $(I - C)^{-1}$ this time?

MATLAB Project: Homogeneous Coordinates for Computer Graphics Name_____

Purpose: To practice using homogeneous coordinates to accomplish translations and other transformations of $\mathbf{R}^2$.

Prerequisite: Section 2.7

MATLAB functions used: **+** , ***** , **cos**, **sin** ; and **coordat** and **drawpoly** from Laydata Toolbox

Background: Read about homogeneous coordinates and study the Practice Problem in Section 2.7.

Example. Exercise 7 in Section 2.7 asks you to use homogeneous coordinates and find a 3x3 matrix M which rotates $\mathbf{R}^2$ through $\pi/3$ about the point (6,8). As shown in the Practice Problem, this can be done easily in three steps and $M = T_2RT_1$ is the desired matrix:

(i) Translate so that (6,8) goes to (0,0). The matrix which does that is $T_1 = \begin{bmatrix} 1 & 0 & -6 \\ 0 & 1 & -8 \\ 0 & 0 & 1 \end{bmatrix}$.

(ii) Rotate through $\pi/3$, about (0,0). The matrix is $R = \begin{bmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$ where $c = \cos(\pi/3)$ and $s = \sin(\pi/3)$.

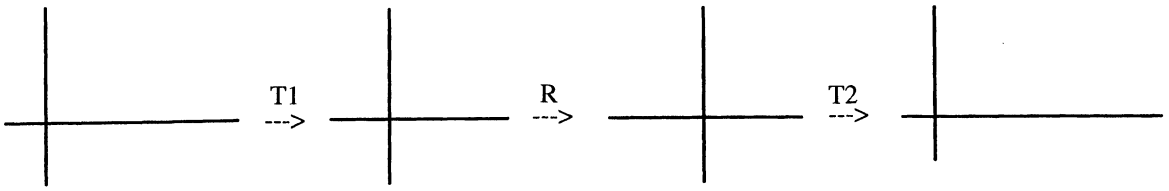
(iii) Translate so that (0,0) goes to (6,8). The matrix is $T_2 = \begin{bmatrix} 1 & 0 & 6 \\ 0 & 1 & 8 \\ 0 & 0 & 1 \end{bmatrix}$.

1. (MATLAB) In this project you will use the M-file **drawpoly**, which draws polygons. To begin, type **coordat** to get the matrices above and three others, $box = \begin{bmatrix} 6 & 7 & 7 & 6 & 6 \\ 8 & 8 & 9 & 9 & 8 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$, $tri = \begin{bmatrix} 2 & 2 & 3 & 2 \\ 2 & 3 & 3 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}$ and $box2 = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$.

The matrix *box* contains homogeneous coordinates for the vertices of the unit square whose lower left vertex is at (6,8). A figure like this can be useful because sketching the effect of each successive transformation on it can help you can see whether your matrices do accomplish the geometric effect you want. Type **drawpoly(box)** to see this square.

(a) To calculate the homogeneous coordinates of each new figure created by applying T_1, R and T_2 successively to that square, type: **v = T1*box, w = R*v, z = T2*w** .

Then type **drawpoly(box, v, w, z)** to plot the original square and each transformation of it. There will be a pause after each figure is graphed. Examine the figure, be sure you understand why it looks like it does, and sketch the result below. Then press [Enter] to see the next figure.



(b) Type **M = T2*R*T1** to calculate the product matrix $M = T_2RT_1$ and record it: $M =$

Also type **drawpoly(box, M*box)** to be sure M to verify that the product matrix M does accomplish the same final effect, in one step.

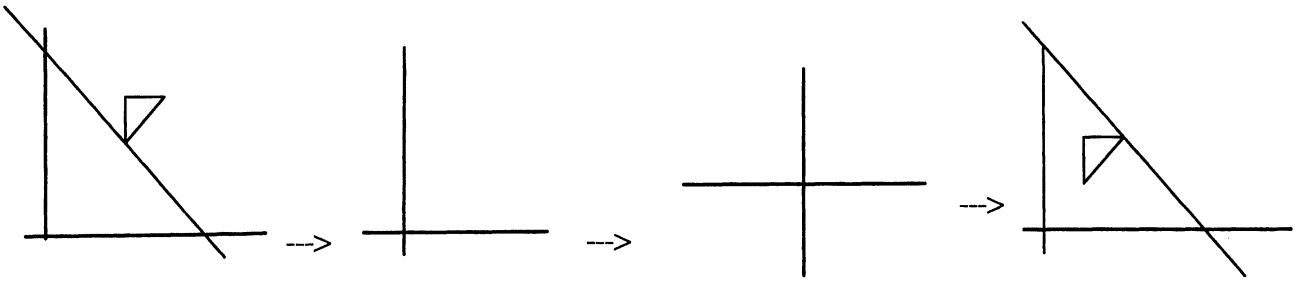
2. (hand) Discuss example 1. Specifically:

(a) Why are homogeneous coordinates needed here? That is, why is it not possible to rotate $\mathbf{R}^2$ about a point like (6,8) using regular coordinates for points in $\mathbf{R}^2$ and a 2x2 matrix?

(b) Why are the translation steps done? That is, how does translation make the solution easier than looking directly for a 3x3 matrix M that accomplishes the desired rotation?

3. Consider the sketch below. On the leftmost axis system are sketched the line $x + y = 4$ and the triangle with vertices (2,2), (2,3) and (3,3). The coordinates of this triangle are stored in the matrix *tri* .

(a) (hand) Use ideas like those in question 1 above to find 3x3 matrices T_1 , R , and T_2 which translate, reflect and translate so that applying them in succession to homogeneous coordinates will ultimately reflect $\mathbf{R}^2$ across the line $x + y = 4$. The effect of this reflection on the triangle is shown in the rightmost axis system. In the diagram, record each of your matrices below the appropriate arrow and sketch the successive images of the line and the triangle:



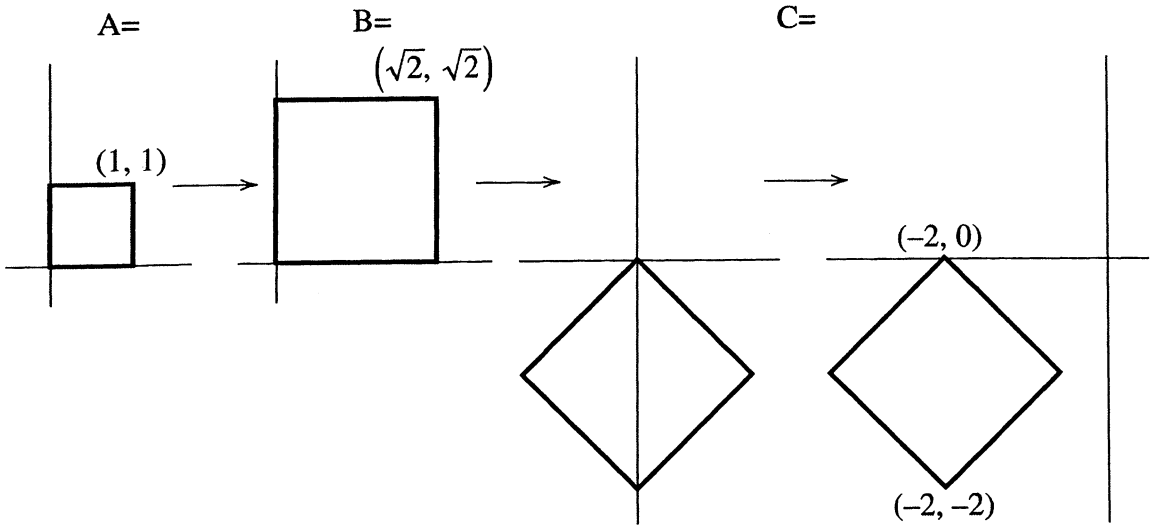
T1 = R= T2 =

(b) (MATLAB) Store the matrices that you defined in this problem, as **T1**, **R** and **T2**. Then type **M = T2*R*T1** to calculate their product. Record it: $M =$

Type **drawpoly(tri)** to check that *tri* contains homogeneous coordinates for the vertices of the small triangle in the first sketch above. To verify that your matrices perform as desired, type **drawpoly(tri, T1*tri, R*T1*tri, T2*R*T1*tri)** to check that each successive figure is what you intended, and then **drawpoly(tri, M*tri)** to verify that the product matrix M does accomplish the same final effect, in one step.

3. Consider the sketch below. The standard unit square is shown on the left.

(a) (hand) Find 3x3 matrices A, B and C so that applying your new matrices in succession to homogeneous coordinates of that square will successively transform it as shown. Write in each of your matrices:



(b) (MATLAB) Store your new matrices *A*, *B* and *C* and type **M = C*B*A** to calculate the product $M = CBA$. This is the composition of the three functions you created. Record it: $M =$

The matrix *box2* contains homogeneous coordinates for the standard unit square. So to verify that your matrices perform as desired, type **drawpoly(box2, A*box2, B*A*box2, C*B*A*box2)** and check to see that each successive figure is as shown above. Also type **drawpoly(box2, M*box2)** to verify that this single matrix transformation creates in one step the same final result.

MATLAB Project: Subspaces Name_____

Purpose: To deepen your understanding of span, basis and dimension. In particular, to understand what is required for two subspaces of R^n, which have the same dimension, to be the same sets.

Prerequisites: Section 4.6 or 2.9

MATLAB functions used: **rank, diary**; and **ref** and **submats** from Laydata toolbox

Remarks: Your instructor will supply a pair of matrices A and B, each having five rows. (There are such pairs in the file **submats**, and you may be assigned one of those.)

Question 1 is easy, but you will need to think how to answer question 2. Discuss it with each other — this can really help. Once you figure out a method it will not take long to do the calculations. Observe that Col A and Col B are obviously subspaces of $\mathbf{R}^5$.

Directions:

Use the matrices A and B which your instructor supplies. Employ MATLAB to do whatever calculations you need and attach the results. Explain your methods briefly and why they work. No credit unless your methods and explanation are valid!

One way to record your work is to just copy the key calculations by hand. An easier way is to create a diary file of your MATLAB session and print that after you finish all calculations.

If you want to create a diary file on a floppy diskette, here is a way: start MATLAB and type **diary a:subsp** before doing any calculations. This will cause everything that appears on the screen after that to be stored in a text file called subsp on your a: drive. When your calculations are finished, type **diary off** (or exit MATLAB) to close the file. Then use your favorite editor to print the file subsp. If you want, you can first clean up the file, add titles, etc. before printing it.

1. Verify that Col A and Col B have the same dimension.

2. Determine whether or not Col A and Col B are the same subspace of $\mathbf{R}^5$. Explain what you calculated and why it worked.

Notice this is not obvious. For example, if two subspaces of $\mathbf{R}^3$ each have dimension 1, each will be a line through the origin, but they might not be the same line. If each has dimension 2, they are planes through the origin, but they might not be the same plane. In general if two subspaces of $\mathbf{R}^n$ have the same dimension k, we can visualize each as looking like $\mathbf{R}^k$ -- but they might not be the same sets. Your job here is to figure out a way to decide if two subspaces of $\mathbf{R}^n$, which have the same dimension, are actually the same set of points, and apply your method to the subspaces Col A and Col B.

MATLAB Project: Markov Chains and Long Range Predictions Name_____

Purpose: To analyze several Markov chains and investigate steady state vectors.
Prerequisite: Section 4.9
MATLAB functions used: *, ^ , - , / , **eye, sum** ; **markdat** and **ref** from Laydata Toolbox

Background. Read Section 4.9 in the text, about Markov chains.

1. Read Exercises 2 and 12 in Section 4.9; they concern a Markov chain with the system matrix P shown below. In these exercises there are three foods and the i,j entry of P is the probability that if an animal chooses food j on the first trial, then it will choose food i on the second trial. Therefore the i,j entry of P^2 is the probability that if an animal chooses food j on the first trial, it will choose food i on the third trial.

(a) Type **markdat** to get the data for this project. The matrix for exercises 2 and 12 is called P and is shown below. Type **P^2** to calculate P^2 and record:

$$P = \begin{bmatrix} .50 & .25 & .25 \\ .25 & .50 & .25 \\ .25 & .25 & .50 \end{bmatrix} \qquad P^2 =$$

(b) (hand) Suppose an animal chooses food #1 on the first trial. Use P and P^2 to answer: what is the probability the animal will:

Choose food #2 on the second trial: _____

Choose food #2 on the third trial: _____

Choose food #3 on the third trial: _____

(c) Type **I = eye(3), ref(P - I)** to calculate the reduced echelon form of $P - I$. Use this to write the general solution **x** to the system $(P - I)\mathbf{x} = \mathbf{0}$. Also, choose a nonzero value for the free variable and write a particular solution **w**. Record your results:

ref (P –I) = **x** = **w** =

(d) Type the following lines to calculate a steady state vector **q** for P:

w = [(your data)] (Use your **w** from above, which satisfies $P\mathbf{w} = \mathbf{w}$; store it as a column)
q = w/sum(w) (Divide **w** by the sum of its components)

Record the result: **q** =

(e) Explain why **q** is a probability vector. Also, type **sum(q), P*q** to verify that **q** satisfies $P\mathbf{q} = \mathbf{q}$.

(f) Explain why P is a regular stochastic matrix, and why **q** must be its unique steady-state vector. (Use the definition of regular, and Theorem 18.)

2. Read Exercise 4 in Section 4.9 and solve it as follows.

(a) Write the matrix W and the initial vector **v** describing weather "today" in Exercise 4. Be careful -- W should be stochastic and **v** should be a probability vector!

W = **v** =

(b) Store your vector **v** as a column and type **W*v** to calculate $W\mathbf{v}$.

Record $W\mathbf{v}$: Using this, what is the chance of bad weather tomorrow? _____

(c) Now store the new initial vector for Monday, $\mathbf{v} = \begin{bmatrix} 0 \\ .4 \\ .6 \end{bmatrix}$ and type **(W^2)*v**.

Record $W^2\mathbf{v}$: Using this, what is the chance of good weather on Wednesday? _____

(d) Calculate the steady state vector **q** for W using the method shown in question 1(c) above.

Record **q**: In the long run, what is the probability the weather will be good on a given day? _____

3. According to Theorem 18, when P is stochastic and regular, and **v** is any probability vector, the sequence of vectors $\mathbf{v}, P\mathbf{v}, P^2\mathbf{v},...$ will converge, and the limit vector will be the steady-state vector of P. In other words, when the power k is big enough, $P^k\mathbf{v}$ will look like the unique steady-state vector. This is not an efficient way to calculate the steady-state vector, but it is interesting to see sequences $\mathbf{v}, P\mathbf{v}, P^2\mathbf{v},...$ converge for a few examples.

Do this for both P and W. Use each of the initial vectors shown below and at least one more probability vector **v** of your own. For each **v**, calculate $P^k\mathbf{v}$ until you find a big enough k so that $P^k\mathbf{v}$ looks like the steady-state vector for P (compare to the steady state vectors you got in 1(d) and 2(d)). Repeat this for each **v** and W, and record the smallest value of k which is big enough in each case.

The following lines will create the first **v** and get you started searching for k, for the matrix P:

format long	(so you can see fifteen significant digits)
v = [1; 0; 0]	(store the first initial vector)
P^18*v, P^19*v	(calculate more P^k as necessary, until result looks like steady state vector)

	P (animal experiment)				W (weather experiment)			
Initial **v** =	$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} .2 \\ .6 \\ .2 \end{bmatrix}$	$\begin{bmatrix} .35 \\ .35 \\ .3 \end{bmatrix}$	$\begin{bmatrix} \\ \\ \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} .2 \\ .6 \\ .2 \end{bmatrix}$	$\begin{bmatrix} .35 \\ .35 \\ .3 \end{bmatrix}$	$\begin{bmatrix} \\ \\ \end{bmatrix}$
k =	----------	----------	----------	----------	----------	----------	----------	----------

4. Consider the following matrices: $P_1 = \begin{bmatrix} .7 & .2 & .6 \\ 0 & .2 & .4 \\ .3 & .6 & 0 \end{bmatrix}$, $P_2 = \begin{bmatrix} .7 & .2 & .6 \\ 0 & .2 & 0 \\ .3 & .6 & .4 \end{bmatrix}$, $P_3 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

(a) Which of P_1, P_2, P_3 are regular? Explain why each is or is not. You may need to do some calculations, and you can type **P1** to get P1 if **markdat** is still loaded on MATLAB.

(b) Type **format** to restore MATLAB's usual short form for display of numbers. Then calculate steady state vectors for P_1, P_2, and P_3, using the method in question 1(c) above. The matrices will be called **P1**, **P2** and **P3** in your workspace. Record the steady state vectors in the table below. Use Theorem 18 or some calculations to decide whether the steady state vector is unique.

	Steady state vector	Is steady state vector unique?	If $\mathbf{v} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$, does $P^k\mathbf{v}$ converge as k gets large? If not, what does happen?
P_1			
P_2			
P_3			

MATLAB Project: Real and Complex Eigenvalues Name_____

Purpose: To see examples of nonreal eigenvalues, and to learn how to use MATLAB's **eig** function.

Prerequisite: Section 5.5

MATLAB functions used: **eig**, *; and **cxeigdat** from Laydata Toolbox

1. (hand) Calculate the eigenvalues for each of the following 2x2 matrices. Show work. Read Section 5.5 in the text for examples.

(a) $U = \begin{bmatrix} 5 & 2 \\ 4 & 3 \end{bmatrix}$

(b) $V = \begin{bmatrix} 4 & 3 \\ -3 & 4 \end{bmatrix}$

(c) $W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

(d) $X = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$

(e) $Y = \begin{bmatrix} 0 & 0 & 5 \\ 1 & 0 & 0 \\ 0 & 5 & 4 \end{bmatrix}$. Hint: one eigenvalue is 5.

2. (hand) Create some new examples. For each of the 2x2 matrices U, V, W, and X, it is easy to change the sign of one entry so that the new matrix has nonreal eigenvalues if they were real before; or has real eigenvalues if they were not real before. Do this, record your new matrices, and show work for calculating their eigenvalues:

(a)

(b)

(c)

(d)

3. (MATLAB) Type **cxeigdat** to get the matrices used in question 1. For each matrix A, use **eig** to get a matrix D whose diagonal entries are the eigenvalues of A, and a matrix P whose columns are associated eigenvectors. Record P and D, and inspect D to be sure the eigenvalues produced by **eig** agree with what you calculated by hand in question 1. Study the proof of Theorem 5, Sec. 5.3 to understand why the important fact $AP = PD$ must true. (The proof also works for complex numbers.) Calculate AP and PD for each A, to verify this. The following lines will get you started: **[P D] = eig(U)**
U*P, P*D

(a) $U = \begin{bmatrix} 5 & 2 \\ 4 & 3 \end{bmatrix}$ $P =$ $D =$

(b) $V = \begin{bmatrix} 4 & 3 \\ -3 & 4 \end{bmatrix}$ $P =$ $D =$

(c) $W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ $P =$ $D =$

(d) $X = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ $P =$ $D =$

(e) $Y = \begin{bmatrix} 0 & 0 & 5 \\ 1 & 0 & 0 \\ 0 & 5 & 4 \end{bmatrix}$ $P =$ $D =$

MATLAB Project: Using Eigenvalues to Study Spotted Owls Name_____

Purpose: To use eigenvalues and eigenvectors to understand the dynamics of this population and determine experimentally the critical rate for survival of juveniles to subadults—the value which that rate must equal or exceed for the population to survive. An extra credit problem asks for a deeper theoretical investigation and an exact calculation of that critical value.

Prerequisites: Sections 5.5 and 5.6

MATLAB functions used: ***** , **** , **:**, **sum, abs, for, eig, plot**; and **owldat** from Laydata Toolbox

ATTACH YOUR PLOTS AND TURN IN WITH THIS PAPER

Background. The spotted owls have three distinct life stages: juvenile (first year), subadult (second year) and adult (third year and older). Let $\mathbf{x_k} = \begin{bmatrix} j_k \\ s_k \\ a_k \end{bmatrix}$ and $A = \begin{bmatrix} 0 & 0 & .33 \\ t & 0 & 0 \\ 0 & .71 & .94 \end{bmatrix}$ where j_k, s_k and a_k denote the number of owls in each stage in year k and $\mathbf{x_{k+1}} = A\mathbf{x_k}$. As you may have seen in the earlier computer exercise on this topic, the owl population seems to die out eventually if $t = .18$ and seems to increase eventually if $t = .30$.

Some complex numbers occur in the eigenvalues and eigenvectors of A; to understand these better, read Section 5.5.

Definition. A *dominant eigenvalue* of a matrix A is an eigenvalue λ_1 of A such that $|\lambda_1| \geq |\lambda_i|$ for all eigenvalues λ_i of A.

It is true that the special type of matrix we are discussing here has only one dominant eigenvalue, so we will speak of "the" dominant eigenvalue λ_1. In fact, for all the matrices here, λ_1 is actually real and positive, so $|\lambda_1| = \lambda_1$.

1. Here you will experiment with several values of t to see how the dominant eigenvalue changes as t increases, and will find the "critical value."

To begin, type **owldat** to get the matrix for $t = .18$, $A = \begin{bmatrix} 0 & 0 & .33 \\ .18 & 0 & 0 \\ 0 & .71 & .94 \end{bmatrix}$. (You will also get $\mathbf{x0} = \begin{bmatrix} 100 \\ 100 \\ 100 \end{bmatrix}$, for later use.) Then type the following lines:

eig(A) (a vector containing the eigenvalues of **A**)

abs(eig(A)) (a vector containing the modulus of each entry of **eig(A)**)

You will see that the largest magnitude entry is 0.9836, so that is λ_1. Record it in the table below, under $t = .18$.

Next type **A(2,1) = .19** and use the up arrow key to execute the two lines above again. Repeat this for each value of t shown in the table below, and record the dominant eigenvalue each time:

Survival rate juv→subadult	t=	.18	.20	.22	.24	.25	.26	.28	.30
Dominant eigenvalue of A:	λ								

(b) Of the values you used for t, which is the smallest one for which $\lambda_1 \geq 1$? _____ We will call this the "critical value" of t.

2. Now assign **A(2,1)** the value that you just found which causes t to have the critical value. Thus, the dominant eigenvalue λ_1 of your matrix A will be slightly larger than 1.

(a) Type **[V D] = eig(A)** , and record the columns of V below as $\mathbf{v_1}$, $\mathbf{v_2}$ and $\mathbf{v_3}$. Before doing that, notice the dominant eigenvalue of A, which we want to call λ_1, is the third diagonal entry of D, hence the third column of V is an eigenvector corresponding to λ_1. So record the third column of V as $\mathbf{v_1}$, and record the first two columns of V as $\mathbf{v_2}$ and $\mathbf{v_3}$:

$\mathbf{v_1} =$ $\mathbf{v_2} =$ $\mathbf{v_3} =$

Also record the eigenvalues of A that correspond to each of these columns:

$\lambda_1 =$ _____ $\lambda_2 =$ _____ $\lambda_3 =$ _____

(b) (hand) It is true that $\{ \mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3} \}$ is a basis for the vector space $\mathbf{C}^3$, so any vector can be written as a linear combination of $\mathbf{v_1}$, $\mathbf{v_2}$, $\mathbf{v_3}$.[1] Let $\mathbf{x_0}$ denote an initial vector and define $\mathbf{x_k}=\mathbf{A}\mathbf{x_{k-1}}$. Suppose c_1, c_2 and c_3 are scalars such that $\mathbf{x_0} = c_1\mathbf{v_1} + c_2\mathbf{v_2} + c_3\mathbf{v_3}$. Using this equation, explain what $\mathbf{x_k}$ will look like after k years, and why, if $c_1 \neq 0$, the population of owls will not die out. You must use the fact that $\lambda_1 \geq 1$.

(c) Let the initial vector be $\mathbf{x_0} = \begin{bmatrix} 100 \\ 100 \\ 100 \end{bmatrix}$. Type the following lines to rearrange the columns of V so the eigenvector corresponding to λ_1 is the first column, and then to solve $\mathbf{x_0} = c_1\mathbf{v_1} + c_2\mathbf{v_2} + c_3\mathbf{v_3}$ for the c_i's:

```
V = V(:, [3 1 2])      (Create V = [v1 v2 v3])
c = V\x0               (Solve Vc = x0 for c)
```

Record the coefficients: $c_1 =$ _____ $c_2 =$ _____ $c_3 =$ _____

Notice c_1 is not zero, so the owl population will definitely not die out when $\mathbf{x_0} = \begin{bmatrix} 100 \\ 100 \\ 100 \end{bmatrix}$ is the initial vector.

---

[1] The space $\mathbf{C}^3$ is much like $\mathbf{R}^3$: its elements are all triples of complex numbers, and its scalars are the complex numbers. Also, {(1,0,0), (0,1,0), (0,0,1)} is a basis for $\mathbf{C}^3$ so its dimension is three, hence any three independent vectors in $\mathbf{C}^3$ form a basis for the space. Finally, the vectors $\mathbf{v_1}$, $\mathbf{v_2}$, $\mathbf{v_3}$ found in question 2 are independent -- you can check that directly, or just notice that the three eigenvalues of A are distinct.

3. Continue to use $\mathbf{x_0} = \begin{bmatrix} 100 \\ 100 \\ 100 \end{bmatrix}$ as the initial vector. Choose two values of t: t_1 less than the critical value you found above, and t_2 greater than that critical value. Record the values you choose:

$t_1 =$ _____ $t_2 =$ _____

(a) Using $t = t_1$, calculate and plot the values of j_k, s_k and a_k from 1997 until 2020. The following commands will do these things for $t = t_1$:

A(2,1) = (your value for t_1)
x = x0; P = x; for i = 1997:2020, x = A*x; P = [P x]; end
yr = 1996:2020; plot(yr, P)

Print this graph. Type **help print** if you need instructions. Label the graph with your value of t_1 and label the individual curves "adult," "subadult," and "juvenile."

(b) Repeat the calculations, printing and labeling using your value of t_2.

(c) Discuss: what long term population trends do your graphs show in the three age groups when $t = t_1$, and what trends when $t = t_2$? Are these the results you expected, based on what you know about the dominant eigenvalue of the matrix A in each case?

4. (Extra credit, all hand work. Use your paper and attach.) Let $A = \begin{bmatrix} 0 & 0 & a \\ t & 0 & 0 \\ 0 & b & c \end{bmatrix}$, and assume a,b,c,t are positive.

(a) Let $f(\lambda)$ denote the characteristic polynomial of A. Calculate it and show work. You should get $f(\lambda) = -\lambda^3 + c\lambda^2 + abt$.

(b) Prove that A has only one real eigenvalue, that it is positive, and that the other two eigenvalues of A must be conjugate complex numbers. Let λ_1 denote the real positive eigenvalue and let λ_2 and λ_3 denote the other two eigenvalues.

Hint: Since $y = f(\lambda)$ has only real coefficients, you can sketch its graph in R^2. It will be helpful to calculate its y-intercept and to use the derivative to find the turning points. Use this graph to explain why there is only one real zero of $f(\lambda)$ and it is positive. Then use things you know about zeros of polynomials to explain why the other two zeros must be conjugate complex numbers.

(c) Prove that λ_1 is greater than $|\lambda_2| = |\lambda_3|$, hence the real positive eigenvalue of A will always be the dominant eigenvalue for this type matrix.

Hint: Explain first why $f(\lambda) = (\lambda_1 - \lambda)(\lambda_2 - \lambda)(\lambda_3 - \lambda)$ is true and use that to explain why $\lambda_1\lambda_2\lambda_3$ equals abt; explain next why $\lambda_2\lambda_3$ equals $|\lambda_2|^2$; thus $\lambda_1|\lambda_2|^2 = abt$; finally, explain why $\lambda_1^3 = c\lambda_1^2 + abt$ is true. Then put this information together.

(d) Assume $\lambda_1 = 1$ and use this to obtain a formula for the exact critical value of t. Evaluate your formula when a = .33, b = .71 and c = .94, and compare this with the critical value you found experimentally in question 1. Are they essentially the same?

Discuss what $\lambda_1 = 1$ means in the owl example. Does it mean no births or deaths? If not, what does it mean?

MATLAB Project: QR Factorization Name_____

Purpose: To learn to use MATLAB's **qr** function and understand the connections between the matrices it produces and the QR factorization described in the text.

Prerequisite: Section 6.4

MATLAB functions used: **qr**; and **gs** and **qrdat** from Laydata Toolbox

Background. In MATLAB, the command **[Q R] = qr(A)** works for any shape matrix A. It creates a square matrix Q whose columns are orthonormal (up to machine accuracy) and a matrix R which is the same shape as A and "upper triangular." Furthermore, $A = QR$ (up to machine accuracy), and this is called a *QR factorization of A*.

1. Verify the statements above for each matrix below. To get you started, type **qrdat** to get the matrices, then type **[Q R] = qr(A)** and record Q and R beside A below. Notice Q is square and R has the same shape as A and is "upper triangular."

Check that the columns of Q are orthonormal: type **format long, Q'*Q** and inspect to see that this product looks essentially like an identity matrix. Also compare QR and A, by typing **Q*R, A** and inspecting the matrices to see they are essentially identical.

Before proceeding, type **format short** to restore the usual display format for numbers. Then type **[Q R] = qr(B)**, record Q and R beside B, etc.

$$A = \begin{bmatrix} 3 & -5 & 1 \\ 1 & 1 & 1 \\ -1 & 5 & -2 \\ 3 & -7 & 8 \end{bmatrix} \qquad Q = \qquad R =$$

$$B = \begin{bmatrix} 3 & -5 \\ 1 & 1 \\ -1 & 5 \\ 3 & -7 \end{bmatrix} \qquad Q = \qquad R =$$

$$C = \begin{bmatrix} 1 & 2 & 5 \\ -1 & 1 & -4 \\ -1 & 4 & -3 \\ 1 & -4 & 7 \\ 1 & 2 & 1 \end{bmatrix} \qquad Q = \qquad R =$$

$$D = \begin{bmatrix} 1 & -1 & -1 & 1 & 1 \\ 2 & 1 & 4 & -4 & 2 \\ 5 & -4 & -3 & 7 & 1 \end{bmatrix} \qquad Q = \qquad R =$$

Background continued. The QR factorization developed in Section 6.4 of Lay's text is somewhat different from the factorization produced by MATLAB's **qr** function. Recall how Lay's algorithm works: A must have independent columns (so A must be square or "tall"); the Gram-Schmidt Process is applied to the columns of A; this yields a matrix Q_1 the same shape as A whose columns are orthonormal, and a square upper triangular matrix R_1 such that $A = Q_1R_1$ is true. So if A is not square, Q_1 and R_1 are clearly different from the Q and R you get from executing **[Q R] = qr(A)** -- they have different shapes.

However, there is a simple connection between the QR factorization algorithm in the text and the output of MATLAB's **qr** function. Exercise 21 in Section 6.4 shows that if you apply the algorithm in the text, getting Q_1 and R_1, then it is not hard to produce a square Q and "upper triangular" R which will agree essentially with what the **qr** function creates. The method is: extend the columns of Q_1 to an orthonormal basis for R^m, use Q_1 and the new vectors to create a square matrix $Q = [Q_1\ Q_2]$, and adjoin zero rows to R_1 to get an mxn matrix $R = \begin{bmatrix} R_1 \\ O \end{bmatrix}$. Then $A = QR$ will be true, and this Q and R will look like what **qr(A)** produces, except possibly for signs.

2. (hand) Using the matrix C above, investigate the connection between the matrices produced by the text's QR factorization algorithm and the matrices produced by the **qr** function. In more detail:

(a) Calculate a QR factorization of C using the algorithm in Section 6.4 (this is Exercise 15, Section 6.4). Record the answer, calling the matrices Q_1 and R_1:

Q_1 = R_1 =

(b) Compare this Q_1 and R_1 with the Q and R you got in question 1, when you executed **[Q R] = qr(C)**. Describe how the matrices are alike:

3. (MATLAB) The function **gs** does the Gram Schmidt Process as described in Theorem 11, Section 6.4. To verify this, type **G = gs(C)** and record G:

G =

(d) How does G compare to the matrix Q_1 that you calculated by hand in question 2(a)?

Background continued. Lay's QR algorithm is based on the Gram Schmidt process. However, the Gram Schmidt method does not do a satisfactory job of producing orthogonal vectors when it is done with computer arithmetic on a lot of columns – they get less and less orthogonal as the process proceeds.

The algorithm used for QR factorization in professional software such as MATLAB's **qr** function is quite different. It is much better numerically and it can be used on any matrix of any size, whether or not the columns are independent. Essentially, this algorithm multiplies A by suitably chosen orthogonal matrices $P_1,..,P_k$ until $P_k..P_1A$ equals a matrix R which has zeros below its "main diagonal," and then it outputs Q and R, where Q is the transpose of the product $P_k..P_1$. It is true that, in the special case that A is mxn and has independent columns and you calculate **[Q R] = qr(A)**, the first n columns of Q are theoretically the same as the columns you would get by applying the Gram-Schmidt Process to the columns of A, except possibly for sign.

The following question asks you to verify a few of the above assertions.

4. (hand) Let A be an mxn matrix, let $P_k,..,P_1$ be mxm orthogonal matrices, let $R = P_k..P_1A$ and let $Q = (P_k..P_1)^T$. Explain why Q must be an orthogonal matrix and why A must equal QR. You may cite theorems from the text. Attach an extra sheet.

MATLAB Project: QR Method for Calculating Eigenvalues Name_____

Purpose: To see how eigenvalues can be calculated by iterative methods that employ QR factorization, and get some understanding of why such methods work. This type of algorithm is used in all professional software for general eigenvalue calculations today, such as MATLAB's **eig** function.

Prerequisites: Sections 5.2 and 6.4

MATLAB functions used: **qr**, *, **eye**, :, **tic**, **toc**, **for**, **eig**; **qreigdat**, **qrbasic**, **qrshift** ; and **randomint** from Laydata Toolbox

Part I. Background.

It is not easy to calculate eigenvalues for most matrices. Characteristic polynomials are difficult to compute. Even if you know the characteristic polynomial, algorithms such as Newton's method for finding zeros cannot be depended upon to produce all the zeros with reasonable speed and accuracy.

Fortunately, numerical analysts have found an entirely different way to calculate eigenvalues of a matrix A, using the fact that any matrix similar to A has the same eigenvalues. The idea is to create a sequence of matrices similar to A which converges to an upper triangular matrix. If this can be done then the diagonal entries of the limit matrix are the eigenvalues of A. The remarkable discoveries are that the method can be done with great accuracy, and it will converge for almost all matrices. In practice the limit matrix is just block upper triangular, not truly triangular (because only real arithmetic is done), but it is still easy to obtain the eigenvalues. See Note 2 below.

The primary reason that modern implementations of this method are efficient and reliable is that a QR factorization can be used to create each new matrix in the sequence. Each QR factorization can be calculated quickly and accurately; it yields easily a new matrix orthogonally similar to the original matrix; and orthogonal similarities tend to minimize the effect of roundoff error on the eigenvalues.

The calculations in this project can be done without Laydata Toolbox; see Note 1 below. For more information on the theory, see Note 2 below.

1. (hand) Here you will verify some of the basic matrix properties that underlie this modern method. Suppose A is nxn. Let $A = Q_0R_0$ be a QR factorization of A and create $A_1 = R_0Q_0$. Let $A_1 = Q_1R_1$ be a QR factorization of A_1 and create $A_2 = R_1Q_1$. Explain why the following are true; use your paper and attach:

(a) $A = Q_0A_1Q_0^T$ (This is exercise 23, Sec. 5.2)

(b) $A = (Q_0Q_1)A_2(Q_0Q_1)^T$

(c) Q_0Q_1 is orthogonal (This is exercise 29, Sec. 6.2)

(d) A, A_1 and A_2 all have the same eigenvalues.

2. (MATLAB) Type **qreigdat** to get the following matrices. Then use MATLAB's **eig** function to calculate their eigenvalues, and record their eigenvalues below each matrix:

$$A_3 = \begin{bmatrix} 1 & -2 & 8 \\ 7 & -7 & 6 \\ 5 & 7 & -8 \end{bmatrix} \qquad A_4 = \begin{bmatrix} 4 & -2 & 3 & -7 \\ 1 & 2 & 6 & 8 \\ 8 & 5 & 1 & -5 \\ -5 & 8 & -5 & 3 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} 2 & 6 & -3 & 4 & -9 \\ -1 & 7 & -4 & -3 & -7 \\ -6 & -6 & -1 & 6 & 5 \\ 9 & 2 & 6 & 2 & -8 \\ -7 & -8 & 6 & -9 & -1 \end{bmatrix} \qquad A_6 = \begin{bmatrix} 0 & 0 & -1 & 4 & -1 & -6 \\ 0 & -2 & 2 & -5 & -2 & -5 \\ -1 & 2 & 8 & -4 & 3 & 2 \\ 4 & -5 & -4 & -6 & 1 & 0 \\ -1 & -2 & 3 & 1 & -2 & 7 \\ -6 & -5 & 2 & 0 & 7 & 10 \end{bmatrix}$$

Eigenvalues of each:

Part II. The basic QR algorithm.

Definition. The *basic* QR *algorithm for eigenvalues* is the iterative process begun in question 1, repeated many times: let $A = Q_0R_0$ be a QR factorization of A and create $A_1 = R_0Q_0$; let $A_1 = Q_1R_1$ be a QR factorization of A_1 and create $A_2 = R_1Q_1$; ...; having created A_m, let $A_m = Q_mR_m$ be a QR factorization of A_m and create $A_{m+1} = R_mQ_m$; etc. Continue until the entries below the diagonal of A_m are sufficiently small (or stop if no convergence is apparent).

3. (MATLAB) For each of the matrices shown above, use the function **qrbasic** to find how many steps of the basic QR algorithm are needed to make the absolute value of every entry below the diagonal smaller than 0.001, and how many seconds this takes. In the first column of the table on page 3, record the number of steps, the time, and the final matrix.

The function **qrbasic** simply does the commands **[Q R] = qr(A), A = R*Q** repeatedly. The program will stop when all entries below the diagonal are smaller than 0.001 (or after 200 steps if that test is never met), and it will report the last matrix and the total number of steps done. Typing **tic, some command, toc** will perform that command and also print the number of seconds required for executing it.

Specifically, to perform the calculations for A_3, type **tic, qrbasic(A3, 0.001), toc**

Record the results on page 3. Then repeat this calculation for A_4, A_5 and A_6.

Part III. Improving the basic QR algorithm by shifting and deflating

It is true that the basic algorithm can fail to converge for some matrices, and even when it does converge it can be extremely slow. There are simple modifications which greatly speed it up and can also make it converge for more matrices.

One of these modifications is *shifting*. The idea is: if A is the original matrix and A_m is the current matrix in the iteration, choose a scalar c, then get a QR factorication of the shifted matrix $A_m - cI$, and then undo the shift when you define A_{m+1}. If the scalars can be chosen so they get closer and closer to an eigenvalue of A, this will dramatically speed up convergence. The next algorithm shows one way to choose the scalars, and also introduces *deflating*.

Before trying out this new algorithm, answer question 4 where you will see the theoretical effect of shifting and why you can deflate after the last row looks like [0 .. 0 x].

4. (hand) Let A be nxn, let I denote the nxn identity matrix, and let c be a constant.

(a) Let λ be an eigenvalue of A. Explain why $A\mathbf{x} = \lambda\mathbf{x}$ is true if and only if $(A - cI)\mathbf{x} = (\lambda - c)\mathbf{x}$ is true. Use this to explain why the eigenvalues of $A - cI$ are the numbers obtained by subtracting c from each eigenvalue of A. (This is the reason that creating $A - cI$ is called "shifting.")

(b) Let $A - cI = QR$ be a QR factorization of $A - cI$ and define $A_1 = RQ + cI$. Show that $A = QA_1Q^T$.

(c) Suppose $A = \begin{bmatrix} B & C \\ O & D \end{bmatrix}$, where B and D are square and O is a zero matrix. Explain why the eigenvalues of A are the eigenvalues of B together with those of D. (This is Supplementary Exercise 12 in Chapter 5.)

Definition. The QR *algorithm for eigenvalues using diagonal shifts* is the following iterative process. Repeat the step described in question 4(b), each time choosing the value for the next scalar c to be the last diagonal entry of the previous matrix A_k. Stop when the last row of A_k looks like $[s_1\ s_2\ \dots\ s_{k-1}\ x]$ where each s_i is very small. Then that last entry x is approximately an eigenvalue of A. (By question 4(c), if each s_i were exactly zero, x would be a true eigenvalue of A.) Now *deflate* -- i.e., create a new smaller matrix by discarding the last row and column. Begin the process again on the new matrix. Continue repeating this process until all eigenvalues have been calculated, or until it appears the limit matrix cannot be improved.

5. (MATLAB) Use the function **qrshift** to apply the shift-deflate algorithm just described to the four matrices, A_3, A_4, A_5, A_6. Find how many seconds and how many steps are needed to make the absolute value of every entry below the diagonal less than 0.001, then for 0.0001. Record results in the table on page 3.

The function **qrshift** does shifting as described in question 4(b), and it deflates after the entries below the diagonal are less than the tolerance you specify. It will display each deflation, the final matrix and how many iterative steps were done. To perform the calculations for A_3, type

tic, qrshift(A3, 0.001), toc

Record the results on page 3. (Note: if you want to see the result of each deflation step, type **qrshift(A3, 0.001, 1)** instead – but this adds CPU time so it prevents you from seeing how much faster **qrshift** really is.)

Next type **tic, qrshift(A3, 0.0001), toc** , record results, and repeat these calculations for the other matrices.

6. (hand) Discuss: based on what you have seen, how does the basic QR method compare with the shift-deflate QR method?

Results from Question 3 **Results from Question 5**

Matrix A_3:

Basic QR, tol = 0.001	QR with diagonal shifts, tol = 0.001	QR with diagonal shifts, tol = 0.0001
# steps ___ time = ___	# steps ___ time ___	# steps ___ time ____

Matrix after you stop iterations:

****

Matrix A_4:

Basic QR, tol = 0.001	QR with diagonal shifts, tol = 0.001	QR with diagonal shifts, tol = 0.0001
# steps ___ time = ___	# steps ___ time ___	# steps ___ time ____

Matrix after you stop iterations:

****

Matrix A_5:

Basic QR, tol = 0.001	QR with diagonal shifts, tol = 0.001	QR with diagonal shifts, tol = 0.0001
# steps ___ time = ___	# steps ___ time ___	# steps ___ time ____

Matrix after you stop iterations:

****

Matrix A_6:

Basic QR, tol = 0.001	QR with diagonal shifts, tol = 0.001	QR with diagonal shifts, tol = 0.0001
# steps ___ time = ___	# steps ___ time ___	# steps ___ time ____

Matrix after you stop iterations:

Note1. You can do this project without Laydata Toolbox. First type in the matrices yourself. The following commands will accomplish the basic QR method used in question 3, for an nxn matrix A.

```
B = A; bound = 0.001; p = 0; num = 200;
while max(max(abs(tril(B,-1)))) > bound    % test size of entries in lower triangle
        [Q R] = qr(B); B = R*Q;
        p = p+1;
        if p > num, break, end  % break out of while loop
end  % while
p, B
```

The following lines will accomplish the shift-deflate algorithm used in question 5, for an nxn matrix A.

```
B = A; bound = 0.001; p = 0; num = 20; [m,n]=size(A);
for i = n:-1:2  % work from row n to row 2
B = B(1:i,1:i);  % deflate
        while max(max((abs((B(i,1:i-1))))) > bound  % test size of entries in row i, up to the diagonal
       [Q R] = qr(B-B(i,i)*eye(i)); B = R*Q + B(i,i)*eye(i) ;
  p = p+1;
  if p > num, break, end  % break out of while loop
        end  % while
A(1:i,1:i) = B;    % store B in upper left corner of A
end  % for
p, A
```

Note 2. Remarks about convergence. There is an excellent discussion of the theory of the QR method in *Understanding the QR Algorithm*, by D. Watkins, SIAM Review 24 (1982), pp. 427-440. This paper explains the geometric meaning of the algorithm and how it is an extension of the power method. (The power method is presented in Section 5.8 in Lay's text.) Briefly, the following things are true about the QR method, for an nxn real matrix A.

(a) If the eigenvalues of A all have different magnitudes, then the basic QR algorithm will converge to an upper triangular matrix. To see that the basic QR method can fail if two different eigenvalues have the same magnitude, try it on the following matrices: $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ and $\begin{bmatrix} 3 & -2 \\ 4 & -3 \end{bmatrix}$, each of which has eigenvalues ± 1.

(b) The matrices used above in this project were chosen so they have only real eigenvalues. However, a general real matrix can have nonreal eigenvalues. In this case, the algorithm described above, which uses only real QR factorizations, cannot possibly converge to an upper triangular matrix (why?). Nevertheless, it is true that a real shift can always be found so that the basic QR method applied to the new matrix will converge to a real block upper triangular matrix whose diagonal blocks are 1x1 or 2x2 matrices; then if you undo the shift, each 1x1 block is an eigenvalue and each 2x2 block easily yields a pair of complex conjugate eigenvalues, of the original matrix.

For example, the following matrices have some complex eigenvalues: $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$ and $\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$. Store the first matrix as A. Calculate **eig(A)** to see what its eigenvalues are. Type **[Q R] = qr(A), A = R*Q** and repeat this command several times. You will see cycling. Then apply the basic method to $A - I$. To do that, type **qrbasic(A + eye(3), .0001)**. Now you will see convergence to a block upper triangular matrix with 1x1 and 2x2 blocks as described above. Subtract I from this limit matrix. Solve the characteristic equation for the 2x2 block, and verify that this gives two complex numbers. These will be the nonreal eigenvalues of A, and the 1x1 block contains the third, real, eigenvalue of A.

(c) The method shown in (b) can be improved by first doing an orthogonal similarity to A to get a *Hessenberg* matrix -- one that has zeros below its first subdiagonal. The reason this is better is, if each entry on the first subdiagonal of a Hessenberg matrix is nonzero, then the basic QR algorithm is guaranteed to converge to a block upper triangular matrix. It is quite easy to do an orthogonal similarity to any A to get a Hessenberg matrix.[1] So all professional software to calculate eigenvalues begins by calculating a Hessenberg matrix which is orthogonally similar to the original A, and then applies the shift-deflate iterative process to this matrix. As soon as the matrix produced after some step has the form $\begin{bmatrix} B & C \\ O & D \end{bmatrix}$ where the entries of O are so small they can be treated as true zeros, then the iterative process is done separately on B and D. Notice this method is a natural for parallel processing.

[1] MATLAB can easily calculate a Hessenberg matrix similar to any A. Try this: **A = randomint(10), hess(A)**.

MATLAB Project: Least-Squares Solutions and Curve Fitting Name_____

Purpose: To practice using the theory of Least-Squares by calculating and plotting the Least-Squares line, quadratic curve, and cubic curve, and the error for each, for two sets of experimental data.

Prerequisite: Section 6.6

MATLAB functions used: **inv, ones, norm, size, plot, hold, print, polyval, polyfit, axis**; and **lsqdat** from Laydata Toolbox.

Background. Here is a summary of the method in Section 6.6 to find a Least-Squares polynomial $y = \beta_k x^k + \ldots + \beta_1 x + \beta_0$ to "fit" given data, $(x_1, y_1), \ldots, (x_n, y_n)$.

Define $X = \begin{bmatrix} x_1^k & \cdots & x_1 & 1 \\ x_2^k & \cdots & x_2 & 1 \\ \vdots & \cdots & \vdots & \vdots \\ x_n^k & \cdots & x_n & 1 \end{bmatrix}$, $\beta = \begin{bmatrix} \beta_k \\ \vdots \\ \beta_1 \\ \beta_0 \end{bmatrix}$ and $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$.

So X and **y** are known, and the entries of β are the unknowns. The system of equations $X\beta = \mathbf{y}$ is usually inconsistent, but one can find an approximate solution by the Least-Squares method; it will be a "best solution" in the sense that it produces a vector β for which the vector $\mathbf{y} - X\beta$ has smallest possible length. The length of $\mathbf{y} - X\beta$ is called the *Least-Squares error.*

In this project you will work with two different sets of experimental data. For each set of data, you will calculate the Least-Squares equations of degree 1, 2 and 3 and the error for each, and you will plot the data and these curves on a single graph. For convenience, round calculated numbers some. Record your calculated results in the appropriate tables, and attach your two graphs to this paper.

To begin, type **lsqdat** . You will get two 13x1 vectors **x0**, **y0** which will be used in question 1, and two 28x1 vectors **vel** and **drag** for question 2.

1. Type **[x0 y0]** to see the vectors side by side, and examine this to familiarize yourself with the data. Then type the following line to plot the 13 data points with * symbols, to establish scaling for the graph, and to hold this picture so future plots will appear on same axes:

plot(x0, y0, '*'), axis([-5 15 -100 700]), hold on

(a) Find the coefficients β_0 and β_1 for the Least-Squares line, $y = \beta_0 + \beta_1 x$. To do this, click on the Command screen and type the following lines. These commands use the formula in the text:

```
X1 = [ x0  ones(13,1)]
b1 = inv(X1' * X1) * (X1') * y0
```

You will get a column vector $\begin{bmatrix} 57.9396 \\ -102.6374 \end{bmatrix}$ and this is $\begin{bmatrix} \beta_1 \\ \beta_0 \end{bmatrix}$. Thus the equation of the Least-Squares line is $y = 57.9x - 102.6$ (after rounding).

Calculate the error for the Least-Squares line by typing:

err1 = norm(y0 – X1*b1)

Record the equation $y = 57.9x - 102.6$ and this error in the first table below. Finally, plot the Least-Squares line. Use semicolons as shown because there are many numbers in **x**, **X** and **y** but they are of interest only for plotting:

```
z = -5:0.1:15 ; z = z' ;                    (create a column vector z containing closely spaced numbers)
Z1 = [ z  ones( size(z)) ] ; y = Z1*b1 ;    (evaluate the linear equation at each value in z)
plot( z, y )                                (plot the line)
```

Remark: Of course you need only two points in order to plot a line, but this vector **z** will be useful in (b) and (c) where you do need a lot of closely spaced points in order to get smooth looking plots of curves.

(b) Find the coefficients β_0, β_1 and β_2 for the Least-Squares quadratic $y = \beta_2 x^2 + \beta_1 x + \beta_0$ for this data, and the associated Least-Squares error. Since **x0**, **y0** and **X1** are still in your workspace, you can do these things by typing the following commands:

```
X2 = [ x0.^2  X1 ]          (x0.^2 causes each entry of the vector x0 to be squared)
b2 = inv( X2' * X2 ) * X2' * y0          (calculate the coefficients for the quadratic)
err2 = norm( y0 – X2 * b2 )          (calculate the Least-Squares error for the quadratic)
```

Record the equation and the error in the first table below. Then plot the quadratic. Since **Z1** is still in your workspace, you can do this by typing the following lines:

```
Z2 = [ z.^2  Z1 ] ;  y = Z2*b2;          (evaluate the quadratic function at each value in z)
plot( z, y, 'r-.' )          (plot the quadratic as a red dash-dot curve)
```

(c) Find the coefficients for the Least-Squares cubic $y = \beta_3 x^3 + \beta_2 x^2 + \beta_1 x + \beta_0$ and the associated Least-Squares error, for the data **x0** and **y0**. Record the equation and error in the table below.

To begin, modify the commands in (b) and write your new commands in the space below. After you're sure the new commands are correct, execute them. Suggestion: display the cubic curve with a different color and symbol, say as a green dotted curve by typing **plot(z, y, 'g:')** .

Remark: Your new lines will be almost identical to those you used in (b). You can avoid some retyping by pressing the up arrow key until you find the command you want; modify it if necessary and then press [Enter] to execute it.

(d) Print your final graph. An easy way to do that is to click on the Figure screen and pull down its File menu to Print. Consult a lab assistant if necessary. Label the plot "Question 1" and attach it to this paper.

Least-Squares Polynomials for Question 1

	Equation	Least-Squares Error
Line		
Quadratic		
Cubic		

2. In this question, repeat the calculations of question 1, using the vectors **vel** and **drag**. This is data from a wind tunnel experiment on a model of an F-16 airplane climbing at a constant 5^o angle. The data was supplied by Aerolab, Laurel, MD.

Drag is the force opposing the forward motion of the plane. It depends on velocity and is parallel to the earth's surface. (In general drag also depends on a lift coefficient, which varies with the angle of attack, and on the area of the wing, but those are constant in this example.)

(a) To begin, type the following commands so you can inspect the data and plot the new data points:

hold off, clf (undo the **hold** command and clear the figure created above)
plot(vel, drag, '*'), axis([0 150 0 2]), hold on

Now repeat the calculations and plotting done in 1(a)-(c) above, for this new data. To simplify matters, start by typing

x0 = vel; y0 = drag;

to rename the vectors. Then you can use the up arrow key to find the lines you typed before and execute them again. Two lines will need to be different because **x0** has 28 entries now and they range from 10 to 145:

X1 = [ones(28,1) x0]

and

z = 5:3:150; z = z' ;

Record your results in the table below, print your graph, label the curves, and attach.

Least-Squares Curves for Question 2

	Equation	Least-Squares Error
Line		
Quadratic		
Cubic		

(b) It is true that one can use the laws of physics to derive a formula for drag. When the area of the wing and the lift coefficient are constant, this formula will be a polynomial

$$\text{drag} = c_m (\text{velocity})^m + .. + c_1 (\text{velocity}) + c_0$$

where each coefficient c_i is a constant and m is a positive integer. Using what you have seen for the LSQ polynomials of degree 1, 2 and 3, guess what the value of m is in that formula (m =1? 2 ? 3?). Explain why you guess that value.

3. There are special functions in MATLAB for calculating Least-Squares (LSQ) solutions, which are much easier to use than the calculations you did above, and they give more accurate results when you have large amounts of data. These functions are **polyfit** and **polyval** , and professionals use them so you should know about them.

Here is how they work. Suppose you have data in vectors **x0** and **y0** as above. To find the LSQ polynomial $y = \beta_k x^k + \ldots + \beta_1 x + \beta_0$ of degree k, you need only type:

b = polyfit(x0, y0, k) (then **b** will be the row vector $[\beta_k \ \ldots \ \beta_1 \ \beta_0]$)

To evaluate the polynomial at chosen values, type

z = (you choose appropriate closely spaced values)
y = polyval(b, z) (this evaluates the polynomial $\beta_k x^k + \ldots + \beta_1 x + \beta_0$ at each value in **z**)

Then you can type **plot(z,y)** to see the graph of the polynomial. For example, try the following commands for yourself, and observe they give exactly the same results as you got in question 2(c):

```
clf, hold off, plot(vel, drag, '*'), hold on
b3 = polyfit(vel, drag, 3)
z = 5:3:150;
y = polyval(b3, z);
plot(z, y, 'g:')
```