MATLAB Exercises for Linear Algebra - M349

Peter Monk
Department of Mathematical Sciences
University of Delaware
Newark, DE 19716

Version of Fall 2003

\footnote{Copyright ©1994 by Peter Monk. This work may be distributed and used according to the GNU General Public License.}
Copyright Notice

MATLAB Exercises for Linear Algebra - M349 by Peter Monk
Copyright ©2003 Peter Monk

This work is free text; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This text is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

To obtain a complete copy of the GNU General Public License write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
# Contents

1 Two Linear Equations in Two Unknowns ................................................. 7
   1.1 Introduction ....................................................................................... 7
   1.2 Getting started in the math lab ....................................................... 7
   1.3 Systems of equations with two unknowns ......................................... 7
   1.4 University Unix Account Use .......................................................... 10
      1.4.1 Activating your computer account ........................................... 10
      1.4.2 Working on MATH349 class assignments ................................. 10

2 Matrix Calculus ....................................................................................... 11
   2.1 About computer assignments ........................................................... 11
   2.2 Some Information About MATLAB .................................................. 11
      2.2.1 Creating and Managing Diary Files ........................................ 11
      2.2.2 Correcting Typos in MATLAB ................................................. 12
   2.3 Entering Matrices into MATLAB ...................................................... 12
   2.4 Addition and subtraction ................................................................. 12
      2.4.1 Addition ..................................................................................... 12
      2.4.2 Subtraction .............................................................................. 13
   2.5 Multiplication, transpose and powers ............................................... 13
      2.5.1 Multiplication ........................................................................... 14
      2.5.2 Transpose ................................................................................ 14
      2.5.3 Matrix powers .......................................................................... 15
   2.6 Special Matricesa ............................................................................... 16
      2.6.1 Some MATLAB details .............................................................. 16
      2.6.2 Special matrices in MATLAB .................................................... 17
      2.6.3 Referencing matrix elements, columns and rows ....................... 18
      2.6.4 What have I created? ................................................................. 18
   2.7 Elementary Matrices in MATLAB ..................................................... 19
      2.7.1 The elementary matrix for interchanging rows .......................... 19
      2.7.2 The elementary matrix for scaling a row ................................... 19
      2.7.3 The elementary matrix for adding a multiple of one row to another 19
   2.8 Exercises on Matrix Calculus and Row Reduction ............................ 21
      2.8.1 Row Echlon Form .................................................................... 21
      2.8.2 Exercise (for credit) ................................................................. 21
      2.8.3 Testing Matrix Algebra ............................................................ 22
      2.8.4 Investigating the transpose ....................................................... 22
      2.8.5 Hand In Check List ................................................................. 22
   2.9 Matrix Inverses and Partitioned Matrices ........................................... 23
      2.9.1 Building matrices using partitioning ....................................... 23
      2.9.2 Extracting parts of matrices .................................................... 23
      2.9.3 The matrix inverse in MATLAB ............................................. 24
### 3 Vector Spaces

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Spanning Sets</td>
<td>27</td>
</tr>
<tr>
<td>3.2 The Null Space</td>
<td>27</td>
</tr>
<tr>
<td>3.2.1 Exercise on finding nullspaces using rref (not for credit)</td>
<td>28</td>
</tr>
<tr>
<td>3.2.2 Exercise on null-spaces using Null (not for credit)</td>
<td>29</td>
</tr>
</tbody>
</table>

### 4 An Exercise on Partitioned Matrices

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0.3 Exercise</td>
<td>31</td>
</tr>
<tr>
<td>4.1 Hand in check list</td>
<td>32</td>
</tr>
</tbody>
</table>

### 5 Exercise on Magic Matrices

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Introduction</td>
<td>33</td>
</tr>
<tr>
<td>5.2 Magic Matrices</td>
<td>33</td>
</tr>
<tr>
<td>5.2.1 Exercise</td>
<td>34</td>
</tr>
<tr>
<td>5.3 Hand in check list</td>
<td>34</td>
</tr>
</tbody>
</table>

### 6 Linear Transformations

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Linear Maps from the plane to the plane</td>
<td>35</td>
</tr>
<tr>
<td>6.1.1 Elementary Matrices</td>
<td>36</td>
</tr>
<tr>
<td>6.2 Composition of Matrices</td>
<td>36</td>
</tr>
<tr>
<td>6.2.1 Does Matrix Multiplication Commute</td>
<td>37</td>
</tr>
<tr>
<td>6.2.2 Null spaces</td>
<td>37</td>
</tr>
<tr>
<td>6.3 Invariant vectors</td>
<td>37</td>
</tr>
<tr>
<td>6.4 Application of Linear Algebra to Electricity</td>
<td>38</td>
</tr>
<tr>
<td>6.4.1 MATLAB</td>
<td>38</td>
</tr>
<tr>
<td>6.4.2 Circuit Analysis 1</td>
<td>38</td>
</tr>
<tr>
<td>6.4.3 More on circuits</td>
<td>41</td>
</tr>
<tr>
<td>6.4.4 Hand in check list</td>
<td>44</td>
</tr>
<tr>
<td>6.5 Acknowledgement</td>
<td>44</td>
</tr>
</tbody>
</table>

### 7 Orthogonality

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 Orthogonal Subspaces</td>
<td>45</td>
</tr>
<tr>
<td>7.2 The Gram Schmidt Process</td>
<td>46</td>
</tr>
</tbody>
</table>

### 8 Programming MATLAB.

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1 MATLAB functions</td>
<td>49</td>
</tr>
<tr>
<td>8.2 Looping</td>
<td>50</td>
</tr>
<tr>
<td>8.3 Logic</td>
<td>51</td>
</tr>
<tr>
<td>8.3.1 Exercise on looping</td>
<td>53</td>
</tr>
<tr>
<td>8.3.2 Exercise on functions</td>
<td>53</td>
</tr>
<tr>
<td>8.3.3 Exercise on MATLAB logic</td>
<td>54</td>
</tr>
<tr>
<td>8.4 Hand-In Check list</td>
<td>54</td>
</tr>
</tbody>
</table>

### 9 The matrix inverse (explicitly) and the determinant

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1 Computing the Inverse of a Matrix</td>
<td>55</td>
</tr>
<tr>
<td>9.1.1 Exercise (not for credit)</td>
<td>55</td>
</tr>
<tr>
<td>9.2 Determinant of a matrix</td>
<td>55</td>
</tr>
<tr>
<td>9.2.1 Exercise (not for credit)</td>
<td>55</td>
</tr>
<tr>
<td>9.3 MATLAB commands (continued)</td>
<td>56</td>
</tr>
<tr>
<td>9.3.1 Continuation</td>
<td>56</td>
</tr>
<tr>
<td>9.4 Toeplitz matrices</td>
<td>57</td>
</tr>
<tr>
<td>9.4.1 Exercise (not for credit)</td>
<td>57</td>
</tr>
</tbody>
</table>
Chapter 1

Two Linear Equations in Two Unknowns

1.1 Introduction

The purpose of this programming assignment is to make sure MATLAB works for you, and demonstrate some properties of solutions of linear systems.

1.2 Getting started in the math lab

If the screen on the workstation is blank, move the mouse. The screen should then display the standard Redhat LINUX login dialogue box. If you don’t see the login prompt, ask the instructor.

Log on Login using your username and password from the central UNIX system (i.e. the username and password you use for e-mail on copland). Then wait. The window system will start up and you should see some icons etc (this takes a while).

Run MATLAB Now you are ready to use MATLAB. Just move the mouse until the pointer over the red hat in the lower left hand corner and click the left mouse button. Choose Run Command from the large list of choices (it’s towards the bottom). Type matlab into the dialog box and click on Run to run it. The MATLAB window should start up. Commands for MATLAB are typed directly into the “Command Window” pane. Note the strip along the top of this pane must be blue indicating that input will go to the command window (left click on the Command Window to focus input there if necessary).

Log-out from the machine You must do this after you have finished the assignment otherwise strangers may be able to access your account and ruin your work. But don’t logout until you have finished the assignment! To logout, simply move the pointer to the background (using the mouse). Then hold down the right mouse button, move the pointer to the entry Logout ... and release the mouse button. Then confirm your intention to quit the window system in the “End Session” dialogue box.

1.3 Systems of equations with two unknowns

Systems of equations involving only two unknowns can be analyzed graphically. For example, consider the system

\[
\begin{align*}
2x + 3y &= 1 \\
x - y &= 2
\end{align*}
\]
(since we only have two unknowns I have switched from using the unknowns $x_1$ and $x_2$ to $x$ and $y$). The first equation, $2x + 3y = 1$ is the equation of a straight line. To see this, note that we can solve for $y$ to get

$$y = \frac{1}{3} - \frac{2}{3}x$$

In the same way $x - y = 2$ is also the equation of a straight line. We shall use MATLAB to draw the graphs.

If you have not started MATLAB, do so now. At the MATLAB >> prompt type (carefully!)

```matlab
ezplot('1/3-2*x/3',[-10,10])
grid
```

The first line uses a simple plotting tool in MATLAB to plot the equation in quotes. You should be able to see how the equation $y = 1/3 - 2x/3$ was translated into MATLAB (* is the MATLAB multiplication operator and we only need enter the right hand side). The entry [-10,10] tells MATLAB to have the x-axis run from -10 to 10. Type help ezplot for more details and to see the MATLAB help system. The command grid puts up some grid lines on the graph.

Now we shall plot the graphs of the first and second equations together and add a title by typing (replace My name by your own name):

```matlab
ezplot('1/3-2*x/3',[-10,10])
hold on
ezplot('x-2',[-10,10])
title('My name')
grid
hold off
```

The command hold on tells MATLAB to hold the graph and not erase it when we ez-plot a second time (otherwise the first graph would be lost). The hold off tells MATLAB to turn off the hold, so that any new graph will overwrite the old graphs.

You can see from the screen that this pair of equations is consistent and has a unique solution.

**Editing tip:** To help you in entering these lines, you can edit the line you are typing using the small “Del” key close to the “Enter” key to delete what you have typed. The arrow keys (left and right) move the cursor along the line you are typing and you can insert or delete text anywhere in the line. Finally, the up-arrow key can recall previously typed lines.

Now you are to draw three graphs using MATLAB commands like the ones you just used. In each case you should solve each equation for $y$ in terms of $x$ and plot all the graphs on one axis. For each linear system

- Is there a solution? Many solutions? No solution?
- If there is a unique solution, you can give an estimate of it’s value by reading off the coordinates of the point where the lines cross.

This assignment is not for credit, but I urge you to do it anyway.

1. Analyze the following system of two equations in two unknowns (2 × 2 system)

   $$
   -x + 2y = 3 \\
   2x - 4y = -6
   $$

2. Analyze the following system of two equations in two unknowns (2 × 2 system)

   $$
   x + y = 1 \\
   2x + 2y = 3
   $$
3. Analyze the following system of three equations in two unknowns (2 × 2 system)

\[
\begin{align*}
    x + y &= 1 \\
    2x - y &= 0 \\
    -x - y &= -1
\end{align*}
\]

4. Analyze the following system of three equations in two unknowns (3 × 2 system)

\[
\begin{align*}
    x + y &= 1 \\
    -x - y &= -1 \\
    2x + 2y &= 2
\end{align*}
\]

Congratulations, you have just completed your first exercise! Don’t forget to logout!
1.4 University Unix Account Use

To complete this assignment and for later work you will need to use MATLAB outside class. It is available on the university UNIX system.

1.4.1 Activating your computer account

During this semester, you can use the central UNIX systems to complete assignments outside class time. If you have used one of the computers named Copland or Strauss in the past, then you already have a username and password. If you have never used a central UNIX machine before, you will need to get a username and password. See the web page http://www.udel.edu/help/.

1.4.2 Working on MATH349 class assignments

You must be on strauss to run MATLAB. MATLAB will not work on copland. If you are not on strauss, login there now. Next you need to make sure that you are charging your computer use time to the class (if you don’t do this you will run down your e-mail account and not be able log in!).

1. **To begin**: type `newgrp 2024`. The group/project number for your project is **2024** (if you forget this number, you can see all your projects by typing `chdgrp`).

2. **Do your work**. For example type `matlab` to start MATLAB.

3. **End your work** for this class: type `exit`. Then, you can either log out or work on something else.

If at any time you need to make sure that 2024 is the current group, type `id` and the number which will appear in parenthesis following `gid` will be the current group. If you get any messages asking for a password or other messages, you are not in the project (maybe you just added the class?). Wait another day.
Chapter 2

Matrix Calculus

2.1 About computer assignments

This assignment is to be handed in one week from today (next thursday). To get full credit for this and future assignments you must:

1. Don’t panic. If you have trouble understanding the assignment or getting on the computer SEE ME. I am very willing to help!

2. Hand in all output requested (for example program listings, graphs, and results of running programs).

3. Answer all questions (correctly of course).

2.2 Some Information About MATLAB

This section is for information only. You do not have to type any of the commands until the next section, but you may find it helpful to try some. You do not have to hand in anything related to this section.

2.2.1 Creating and Managing Diary Files

You will often be asked to hand in a record of all your work in MATLAB. Rather than copy from the screen, you can create a “diary” of your efforts. This is an ordinary text file stored in your directory. When you are in MATLAB you can type

```
diary junk.txt
```

which will start a file call junk.txt containing this and future MATLAB commands and results. Of course you can replace “junk.txt” with any valid UNIX file name of your choice. To temporarily suspend diary output (for example, if you want to experiment with something you don’t want me see), you can type

```
diary off
```

To resume recording type

```
diary on
```

Diary files can be manipulated from within MATLAB (to some extent). The MATLAB commands `dir` or `ls` will give a list of all ordinary files in your current directory. The command `type <filename>` where you replace a `<filename>` by the name of an appropriate file will type out the contents of the file on your screen, and `delete <filename>` will delete a file from your disk.

If you wish, start MATLAB and try typing the following sequence of commands
dir
diary example.list
A=[1,2,3]
B=A-1
diary off
dir
type example.list
delete example.list

Notice that the results of your calculations $A=[1,2,3]$ and $B=A-1$ are written into the diary file.
You can print a diary file while on a Math Workstation Lab machine by typing (in MATLAB)

!lpr <filename>

(where <filename> is replaced by the name of the file you want to print). **Note:** This does not work on strauss. On that machine you would type (again in Matlab)

!qpr -q smips <filename>

### 2.2.2 Correcting Typos in MATLAB

You will almost certainly make some typos during your work. You can edit the command line you are typing by using the arrow keys → and ← to move back and forth, and the “Del” key to delete (this is true on Math Lab Workstation but may differ when you use outside terminals). Previous command lines can be accessed using the uparrow key so that if you make a mistake you don’t need to type the line over.

### 2.3 Entering Matrices into MATLAB

Next we need to know how to enter matrices into MATLAB. For example to enter the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

you type

$A=[1,2,3,4;5,6,7,8]$

so matrices are entered row by row between [ and ]. Individual entries are separated by “,” or a space, and a new row is signalled by a “;”.

### 2.4 Addition and subtraction

The big advantage of MATLAB from our point of view is that it understands matrices and vectors. In particular it understands matrix calculus (a calculus is a method of calculating, THE CALCULUS just happens to be the first and most important one).

#### 2.4.1 Addition

If $A$ and $B$ are either scalars, vectors or matrices that can be added and if you type $A+B$ in MATLAB you will get a correctly added sum. There is one way that MATLAB differs from standard mathematical usage of +. If $A$ is any matrix, vector or scalar and $b$ is any scalar then $A+b$ is a matrix of the same shape as $A$ calculated by adding $b$ to every entry in $A$. 

12
Exercise (Not for credit)

Start MATLAB. Either type the following matrices into MATLAB or type the command `matcalc` which will enter the matrices into MATLAB for you.

\[
A = \begin{pmatrix}
1 & -1 & 2 \\
-3 & 1 & 1 \\
1 & 4 & -6
\end{pmatrix}, \quad
B = \begin{pmatrix}
0.5 & 0.35 & 0.15 \\
0.35 & 0.6 & 0.05 \\
0.15 & 0.05 & 0.8
\end{pmatrix}, \quad
C = \begin{pmatrix}
1 & -1 \\
1 & 2 \\
-3 & 2
\end{pmatrix}.
\]

In MATLAB, type

1+2
A+A
A+B
B+A
A+C
1+C

- Did MATLAB refuse to do any of the requested calculations? Why?
- Does \(A + B = B + A\)?
- What did \(1 + C\) do? Is that what you expect?

### 2.4.2 Subtraction

The `-` operator in MATLAB works just like `+` (except it subtracts instead of adds).

Exercise (Not for credit)

Using the matrices \(A\), \(B\) and \(C\) from the previous example, type the following:

2-1
A-A
A-B
B-A
-(B-A)
A-C
1-C

- Did MATLAB refuse to do any of the requested calculations? Why?
- Does \(A - B = B - A\)? What is the relationship between \(A - B\) and \(B - A\)?
- What did \(1 - C\) do? Is that what you expect?

### 2.5 Multiplication, transpose and powers

Start MATLAB. Enter the following matrices into MATLAB.

\[
A = \begin{pmatrix}
1 & -1 & 2 \\
-3 & 1 & 1 \\
1 & 4 & -6
\end{pmatrix}, \quad
B = \begin{pmatrix}
0.5 & 0.35 & 0.15 \\
0.35 & 0.6 & 0.05 \\
0.15 & 0.05 & 0.8
\end{pmatrix}, \quad
C = \begin{pmatrix}
1 & -1 \\
1 & 2 \\
-3 & 2
\end{pmatrix}.
\]
2.5.1 Multiplication

The multiplication operator in MATLAB is ∗. If A and B are two matrices that have the right number of rows and columns to be multiplied then $A \times B$ is the matrix product. Note that if b is any scalar then $b \times A$ is a matrix of the same shape as A with entries that are just b times the corresponding entry in A (as we would expect from the definition of scalar multiplication given in class).

**Exercise (not for credit)**

Using the matrices A, B and C above type the following:

- $2 \times 3$
- $A \times A$
- $A \times B$
- $B \times A$
- $A \times C$
- $C \times A$
- $2 \times B$

- Did MATLAB refuse to do any of the requested calculations? Why?
- Does $A \times B = B \times A$ in general?
- What did $2 \times B$ do? Is that what you expect?

2.5.2 Transpose

MATLAB also knows how to take the transpose of a matrix. If A is any matrix with real number entries, then $A^{\prime}$ is its transpose.

**Exercise (for credit)**

Using the matrices A, B and C from above type the following:

- $2^{\prime}$
- $A^{\prime}$
- $B^{\prime}$
- $C^{\prime}$
- $C^{\prime} \times A$
- $A \times C^{\prime}$
- $(A^{\prime})^{\prime}$
- $(A^{\prime} + A)/2$

- Did MATLAB refuse to do any of the requested calculations? Why?
- Does $B = B^{\prime}$? Is $B$ symmetric?
- What is the relationship between $(A^{\prime})^{\prime}$ and $A$?
- Is $(A^{\prime} + A)/2$ symmetric? Is this quantity symmetric for any square matrix $A$ (prove or disprove your statement)?
2.5.3 Matrix powers

MATLAB has all sort of useful operations that help with matrix calculations. One example is the power operator. If $A$ is a square matrix (same number of rows and columns) and $n$ is a positive integer then

$$A^n$$

is just $A^n$ as you would expect.

Exercise (for credit)

Using the matrices $A$, $B$ and $C$ from above, and the vector

$$x = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

and type the following MATLAB commands:

- $2^3$
- $A^3$
- $B^{10} \ast x$
- $B^{20} \ast x$
- $B^{30} \ast x$
- $C^2$

- Did MATLAB refuse to do any of the requested calculations? Why?
- On the basis of these numerical results, what do you think is the value of

$$\lim_{n \to \infty} B^n x?$$

It turns out that this limiting vector is called an eigenvector, and we shall study it in some detail towards the end of the course. Let $y$ be your guess of $\lim_{n \to \infty} B^n x$. Evaluate $B \ast y$. Is it true that $B \ast y = y$?
2.6 Special Matrices

2.6.1 Some MATLAB details

The : in MATLAB

The : is one of the most important commands in MATLAB and has many different forms. Try typing

```
1:10
```

You should see the integers 1 to 10 saved in the variable `ans`. More generally

```
y=2:7
```

creates a vector `y` of integers starting at 2 and ending at 7. At this point the most general form of the command is

```
<variable>=<start>:<step>:<end>
```

where `<start>` is the first entry to be put in the vector, the next entry will be `<start>+<step>`, the next `<start>+2<step>` and so on. The last entry in the vector is the largest value of `<start>+P<step>` less than `<end>` for some integer `P`. Generally we use only integer values for `<start>`, `<step>` and `<end>`, but you are allowed to use non-integer values. Note that vectors created this way are row vectors.

Exercise (not for credit)

Use the : operator to create the following vectors

1. `y = (5, 6, 7, 8)`
2. `z = (8, 7, 6, 5)`
3. `t = (2.1, 2.2, 2.3, 2.4, 2.5, 2.6)`
4. What will happen if you type `pi:0.1:4` in MATLAB? Try the command and see if you were correct.

The ; in MATLAB

The function of the semi-colon is to suppress screen output from MATLAB. Type the following lines in MATLAB and observe what happens:

```
x=-10:10;
x=-10:10
```

If you want to see the results of a MATLAB command, leave off the semi-colon. If you would prefer not to see the results (if they are boring or too long to fit on the screen) use a semi-colon.

Getting help in MATLAB

As you progress in learning MATLAB, you will begin to know hundreds of different functions. Unfortunately, you will forget exactly how to use most of these functions. It’s useful to be able to look up commands in the manual to remind yourself of the details of usage. This is where `help` comes in. Enter MATLAB (if you are not already in MATLAB) and type

```
help
```

This will give you a list of directories that contain MATLAB commands accessible to MATLAB. Some directories contain commands that implement things like signal processing, optimization, data fitting (people have put together many libraries of MATLAB functions devoted to areas of engineering and mathematics that use linear algebra). Others like `matlab/general` contain more general commands like `dir`. To get help on a particular function, for example `dir`, type
help dir

What do you suppose the command

help *

will do? Type it and see. Recall that * was used in the first programming assignment. We shall learn all
about it (and + and −) next.

Controlling Scrolling

When you typed help you may have noticed that a lot of lines scrolled off the screen. You can tell MATLAB
to show a page at a time by using the command more. Precisely, if you type:

more on

this will turn on paging (like more in UNIX). You advance a page at a time by hitting the space bar. To
turn more off (this is usually how MATLAB is used), you type:

more off

2.6.2 Special matrices in MATLAB

MATLAB has commands to build two special matrices that we have discussed in class. These are the identity
and zero matrix.

The zero matrix

The zero matrix is made using the command zeros which has the three forms:

| zeros(n) | Here n is an integer. This produces an n × n matrix of zeros. |
| zeros(m,n) | This produces an m × n matrix of zeros. |
| zeros(size(A)) | Here A is a matrix. This gives a matrix of zeros of the same shape as A. |

Try typing in the following commands to MATLAB:

zeros(5)
zeros(3,2)
A=[1,2;3,4]
zeros(size(A))

The identity matrix

The identity matrix can also be created easily using a command that has similar syntax to zeros. The
command is eye. It has the following three forms:

| eye(n) | Here n is an integer. This produces the n × n identity matrix. |
| eye(m,n) | This produces an m × n matrix with ones down the main diagonal and remaining entries zero. If m ≠ n this is a generalization of the identity matrix introduced in class. |
| eye(size(A)) | Here A is a matrix. This gives a matrix of the same shape as A with ones down the main diagonal and remaining entries zero. |

Try typing in the following commands to MATLAB:

eye(5)
eye(3,2)
A=[1,2;3,4]
eye(size(A))
2.6.3 Referencing matrix elements, columns and rows

It is sometimes necessary to modify individual entries in a matrix. You can access these entries in the usual way using row and column indices. Thus $A(i,j)$ is the $i,j$ entry of $A$. Try typing the following commands into MATLAB:

\[
A(1,2)=A(1,2)+3 \\
i=3; j=4 \\
A(i,j)=0 \\
A(1,2)=0
\]

Note that the command $A(i,j)=0$ worked even though $A$ was a $2 \times 3$ matrix! MATLAB simply enlarged the matrix to make the command work!! This is something you have to be careful with.

MATLAB also allows you to access whole columns or rows of a matrix easily. In MATLAB a “:” means all allowable values of an index. Thus the MATLAB command $A(1,:)$ means the first row of $A$ and $A(:,3)$ represents the third column of $A$. Try typing the following in MATLAB:

\[
A=[1,2,3; 3,2,1] \\
A(1,:) \\
A(:,3) \\
A(:,3)=A(:,3)+1
\]

The last line adds 1 to the third column of $A$. The “:” operator is very useful and important to MATLAB. In fact it is so useful that some scientific programming languages have adopted it (FORTRAN-90).

Exercise (not for credit)

Use the MATLAB commands discussed in this hand out to enter the following matrices into MATLAB (do not type them in entry by entry):

\[
A = \begin{pmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{pmatrix}, \quad B = \begin{pmatrix}
0 & 1 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{pmatrix}, \quad C = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{pmatrix}
\]

\[
D = \begin{pmatrix}
2 & 0 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 0 \\
0 & 0 & 0 & 2 & 0 \\
0 & 0 & 0 & 0 & 2 \\
\end{pmatrix}, \quad E = \begin{pmatrix}
1 & 2 & 0 & 0 & 0 \\
3 & 7 & 0 & 0 & 0 \\
0 & 0 & 3 & 0 & 0 \\
0 & 0 & 0 & 3 & 0 \\
0 & 0 & 0 & 0 & 3 \\
\end{pmatrix}
\]

\[
F = \begin{pmatrix}
2 & 0 & 0 & 0 & 1 \\
0 & 2 & 0 & 0 & 1 \\
0 & 0 & 2 & 0 & 1 \\
0 & 0 & 0 & 2 & 1 \\
0 & 0 & 0 & 0 & 3 \\
\end{pmatrix}
\]

2.6.4 What have I created?

You have now entered a lot of matrices into MATLAB and so have created a lot of variables. Two commands allow you to see the names of the variables you have created:

\texttt{who} This list all variables

\texttt{whos} This lists all variables with their attributes.

Try typing \texttt{who} and \texttt{whos} in MATLAB.
2.7 Elementary Matrices in MATLAB

In this exercise we shall see an example of each type of elementary matrix and its inverse.

2.7.1 The elementary matrix for interchanging rows

Here we will demonstrate the use of elementary matrices to interchange rows. In the following example, we shall interchange rows 1 and 4 of a matrix. First we create a 4 \times 4 matrix of integers between -10 and 10 to act as a test matrix:

\[ A = \text{round}(20 \times (\text{rand}(4,4) - 0.5)) \]

Next we construct the elementary matrix for interchanging rows 1 and 4 using the following compound command. Notice you can have more than one MATLAB command on a line:

\[ E = \text{eye}(|A|); E(1,1) = 0; E(1,4) = 1; E(4,4) = 0; E(4,1) = 1 \]

Finally we multiply by \( E \times A \) to interchange the rows of \( A \):

\[ E \times A \]

Check that left multiplication by \( E \) had the desired effect.

Exercise (not for credit):

What elementary matrix \( E \) would you use to interchange rows 2 and 3 of \( A \). Enter it into MATLAB and check that it works.

What is \( E^{-1} \)? Type the following in MATLAB and explain the result:

\[ E \times E \]

2.7.2 The elementary matrix for scaling a row

Here we will demonstrate the use of elementary matrices to scale a row by a constant multiple. In this example we shall multiply row 3 by the scale factor 7. Type the following commands

\[ A = \text{round}(20 \times (\text{rand}(4) - 0.5)) \]
\[ E = \text{eye}(|A|); E(3,3) = 7 \]
\[ E \times A \]

Exercise (not for credit):

What elementary matrix \( E \) would you use to multiply row 4 of \( A \) by \( 1/4 \). Enter it into MATLAB and check that it works.

Enter the inverse of \( E \) into MATLAB (call it \( F \)) and check that you got the correct matrix by computing the product \( F \times E \).

2.7.3 The elementary matrix for adding a multiple of one row to another

Here we will demonstrate the use of elementary matrices to add a multiple of one row to another row. In particular we shall show how to use matrix multiplication to add 1/3 of row 2 to row 3. Type the following commands

\[ A = \text{round}(20 \times (\text{rand}(4) - 0.5)) \]
\[ E = \text{eye}(|A|); E(3,2) = 1/3 \]
\[ E \times A \]
Exercise (not for credit):

What elementary matrix $E$ would you use to add 3 times row 4 to row 1? Enter it into MATLAB and check that it works.

Enter the inverse of $E$ into MATLAB (call it $F$) and check that you got the correct matrix by computing the product $F\cdot E$. 
2.8 Exercises on Matrix Calculus and Row Reduction

2.8.1 Row Echelon Form

Not surprisingly MATLAB knows about reduced row echelon form and has a function that can reduce a matrix in one single call. The function is \texttt{rref}, and if \texttt{B} is a matrix, you reduce it by simply typing \texttt{rref(B)}.

For example, the system
\[
\begin{align*}
    x_1 + 3x_2 &= 3 \\
    2x_1 - x_2 &= 8
\end{align*}
\]
has the augmented matrix
\[
\begin{pmatrix}
    1 & 3 & 3 \\
    2 & -1 & 8
\end{pmatrix}.
\]

This matrix is entered into MATLAB by typing
\[
A=[1,3,3;2,-1,8]
\]

You can then type
\[
\texttt{rref(A)}
\]
to obtain the following reduced row echelon form corresponding to \( A \)
\[
\text{ans =}
\begin{pmatrix}
    1 & 0 & 3.8571 \\
    0 & 1.0000 & -0.2857
\end{pmatrix}
\]

From this, you can read off the solution \( x_1 \approx 3.8571 \) and \( x_2 \approx -0.2857 \).

It’s even more fun(!) to watch a movie of row reduction. Type in the matrix \( A \) defined by
\[
A=[1,3,4,1,3;1,3,8,2,5;2,6,-4,3,0;3,9,0,4,3]
\]

Next have MATLAB show you a movie of row reduction for this matrix by typing
\[
\texttt{rrefmovie(A)}
\]

2.8.2 Exercise (for credit)

Write down the augmented matrix for the following system:
\[
\begin{align*}
    x_1 + 3x_2 + x_3 + x_4 &= 3 \\
    2x_1 - 2x_2 + x_3 + 2x_4 &= 8 \\
    3x_1 + x_2 + 2x_3 - x_4 &= -1
\end{align*}
\]

Now start MATLAB and start a diary file. Enter the augmented matrix into MATLAB in the usual way, and then use \texttt{rref} to reduce it to reduced row echelon form. Stop the diary and print the diary file. Using the reduced matrix, write down the general solution of this system. Hand in both the diary file and your solution as part of this assignment.
2.8.3 Testing Matrix Algebra

Suppose we want to use MATLAB to test computationally how to expand \((2A + 3B)^2\) where \(A\) and \(B\) are \(n \times n\) matrices. More precisely, suppose we want to know which of the following formulae are equal to \((2A + 3B)^2\):

1. \((3B + 2A)^2\)
2. \(4A^2 + 12AB + 9B^2\)
3. \(4A^2 + 6AB + 6BA + 9B^2\)
4. \( (2A + 3B)(3B + 2A)\)
5. \(2A(2A + 3B) + 3B(2A + 3B)\)
6. \(2A(2A + 3B) + 3(2A + 3B)B\)

To do this, generate two \(4 \times 4\) matrices with random integer entries by typing the following commands in MATLAB (of course start a diary file first!!):

\[
A = \text{round}(10 \times \text{rand}(4))
\]
\[
B = \text{round}(10 \times \text{rand}(4))
\]

For your information the command \text{rand}(N) generates an \(N \times N\) matrix with random entries uniformly distributed in \([0, 1]\). Then \text{round} rounds the numbers to the nearest integer.

Now use MATLAB to compute \(C = (2A + 3B)^2\), then compute \(C1 = (3B + 2A)^2\). Compare \(C\) and \(C1\) by subtracting one from the other. Are \(C\) and \(C1\) equal? Now repeat this using the other expressions above. Record which expressions are equal to \((2A + 3B)^2\). Once you have decided which of the above expressions are likely to be equal to \((2A + 3B)^2\), use the rules of matrix algebra to PROVE mathematical equality for all matrices regardless of size.

2.8.4 Investigating the transpose

Now we will use a method like the previous one to investigate some relationships between the product of the transpose of two matrices. Generate two random matrices as above by typing

\[
A = \text{round}(10 \times \text{rand}(4))
\]
\[
B = \text{round}(10 \times \text{rand}(4))
\]

Then compute the following matrices

- \(C = (AB)^T\)
- \(D = A^T B^T\)
- \(E = B^T A^T\)
- \(F = (BA)^T\)

Again compare \(C\), \(D\), \(E\) and \(F\) by subtraction. Which of \(C\), \(D\), \(E\) and \(F\) are equal?

2.8.5 Hand In Check List

Make sure that you hand in

- A diary file showing the use of \text{rref}
- A diary file for the calculations in section 2.8.3 and proofs where necessary.
- A diary file for section 2.8.4 with a clear indication of which of the matrices are equal.
2.9 Matrix Inverses and Partitioned Matrices

2.9.1 Building matrices using partitioning

Matrices in MATLAB can be built up by patching together smaller matrices into a big matrices. The smaller matrices must fit together exactly along rows and columns and not leave any spaces un-filled. For example type the following in MATLAB:

\[
A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \\
B = \begin{bmatrix} 7 \\ 8 \end{bmatrix} \\
C = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} \\
D = \begin{bmatrix} A,B;C,0 \end{bmatrix} \\
E = \begin{bmatrix} A',\text{zeros}(3,7);\text{eye}(2,6),A \end{bmatrix} \\
F = [[B,B]^2,B]
\]

All these cases should have worked fine. Try to understand why. Note that you can build matrices within matrices using all the commands like multiplication etc that work in MATLAB.

Now type

\[
D = [A,B;C]
\]

Why did this case fail?

Exercise (not for credit)

Use matrix partitioning techniques to enter the following matrices into MATLAB:

\[
A = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ 3 & 7 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix} \\
F = \begin{bmatrix} 2 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}
\]

2.9.2 Extracting parts of matrices

It is frequently the case that one wishes to extract part of one matrix and put it in another. In other words one wishes to extract a sub-matrix from a partitioned matrix. One way of doing this is to use a more complex version of the "::" command from last time to index certain rows and columns of a matrix.

The general form of the "::" command creates a vector of values. The form of this command is

\[j:k\] This is the same as \([j,j+1,j+2,\cdots,k]\)

\[j:k\] This is empty if \(j > k\).

\[j:i:k\] This is the same as \([j,j+i,j+2i,\cdots,k]\)

\[j:i:k\] This is empty if \(i > 0\) and \(j > k\) or \(i < 0\) and \(j < k\).

Try typing the following to MATLAB:

\[\text{Enter code here}\]
Notice that the numbers can be non-integers (but look at \(f\), it’s more difficult to tell what will happen if the numbers aren’t integers). You can use the colon operator to extract rows and columns from a matrix. For example, if \(B\) is a previously defined matrix then (don’t type this)

\[
A = B(2:5,1:2:7)
\]

will result in a matrix \(A\) with entries

\[
A = \begin{pmatrix}
    b_{2,1} & b_{2,3} & b_{2,5} & b_{2,7} \\
    b_{3,1} & b_{3,3} & b_{3,5} & b_{3,7} \\
    b_{4,1} & b_{4,3} & b_{4,5} & b_{4,7} \\
    b_{5,1} & b_{5,3} & b_{5,5} & b_{5,7}
\end{pmatrix}
\]

You can see that rows 2 through 5 of columns 1,3,5,7 have been copied to \(A\). If \(B\) is a \(7 \times 7\) matrix, the command

\[
A = B(:,7:-1:1)
\]

will copy \(B\) to \(A\) but reversing the order of the rows. For example, in MATLAB, type

\[
B = [4,1,0;1,4,1;0,1,4]
\]

\[
A = B(:,3:-1:1)
\]

\[
C = B(1:2,2:3)
\]

\[
D = B(:)
\]

Note that \(B(:)\) just converts \(B\) to a vector by stringing together the columns of \(B\).

**Exercise (not for credit)**

Enter the following matrix into MATLAB:

\[
A = \begin{pmatrix}
    3 & 2 & 1 & 0 \\
    2 & 3 & 2 & 0 \\
    1 & 2 & 3 & 2 \\
    0 & 1 & 2 & 3
\end{pmatrix}
\]

1. Set \(a\) to be the first column of \(A\) by extracting the column directly from \(A\) using colon notation.

2. Set \(b\) to be the matrix formed from rows 1 and 3 of \(A\) by extracting the submatrix directly from \(A\) using colon notation.

3. Set \(c\) to be the matrix formed by the last two columns and first two rows of \(A\).

### 2.9.3 The matrix inverse in MATLAB

Consider the problem of solving \(Ax = b\) where \(A\) is an \(n \times n\) matrix, \(b \in \mathbb{R}^n\) and the unknown vector is \(x \in \mathbb{R}^n\). If \(A\) is non-singular, \(x = A^{-1}b\). In MATLAB we can compute \(A^{-1}b\) very easily by simply typing \(A\backslash b\). In fact, \(b\) can be an \(n \times m\) vector and then \(A^{-1}b\) is an \(n \times m\) solution matrix and can be computed just by typing \(A\backslash b\).

In the same way, if \(b\) is an \(m \times n\) vector and \(A\) is a non-singular \(n \times n\) matrix we can compute \(bA^{-1}\) in MATLAB by simply typing \(b/A\). **Please Note** the difference between \(A\backslash b\) and \(b/A\).
As an example, we can solve
\[
\begin{pmatrix}
1 & 2 \\
3 & 4
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2
\end{pmatrix}
= \begin{pmatrix}
-1 \\
1
\end{pmatrix}
\]
by typing the following commands in to MATLAB:

\begin{verbatim}
A=[1,2;3,4]
b=[-1;1]
x=A\b
\end{verbatim}

**NOTE:** Use these commands with care. If MATLAB reports problems such as that the matrix may be singular, the solution MATLAB returns may be unreliable. Also \ will even give you a solution if you have an underdetermined system \(m\) equations in \(n\) unknowns with \(m < n\). But this solution has a special meaning (the least squares solution), so for this course, don’t use \ or / unless you have a square, non-singular matrix.

**Exercise (not for credit)**

Solve the following linear problem in MATLAB (enter the matrix using the special matrix functions and partitioning if you like):

\[
\begin{pmatrix}
3 & 2 & 0 & 0 & 0 & 1 \\
2 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 7 & 0 & 0 & 0 \\
0 & 0 & 0 & 7 & 0 & 0 \\
0 & 0 & 0 & 0 & 7 & 2 \\
0 & 0 & 0 & 0 & 1 & 3
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6
\end{pmatrix}
= \begin{pmatrix}
-1 \\
2 \\
3 \\
-2 \\
-1 \\
0
\end{pmatrix}
\]

How can you check that your computed solution is correct?
Chapter 3

Vector Spaces

3.1 Spanning Sets

It is sometimes necessary to show that two spaces are equal. If we know a basis, MATLAB can sometimes help. For example suppose \( U = \text{span} (u_1, u_2, \ldots, u_n) \) and suppose \( V = \text{span} (v_1, v_2, \ldots, v_n) \). How do we prove that \( U = V \)? It suffices to prove that for \( 1 \leq i \leq n \) \( u_i \in V \) and \( v_i \in U \) (prove this!).

So given a spanning set \((u_1, u_2, \ldots, u_n)\) and a vector \( v \) we need to know if \( v \in \text{span} (u_1, u_2, \ldots, u_n) \)

But this occurs if and only if there are constants \( a_1, a_2, \ldots, a_n \) such that

\[
v = a_1 u_1 + a_2 u_2 + \cdots + a_n u_n
\]

For example suppose

\[
U = \text{span} \left((2, -1, 3)^T, (1, 1, 2)^T\right)
\]

and \( v = (1, -5, 0)^T \). Then \( v \in U \) if and only if there are constants \( a_1 \) and \( a_2 \) such that

\[
a_1 \begin{pmatrix} 2 \\ -1 \\ 3 \end{pmatrix} + a_2 \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ -5 \\ 0 \end{pmatrix}
\]

We need to try to find \( a_1 \) and \( a_2 \), so we form the augmented matrix

\[
A = \begin{pmatrix} 2 & 1 & 1 \\ -1 & 1 & -5 \\ 3 & 2 & 0 \end{pmatrix}
\]

and use \texttt{rref} to reduce \( A \) to reduced row echelon form. If the resulting system is consistent then a suitable \( a_1 \) and \( a_2 \) exist and \( v \in U \). If the system is inconsistent then \( v \) is not in \( U \).

Determine if the following vectors are in the space \( U \)

1. \((1, 1, 1)^T \) (Answer is no)
2. \((1, 7, 4)^T \) (Answer is yes)

3.2 The Null Space

To use MATLAB to find a spanning set for the nullspace of a matrix \( A \), you can use \texttt{rref} to reduce \( A \) to reduced row echelon form and then write down the general solution.
For example, to find a spanning set for the matrix

\[
A = \begin{pmatrix}
0 & 0 & 1 & -4 \\
2 & -4 & -1 & -2 \\
4 & -8 & 0 & 12
\end{pmatrix}
\]

we enter \( A \) into MATLAB and use \texttt{rref} as follows:

```matlab
>> A=[0 0 1 -4;2 -4 -1 -2;4 -8 0 12];
>> rref(A)
ans =
1  -2  0  0
0   0  1  0
0   0  0  1
```

From this we can see that the spanning set has one vector:

\((2, 1, 0, 0)^T\).

### 3.2.1 Exercise on finding nullspaces using \texttt{rref} (not for credit)

You can compute a spanning set for the null-space of a matrix by entering an appropriate augmented matrix into MATLAB, using \texttt{rref} to reduce the matrix, and then writing down the general solution of the corresponding matrix problem. Use MATLAB to help you determine a spanning set for the null-space of each of the following matrices:

1. 

\[
\begin{pmatrix}
1 & 1 & -1 & 2 \\
2 & 2 & -3 & 1 \\
-1 & -1 & 0 & -5
\end{pmatrix}
\]

2. 

\[
\begin{pmatrix}
2 & 1 & -1 & 4 \\
3 & 4 & 6 & 1 \\
-1 & 2 & 8 & -7
\end{pmatrix}
\]

3. 

\[
\begin{pmatrix}
1 & -1 & 2 \\
3 & 2 & 1 \\
4 & 1 & 3 \\
1 & 4 & -3
\end{pmatrix}
\]

4. The matrix created by typing the MATLAB command

\[
A=0.01*\text{round}(100*\text{rand}(4))
\]

Do you expect this matrix to have a non-trivial null-space and why?
3.2.2 Exercise on null-spaces using Null (not for credit)

MATLAB has a built-in function for finding the null-space of a matrix $A$. The command is `null`. If $A$ is a matrix then

$$X = \text{null}(A)$$

will put a spanning set for the null-space of $A$ into the columns of $X$.

Repeat Exercise 3.2.1 parts 1) and 2) using the command `null`. In each case say if the results of `null` are different from your results, and say, giving reasons, if the results are consistent (i.e., span the same null-space). Hint: for each matrix, can each of the vectors in the spanning set you calculated via `rref` be expressed as a linear combination of the vectors computed using the `null` command?
Chapter 4

An Exercise on Partitioned Matrices

As we have already seen, the MATLAB command \texttt{rand(n)} returns an \( n \times n \) matrix of random numbers each uniformly distributed in the interval \([0, 1]\). The command \texttt{round(C)} rounds each entry in the matrix \( C \) to the nearest integer. As usual we use these commands to produce a matrix of random integers.

4.0.3 Exercise

This is a reworded version of question 10 on page 79 of your book - but you do not need to read the book to answer this exercise. Record your MATLAB answers in a diary file. The MATLAB commands

\[
A = \text{round}(10 \times \text{rand}(6))
B = A' \times A
\]

will result in a symmetric matrix \( B \) with integer entries. Compute another random matrix \( B \) in this way using MATLAB and verify that \( B \) is also symmetric (how do you check if a matrix is symmetric in MATLAB?). Prove mathematically that if \( A \) is any matrix then the matrix \( B = A^T A \) is always symmetric (write the proof on your MATLAB diary file output).

Next partition \( B \) into four \( 3 \times 3 \) submatrices. In other words

\[
B = \left( \begin{array}{cc} B_{11} & B_{12} \\ B_{21} & B_{22} \end{array} \right)
\]

where \( B_{11}, B_{12}, B_{21} \) and \( B_{22} \) are \( 3 \times 3 \) matrices. To determine the submatrices in MATLAB define

\[
B_{11} = B(1:3,1:3)
B_{12} = B(1:3,4:6)
\]

and define \( B_{21} \) and \( B_{22} \) in a similar manner using rows 4 through 6 of \( B \).

Compute \( C = \text{inv}(B_{11}) \) in MATLAB. \( C \) should be symmetric so check that \( C^T = C \) using MATLAB. Why must \( C \) be symmetric? Explain (hint: is \( B_{11} \) always symmetric?). Check also that \( B_{21}^T = B_{12} \) (why do we expect this?). Next set

\[
E = B_{21} \times C
F = B_{22} - B_{21} \times C \times B_{21}'
\]

Then use the MATLAB functions \texttt{eye} and \texttt{zeros} to construct

\[
L = \left( \begin{array}{cc} I & 0 \\ E & I \end{array} \right), \quad D = \left( \begin{array}{cc} B_{11} & 0 \\ 0 & F \end{array} \right).
\]

Using MATLAB compute \( H = LDL^T \) and compare it to \( B \) by computing \( H - B \). Prove that if all the computations had been done exactly, then \( LDL^T \) would equal \( B \) exactly.
4.1 Hand in check list

- From Exercise 4.0.3, hand in the diary file as directed. Also hand in explanations hand written as detailed in the text. Be careful to answer all the questions and hand in all requested proofs.
Chapter 5

Exercise on Magic Matrices

5.1 Introduction

The purpose of this exercise is to give you more practice with three things:

- Setting up a problem as a linear system (in other words writing down a suitable linear system).
- Using MATLAB to obtain reduced row echelon form.
- Interpreting the MATLAB output so as to write down the general solution to each problem (assuming one exists).

5.2 Magic Matrices

A magic matrix is a square matrix with integer entries in which all rows, columns and the two diagonals have the same sum.

For example the matrix

\[
\begin{pmatrix}
8 & 1 & 6 \\
3 & 5 & 7 \\
4 & 9 & 2 \\
\end{pmatrix}
\]

is a magic matrix in which each row adds up to 15, each column adds up to 15 and the sum of the diagonals is \(8 + 5 + 2 = 15\) and \(6 + 5 + 4 = 15\).

Suppose you are given a partial magic matrix. Can you fill in the rest? It depends what you are given. Let’s look at an example

\[
\begin{pmatrix}
8 & 1 & 3 \\
? & ? & ? \\
? & ? & ? \\
\end{pmatrix}
\]

Is there a magic matrix with this top row? We can answer the question by using linear algebra. We start by labeling all the unknown entries

\[
\begin{pmatrix}
8 & 1 & 3 \\
x_1 & x_2 & x_3 \\
x_4 & x_5 & x_6 \\
\end{pmatrix}
\]

Now we can write lots of linear equations that must be satisfied by \(x_1, \cdots, x_6\). From the top row, we know that the rows, columns and diagonals of the matrix must add to 12. So

\[
x_1 + x_2 + x_3 = 12 \\
x_4 + x_5 + x_6 = 12
\]
\[
\begin{align*}
8 + x_1 + x_4 &= 12 \\
1 + x_2 + x_5 &= 12 \\
3 + x_3 + x_6 &= 12 \\
8 + x_2 + x_6 &= 12 \\
3 + x_2 + x_4 &= 12
\end{align*}
\]

This is a system of seven equations in six unknowns so it’s overdetermined and we might not have a solution. To find out we write down the augmented matrix
\[
\begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 12 \\
0 & 0 & 0 & 1 & 1 & 1 & 12 \\
1 & 0 & 0 & 1 & 0 & 0 & 4 \\
0 & 1 & 0 & 0 & 1 & 0 & 11 \\
0 & 0 & 1 & 0 & 0 & 1 & 9 \\
0 & 1 & 0 & 0 & 0 & 1 & 4 \\
0 & 1 & 0 & 1 & 0 & 0 & 9 \\
\end{pmatrix}
\]

Now we can reduce this matrix to reduced row echelon form in MATLAB and hence determine if there is a solution and if it is unique. Using \texttt{rref} or \texttt{rrefmovie} we find that the corresponding reduced row echelon form is
\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 1 & 0 & 0 & 0 & 0 & 4 \\
0 & 0 & 1 & 0 & 0 & 0 & 9 \\
0 & 0 & 0 & 1 & 0 & 0 & 5 \\
0 & 0 & 0 & 0 & 1 & 0 & 7 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

So the only possibility is
\[
\begin{pmatrix}
8 & 1 & 3 \\
-1 & 4 & 9 \\
5 & 7 & 0 \\
\end{pmatrix}
\]

Often people won’t allow negative entries in magic matrices, but we shall. Also some people require that you use each integer from 1 to 9 only once. So this the magic matrix we have found fails on both accounts.

### 5.2.1 Exercise

Record this exercise in a diary file. Hand in your hand calculations where you set up the linear system. Explicitly write on your diary file if you succeeded in finding the magic matrix requested and if it is unique.

\[
\begin{pmatrix}
1 & ? & ? \\
? & 2 & ? \\
? & ? & 3 \\
\end{pmatrix}, \quad
\begin{pmatrix}
1 & ? & 3 \\
? & 2 & ? \\
? & ? & ? \\
\end{pmatrix}, \quad
\begin{pmatrix}
1 & ? & 3 \\
2 & ? & ? \\
? & ? & ? \\
\end{pmatrix}, \quad
\begin{pmatrix}
1 & ? & ? \\
? & 3 & 2 \\
1 & ? & ? \\
\end{pmatrix}
\]


### 5.3 Hand in check list

Make sure you hand in annotated diary output for the following exercise:

- The magic matrices requested in exercise 5.2.1. Make sure you hand in written material in which you derive the linear system in each case. Make sure you say if you failed in any of the cases, also make sure you explicitly write down the general form of the magic matrices you find.
Chapter 6

Linear Transformations

6.1 Linear Maps from the plane to the plane

A linear transformation \( L : \mathbb{R}^2 \rightarrow \mathbb{R}^2 \) can be represented (with respect to the standard basis) as a two by two matrix \( A \). We can visualize the transformation geometrically if we look at the effect of the transformation on a set of vectors. Suppose a geometrical shape (for example a rectangle) has vertices with coordinate vectors \( v_1, v_2, \ldots, v_n \) with respect to the standard basis. If we apply \( L \) to each position vector we get a new set of coordinate vectors \( Av_1, Av_2, \ldots, Av_n \). These coordinate vectors describe a new transformed figure.

Note that a straight line connecting two vertices \( v_1 \) and \( v_2 \) is transformed into a straight line connecting the two transformed vertices \( Av_1 \) and \( Av_2 \). Why?

The purpose of this exercise is for you to investigate the geometry of linear transformations from \( \mathbb{R}^2 \) to \( \mathbb{R}^2 \). To do this you can use a MATLAB program called \texttt{plane2plane} which is based on the program \texttt{planelt} written by D. Hill of Temple University. \texttt{Plane2plane} allows you to

- Read an introduction and description of the code
- Choose a simple figure to transform (ie. vectors \( v_1, \ldots, v_n \)).
- Transform the figure using built in linear transformations or use your own linear transformation (represented by a matrix).
6.1.1 Elementary Matrices

Let us see what the geometrical effect of elementary matrices is. Start plane2plane and do the following:

- Choose the “flag” figure. Then apply the elementary matrix corresponding to interchange of rows of a matrix:

\[ E = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \]

What is the geometric effect of this transformation? Is it a built in transformation in plane2plane? Try some other figures until you are convinced that you understand this transformation.

- Again choose the “flag” figure. Then apply the elementary row scaling matrix

\[ E = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \]

What does it do in geometric terms? Identify the corresponding plane2plane built in transformation. What does the elementary matrix

\[ E = \begin{pmatrix} 1 & 0 \\ 0 & 0.5 \end{pmatrix} \]

do? Again try these matrices with other figures (the rectangle for example).

- Choose the “rectangle” figure. Then apply the elementary matrix

\[ E = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \]

What is this type of transformation? Try this transformation on other figures and identify it in plane2plane’s builtin transformations.

6.2 Composition of Matrices

If

\[ A = \begin{pmatrix} 1 & -1 \\ 2 & 0.5 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \]

then

\[ C = AB = \begin{pmatrix} 0 & -1 \\ 2.5 & -1 \end{pmatrix} \]

Choose any figure of your choice. First apply C directly to the figure and see the result. Now start again with the same figure. First apply B and then A. You should find out that applying B then A is the same as applying C alone (can you be sure by just looking at one figure?). You can see that matrix multiplication was defined to agree with composition of transformations (ie if \( L_A \) is the transformation corresponding to A, \( L_B \) corresponding to B and \( L_C \) corresponding to C then

\[ L_C = L_A \circ L_B \quad \text{and} \quad C = AB \]

Now select another figure and apply C followed by \( A^{-1} \) and then \( B^{-1} \). You should end up where you start. This is a demonstration that

\[ C^{-1} = B^{-1}A^{-1} \]
6.2.1 Does Matrix Multiplication Commute

Choose a figure and apply the matrix

\[ A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \]

followed by the matrix

\[ B = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \]

Note down the result of the successive application of these transformations. Now start over with the same figure and apply \( B \) followed by \( A \). Do you end up with the same figure at the end? Do this for a few different figures until you are convinced that you do not always end up at the same figure. This is an example that \( AB \neq BA \) in general.

6.2.2 Null spaces

Suppose \( A \) has a non-trivial null space. How does this appear geometricaly? Suppose

\[ A = \begin{pmatrix} 1 & 2 \\ -1 & -2 \end{pmatrix} \]

Verify that this matrix has the null-space

\[ N(A) = \text{span} \left\{ \begin{pmatrix} -2 \\ 1 \end{pmatrix} \right\} \]

Apply \( A \) to some figures of your choice and describe what happens.

6.3 Invariant vectors

Sometimes there will be vectors with are unchanged by matrix multiplication. For example enter the line connecting the origin \((0,0)^T\) and \((1,1)^T\) as a figure in plane2plane and apply

\[ A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \]

to it. The line should not move. This case is a little unusual since the line did not change length. More commonly, there are some vectors which when multiplied by the matrix remain parallel to the original vector but change length. For example take

\[ A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \]

Enter the line connecting the \((0,0)^T\) origin and \((1,1)^T\) again as your figure. Apply \( A \) to this figure. You should see that the transformed figure is a straight line parallel to \((1,1)^T\) but three times as long. So we have verified geometricaly that

\[ \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 3 \begin{pmatrix} 1 \\ 1 \end{pmatrix} \]

What does the matrix \( A \) do to the vector \((-1,1)^T\)? In general if a vector \( \mathbf{x} \) and a scalar \( \lambda \) is such that

\[ A \mathbf{x} = \lambda \mathbf{x} \]

(the transformed vector is \( \lambda \) times as long as the original vector), we say that \( \mathbf{x} \) is an eigenvector for \( A \) with corresponding eigenvalue \( \lambda \).
6.4 Application of Linear Algebra to Electricity

6.4.1 MATLAB

There is one more MATLAB command that might help you with this assignment. To create an $n \times n$ diagonal matrix $D$ with diagonal entries $D_{i,i} = v_i$ you create a vector $v$ then type (in MATLAB) $D=\text{diag}(v)$. Try the following command:

\[
v=[-1,2,3,-3] \\
\text{diag}(v)
\]

6.4.2 Circuit Analysis 1

Utility companies, and anyone designing electrical apparatus, needs to compute the voltage and current in each part of the electrical network. This assignment concerns a very simple electrical network consisting only of batteries and resistors. Figure 6.1 shows an example of an electrical circuit. The symbols have the following meaning:

A path along which charge may flow

A battery. This is a source of e.m.f. (electromotive force) that makes charge flow. The strength of this force is measured in Volts (symbol: $V$).

A resistor. This causes a voltage drop (it resists charge movement). The unit of resistance is the Ohm (symbol: $\Omega$).

There are other interesting circuit elements (for example the inductance) but we shall ignore them. An example of a circuit is shown in Figure 6.1.

Given a circuit, the problem is to find the voltage drop across each resistor and the current in each part of the circuit. To do this you need to know three laws from physics.

**Ohms Law** This states that the voltage drop $V$ Volts across a resistor is related to the resistance $R$ Ohms and current flowing in the resistor denoted $i$ Amps by the following equation:

\[
V = iR.
\]
Kirchoff’s first law At every node (meeting of circuit lines) in the circuit the sum of the incoming currents must equal the sum of the outgoing currents.

Kirchoff’s second law Around any closed loop in the circuit, the sum of the voltages in the loop must equal the applied e.m.f. (from batteries for example).

To analyze the circuit in Figure 6.1, we first label all the nodes as in Figure 6.2 (A, B etc). Then we assign currents (with direction) along each circuit path connecting nodes. For example $i_1$ denotes the current in the path from node C to node A via the battery (see Figure 6.2). Now we can apply Kirchoff’s first law at the nodes. Remember that you must take into account if the current flows into or out of a particular node. We obtain:

At node A: $i_1 = i_2 + i_5$, \hspace{1cm} (6.2)
At node B: $i_2 = i_3 + i_6$, \hspace{1cm} (6.3)
At node C: $i_3 = i_4 + i_1$, \hspace{1cm} (6.4)
At node D: $i_4 + i_5 + i_6 = 0$. \hspace{1cm} (6.5)

Next we apply Kirchoff’s second law. The little battery symbol tells you which sign to make the applied voltage (e.m.f.). It’s plus in the direction from the small line to the big line! (which is the direction we have chosen for $i_1$). We use the notation that $V_k$ is the voltage across resistor number $k$ and the direction for this voltage is the same as for the current (so that Ohms law says $V_k = R_k i_k$ where $R_k$ is the resistance of the $k$th resistor. To apply Kirchoff’s second law you need to choose some loops in the circuit. There are three simple loops ABD, BCD and CAD. Applying the second law to each loop we obtain:

In circuit ABD: $V_2 - V_5 + V_6 = 0$ \hspace{1cm} (6.6)
In circuit BCD: $V_3 + V_4 - V_6 = 0$ \hspace{1cm} (6.7)
In circuit CAD: $V_5 - V_4 + V_1 = 10$ \hspace{1cm} (6.8)

Notice that in writing down these equations, if the direction of the voltage is against the direction of the loop, we count the voltage as minus. Finally, we use Ohms law to eliminate the voltages. Remember that
you must get the sign of the voltage and currents to agree. Thus

\[ V_1 = i_1, \quad V_2 = 2i_2 \]
\[ V_3 = 3i_3, \quad V_4 = 4i_4 \]
\[ V_5 = 5i_5, \quad V_6 = 6i_6. \]

Using these equations to eliminate the voltages in (6.4.2) gives us the following equations

\[ 2i_2 - 5i_5 + 6i_6 = 0 \] \quad (6.9)
\[ 3i_3 + 4i_4 - 6i_6 = 0 \] \quad (6.10)
\[ i_1 - 4i_4 + 5i_5 = 10. \] \quad (6.11)

The seven equations (6.4.2) and (6.4.2) have six unknowns \( i_1, \ldots, i_6 \). We can show that equations determined in this way are always consistent and that we can solve them uniquely by reduction to reduced row echelon form (in fact any three of (6.4.2) and all of (6.4.2) make a consistent square system). So all that's left to do is enter the seven equations into MATLAB as an augmented matrix and then use \texttt{rref} to find the currents.

**Exercise**

For each of the circuits below do the following (and hand it in):

- By hand, apply a Kirchoff analysis to the circuit **writing down the equations for the unknown currents**. You must hand in the equations written neatly on a piece of paper, and a diagram showing how you labelled the nodes and assigned the currents.

- Use MATLAB to find the unknown currents (record this part in a diary file). Make sure you mark clearly your prediction of the currents.

1. Analyze the following circuit

2. Analyze the following circuit
6.4.3 More on circuits

The analysis presented in the previous section is correct but suffers from the problem that if you change the resistance anywhere then the whole calculation has to be redone. In this section we shall show how the “topology” (or connections) in the network can be separated from the values of the elements in the network.

First we explicitly label all the “loop” currents. These are currents that we suppose flow around each closed circuit in the network. Figure 3 shows the loop currents $\alpha, \beta$ and $\gamma$ marked on our example network. Notice that we have also given a direction to the loop current (clockwise or counterclockwise). Our choices are summarized in the next table:

- $\alpha \equiv$ Current in loop $ABD$ (clockwise)
- $\beta \equiv$ Current in loop $BCD$ (clockwise)
- $\gamma \equiv$ Current in loop $ADC$ (clockwise)

The currents in each branch of the circuit are the sum (with the correct sign) of the loop currents that go through the branch. For example $i_1$ is caused only by loop current $\gamma$ and is in the same direction so $i_1 = \gamma$. But $i_5$ receives contributions from $\alpha$ and $\gamma$. Taking into account the directions of the branch and loop currents we have $i_5 = \gamma - \alpha$. Altogether we obtain

$$
\begin{align*}
  i_1 &= \gamma \\
  i_2 &= \alpha \\
  i_3 &= \beta \\
  i_4 &= \beta - \gamma \\
  i_5 &= \gamma - \alpha \\
  i_6 &= \alpha - \beta.
\end{align*}
$$

We can write these equations in matrix form

$$
\begin{pmatrix}
  i_1 \\
  i_2 \\
  i_3 \\
  i_4 \\
  i_5 \\
  i_6
\end{pmatrix} =
\begin{pmatrix}
  0 & 0 & 1 \\
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 1 & -1 \\
 -1 & 0 & 1 \\
  1 & -1 & 0
\end{pmatrix}
\begin{pmatrix}
  \alpha \\
  \beta \\
  \gamma
\end{pmatrix}
$$

(6.12)
If we define
\[ \mathbf{J} = \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix}, \quad \mathbf{I} = \begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \end{pmatrix} \] and \[ \mathbf{M} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix}. \]

We may summarize the relationship between branch and loop currents in (6.12) as \( \mathbf{I} = \mathbf{M} \mathbf{J} \). This is essentially Kirchhoff’s first law.

Next we must use Kirchhoff’s second law. Let \( V_1, \ldots, V_6 \) be the voltages corresponding to \( i_1, \ldots, i_6 \). Using Kirchhoff’s second law as usual on each loop we have
\[
\text{Loop } \alpha : \quad V_2 - V_5 + V_6 = 0 \\
\text{Loop } \beta : \quad V_3 + V_4 - V_6 = 0 \\
\text{Loop } \gamma : \quad V_1 - V_4 + V_5 = 10
\]

We define the vector \( \mathbf{V} \) of unknown voltages and the vector \( \mathbf{e} \) of known e.m.f.s as follows
\[
\mathbf{V} = \begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \end{pmatrix}, \quad \mathbf{e} = \begin{pmatrix} 0 \\ 0 \\ 10 \end{pmatrix}.
\]

The we can express \( \mathbf{e} \) in terms of \( \mathbf{V} \) by writing (6.4.3) as
\[
\mathbf{e} = \begin{pmatrix} 0 & 1 & 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 1 & 0 & -1 \\ 1 & 0 & 0 & -1 & 1 & 0 \end{pmatrix} \mathbf{V}.
\]

Thus there is a matrix \( \mathbf{P} \) such that \( \mathbf{e} = \mathbf{P} \mathbf{V} \). Compare \( \mathbf{P} \) and \( \mathbf{M} \). You can see that \( \mathbf{P} = \mathbf{M}^T \) and this can be proved always to be true. Thus
\[
\mathbf{e} = \mathbf{M}^T \mathbf{V}
\]
Finally, we need to relate \( V \) and \( I \). This is where Ohm’s law comes in. We obtain

\[
V = \begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \end{pmatrix} = \begin{pmatrix} i_1 \\ 2i_2 \\ 3i_3 \\ 4i_4 \\ 5i_5 \\ 6i_6 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 6 \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \end{pmatrix}
\]

Thus there is a diagonal matrix \( R \) such that \( V = RI \). We have show that in this special case (and in fact in general for simple planar circuits of this type)

\[
I = MJ, \quad e = M^T V, \quad \text{and} \quad V = RI
\]

hence \( e = M^T RMJ \) (prove it - but there is no need to hand in the proof). We can thus compute the unknown currents by solving this equation or \( J = (M^T RM)^{-1} e \).

In summary to use the loop current method we do the following steps:

- Label the loop currents and branch currents.
- Write down \( M \)
- Write down \( e \)
- Write down \( R \)
- Compute \( J = (M^T RM)^{-1} e \).

We have succeeded in decoupling the circuit topology \( (M) \) from the circuit elements \( (R) \).

**Exercise**

For this exercise make sure you hand in clearly marked diagram on which you have drawn the loop currents you have used, the label you gave them (eg. \( \alpha \) etc), and their direction (with an arrow). You must hand in a diagram to get credit for this part of the assignment.

1. Do part (1) of the exercise in § 6.4.2 by the loop current method outlined above. First write down by hand the matrices concerned. Then have MATLAB solve the resulting linear system (record this in a diary file). Finally write down your prediction for the currents.

2. Analyze the following circuit which has five loop currents:
6.4.4 **Hand in check list**

Hand in the solution to the exercises in section § 6.4.2 and § 6.4.3. Each exercise asks you to analyze two circuits. For each circuit, make sure you hand in

- Hand written derivations of the equations and matrices you solve.
- MATLAB diary file output showing the use of MATLAB to solve the relevant equations.
- A clear statement of your solution

6.5 **Acknowledgement**

Some of the exercises here, and the example are from the SMP Further Mathematics Series: III Differential Equations and Circuits, Cambridge University Press.
Chapter 7

Orthogonality

7.1 Orthogonal Subspaces

1. Use MATLAB to help you write down a basis for each of the spaces \( R(A^T) \), \( N(A) \), \( R(A) \) and \( N(A^T) \) if

\[
A = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
1 & 1 & 2 & 2 \\
\end{pmatrix}
\]

Use MATLAB to help verify that in this case \( N(A) = (R(A^T))^\perp \) and \( N(A^T) = (R(A))^\perp \).

2. Use MATLAB to compute a basis for \( S^\perp \) if

\[
S = \text{Span} \left\{ \begin{pmatrix} 1 \\ 2 \\ -1 \\ 3 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \\ 3 \\ 6 \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \\ 2 \\ 9 \end{pmatrix} \right\}
\]

45
7.2 The Gram Schmidt Process

Find an orthonormal basis for the subspace $S$ of $\mathbb{R}^4$ spanned by

$$
\begin{pmatrix}
3 \\
4 \\
0 \\
1
\end{pmatrix}, \quad 
\begin{pmatrix}
1 \\
0 \\
1 \\
1
\end{pmatrix}, \quad 
\begin{pmatrix}
2 \\
-1 \\
0 \\
1
\end{pmatrix}
$$

Use the Gram-Schmidt algorithm in MATLAB

**Step 1** In MATLAB set

$$
x_1 = \begin{pmatrix}
3 \\
4 \\
0 \\
1
\end{pmatrix}
$$

Then set

$$
u_1 = \frac{x_1}{\|x_1\|}
$$

**Hint:** Use the MATLAB `norm` command to compute $\|x\|$. You should obtain the following answer in MATLAB for $u_1$

<table>
<thead>
<tr>
<th>$u_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5883</td>
</tr>
<tr>
<td>0.7845</td>
</tr>
<tr>
<td>0.1961</td>
</tr>
</tbody>
</table>

**Step 2** Input $x_2$ into MATLAB and compute

$$
<x_2, u_1> = x_2^T u_1
$$

then compute

$$
p_1 = <x_2, u_1> u_1
$$

and compute the temporary vector $y_2 = x_2 - p_1$. Finally compute

$$
u_2 = \frac{y_2}{\|y_2\|}
$$

You should obtain

<table>
<thead>
<tr>
<th>$u_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3487</td>
</tr>
<tr>
<td>-0.3985</td>
</tr>
<tr>
<td>0.6476</td>
</tr>
<tr>
<td>0.5479</td>
</tr>
</tbody>
</table>

**Step 3** Input $x_3$ into MATLAB and compute

$$
<x_3, u_1>, \text{ and } <x_3, u_2>
$$

Hence compute

$$
p_2 = <x_3, u_1> u_1 + <x_3, u_2> u_2
$$

Finally find $y_3$ and $u_3$ by

$$
y_3 = x_3 - p_2, \text{ and } u_3 = \frac{y_3}{\|y_3\|}
$$

You should obtain
u3 =

0.6290 
-0.4694 
-0.6196 
0.0094

Now compute an orthonormal basis for the subspace of \( \mathbb{R}^5 \) spanned by

\[
\begin{pmatrix}
3 \\
4 \\
0 \\
1 \\
2 \\
-1
\end{pmatrix}, \begin{pmatrix}
0 \\
4 \\
-2 \\
1 \\
2 \\
-1
\end{pmatrix}, \begin{pmatrix}
3 \\
5 \\
0 \\
1 \\
-2 \\
-1
\end{pmatrix}
\]
Now I can let you in to a secret. MATLAB has a single command for computing and orthogonal basis from a given basis. It is \texttt{orth}. Suppose $S = \text{Span}\{v_1, v_2, \cdots, v_n\} \subset \mathbb{R}^n$ then we form the matrix $A = (v_1|v_2|\cdots|v_n)$. Then the columns of $\text{orth}(A)$ form an orthonormal basis for $S$. Try using \texttt{orth} to compute an orthonormal basis for the subspace of $\mathbb{R}^5$ discussed in the previous question. Is it the same basis that you computed via Gram-Schmidt? Do you think it is a valid orthonormal basis (and how would you check)? What is $A^T A$?

Let $A_1$ and $A$ be the matrices defined by the following MATLAB commands (which you should type in to MATLAB now)

\begin{verbatim}
A1=round(20*(rand(3,5)-0.5))
A=A1'*A1
\end{verbatim}

We expect the rank of $A_1$ to be 3. Check this using the MATLAB \texttt{rank} command (ie type \texttt{rank(A1)!}). What is the rank of $A$? Find an orthonormal set with the same span as the columns of $A$ (how many vectors will be needed?) using the MATLAB \texttt{orth} command as follows:

\begin{verbatim}
B=orth(A)
\end{verbatim}

Now find an orthonormal basis for the null space of $A^T$ (ie $\text{N}(A^T)$) by typing

\begin{verbatim}
C=null(A')
\end{verbatim}

Let the matrix $D$ be defined by

\begin{verbatim}
D=[B,C]
\end{verbatim}

Check (using MATLAB) that $D$ is orthogonal. Why must this be the case (regardless of the particular matrix $A$)?
Chapter 8

Programming MATLAB.

Today we are going to learn some of MATLAB’s programming language so that you can combine the matrix analysis techniques you already know into more sophisticated linear algebra programs.

8.1 MATLAB functions

You can write your own functions in MATLAB. Functions are stored in files, and created using the MATLAB editor.

A function file should be called the same name as the function it contains (but with the addition of the extension .m). Also the first word of the first line of the function must be function. For example, suppose we want to write a function that computes the average value of all the entries in a matrix. We want to be able to type

\[
\text{average}(X)
\]

in MATLAB to get the required average. First we need to sum together all the entries of \(X\). If we type \(\text{sum}(X)\) in MATLAB we will get a vector consisting of the sum of each column of \(X\) (type \text{help sum} to get information on \text{sum}). Then \(\text{sum}((\text{sum}(X)))\) will compute the desired sum of entries. Finally we need to divide this by the number of entries in the matrix. Thus we create a disk file called average.m (using the MATLAB editor) which might look like the following:

```matlab
function y=average(X)
% AVERAGE Compute the average of the entries in the matrix X. To
% use this function, you type average(X).
[m,n]=size(X);
nentries=m*n;
y=sum(sum(X))/nentries;
```

If you haven’t already done so, create the file average.m using the matlab editor (in matlab type edit average.m) and then test the function on the matrix

\[
A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}
\]

which has the average value 2.5. Also type \text{help average}.

Let me explain the average value function. The first line of the function file tells MATLAB that the file is a function called average which takes a single argument (denoted \(x\)) and returning a single result called \(y\). These are dummy variables only, and may be replaced by other variables when the function is being used (for example \(\text{av=average}(A)\)). The lines beginning \% are comment lines that are there to help humans and are ignored by MATLAB. The first group of comment lines in a function are special in that MATLAB displays them when you ask for \text{help} on the function. Later comments are not displayed. The line \([m,n]=\text{size}(X)\)
uses the MATLAB \texttt{size} function to find the number of rows and columns in $X$ ($m$ is the number of rows, $n$ is the number of columns). Note that the \texttt{size} function returns a vector of values. The rest of the function should be obvious.

Note that the variables $n$, $m$ and \texttt{nentries} are local to the function \texttt{average}. So you can't see them from within your MATLAB session and they won't interfere with variables that have the same name in your MATLAB session.

### 8.2 Looping

Sometimes it is useful to repeat a group of statements many times. To do this you can use a \texttt{for} loop which has the general form

\begin{verbatim}
for v=e
    <statements>
end
\end{verbatim}

where $v$ is a variable (or counter), $e$ is a MATLAB expression and $<$statements$>$ are any number of valid MATLAB statements. The variable takes on successive values of the expression, and for each value of $v$ the $<$statements$>$ are executed. There are some rather obscure forms of the \texttt{for} statement, but usually the expression $e$ is a vector formed using the colon 
"\texttt{\:\:\texttt{:}}\" notation. For example the MATLAB code

\begin{verbatim}
for i=1:5
    a(i)=i^2
end
\end{verbatim}

results in $i$ taking on the values 1, 2, 3, 4, 5 and so the result is equivalent to

$a=[1,4,9,16,25]$  

(try typing the above \texttt{for} loop into MATLAB and verify the claim). Actually, the \texttt{for} loop is a very bad way to calculate $a$ in this case since it would be quite slow compared to executing the statement $a=[1,4,9,16,25]$. In general you should try to avoid using \texttt{for} loops in MATLAB but sometimes these loops are essential.

Let us discuss another example. Suppose

$a=\text{rand}(25)$  

(type the statement in MATLAB) and that you want to compute the average value of the entries of $a$. We saw one way (using \texttt{sum}) before. Here is another way using nested for loops like this

\begin{verbatim}
s=0
for i=1:25
    for j=1:25
        s=s+a(i,j);
    end
end
average=s/25^2
\end{verbatim}

(Type in this code and see if it works). The \texttt{for} loops successively sums up all the entries in $a$ storing the result in the variable $s$. I have used a semi-colon 
\";\" after the statement in the loops since we don't want to see all the partial sums as the \texttt{for} loops execute. It turns out that the method using \texttt{sum} is better since it is much faster.

As a final example, we use functions and looping to write a function to return the factorial of a number. Here is the code
function y=factorial(x)
% FACTORIAL This function computes the factorial of x assuming that
% x is an integer (if x is not an integer, the answer will
% be wrong!)
y=1;
for j=1:x
    y=y*j;
end

I hope this function is easy to understand! All we do is successively multiply integers until we reach x. I have made this function available to you, so try it. For example type

factorial(3)

and the answer should be 6!

8.3 Logic

In order to write MATLAB programs you need to be able to control the flow of execution through a program. There are two useful constructs in MATLAB for this purpose: if-else-end and while. Since if is more important than while we shall discuss it first.

The general form of the if statement is

if <expression>
    <statements>
end

where the <statements> are executed provided all the elements in the <expression> are non-zero (true). Usually the expression will involve relational and logical operators. For example, create the following script called ifexample.m by typing
edit ifexample.m

from within matlab and type in the following code:

n=input('Input a number ')
if n == 6
    disp('the number is 6')
else
    if n < 6
        disp('the number is less than 6')
    end
    if n > 6
        disp('the number is greater than 6')
    end
    if n ~= 6
        disp('the number does not equal 6')
    end
end

Now run the script by typing ifexample. The code compares your number n to 6 and if the comparison is true, an informative message should be printed. Notice that I used the input function to prompt you to input a number. input is very useful in MATLAB programming (as usual type help input to get more information). The full set of relational operators is
Less than

Less than or equal

Greater than

Greater than or equal

Equal

Not equal

You can also use the more complex construction

if <expression>
  <statements>
elsif <expression>
  <statements>
else
  <statements>
end

You may need to combine relational operators. For example if you want to program the statement if a is less than 10 or a equals 12 add five to a you would type

if a < 10 | a == 12
  a=a+5;
end

The logical operator | is **or**. MATLAB has three logical operators:

| AND
| OR
| NOT

The remaining MATLAB flow control statement is **while** which has the general form

while <expression>
  <statements>
end

This construct executes the <statements> for as long as all the elements of the <expression> are non zero (or true). We can combine **while** and **for** constructs as the next example shows. Create the script **whileexample.m** by typing

```
edit whileexample.m
```

from matlab and entering the following lines of code:

```
n=0;
while n < 27
  n=input(' Type a number ');
  if n < 6
    disp(' Your number is less than 6')
  else
    disp(' Your number is greater than or equal to 6')
  end
end
```

What must you type to stop the loop? Now run the program by typing **whileexample**. To stop the program input a number bigger than or equal to 27.
8.3.1 Exercise on looping

Use a for loop to add the following sum in MATLAB

\[ \sum_{j=0}^{10} \frac{2^j}{j!} \]

Hand in a diary file showing the MATLAB code you used and the result. Verify that your result is close to \( e^2 \) (which you can compute in MATLAB using the internal function \( \text{exp} \) by typing \( \text{exp}(2) \)).

8.3.2 Exercise on functions

Recall that the function \( e^x \) can be defined by the Taylor series

\[ e^x = \sum_{j=0}^{\infty} \frac{1}{j!} x^j. \]

If \( A \) is a square matrix, we can also define the exponential of a square matrix by the same formula. So if \( A \) is a square matrix we define

\[ e^A = \sum_{j=0}^{\infty} \frac{1}{j!} A^j \]

with the convention that \( A^0 = I \). We shall prove later in the course that this series is well defined. Note that the \((i, j)\) entry of \( e^A \) is NOT \( e^{a_{i,j}} \).

The series is not very useful for computing \( e^A \), but we can approximate \( e^A \) by adding the first few terms of the series

\[ e^A \approx \sum_{j=0}^{N} \frac{1}{j!} A^j \quad (8.1) \]

where \( N \) is a sufficiently big integer. Write a MATLAB function called \( \text{matexp} \) with the following first line:

function \( B=\text{matexp}(A,N) \)

Given \( A \) and \( N \) the function \( \text{matexp} \) should compute the approximate exponential \( B \) using (8.1) so that

\[ B = \sum_{j=0}^{N} \frac{1}{j!} A^j. \]

Test your function using the matrix

\[ A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \]

which has the exponential (accurate to 16 decimal places)

\[ e^A = \begin{pmatrix} 11.40190937582335 & 8.68362754736431 \\ 8.68362754736431 & 11.40190937582336 \end{pmatrix}. \]

Use your function with \( N = 2, 4, \ldots \) until you obtain an approximation to \( e^A \) that is gives the first four numbers in each decimal correctly. What is the value of \( N \)?

Using the values of \( N = 11 \) compute an approximation to \( e^A \) for the following three matrices.

1. \[ A = \begin{pmatrix} 3 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 3 \end{pmatrix} \]
2. 

\[ A = \begin{pmatrix} 5 & 2 & 1 \\ -1 & 3 & 2 \\ 1 & 1 & 1 \end{pmatrix} \]

3. \( A = 0.01 \times \text{round}(100 \times \text{rand}(6)) \)

Now you have done all this work let me tell you that this way of approximating \( e^A \) is VERY BAD. It’s time consuming and prone to inaccuracy. MATLAB already has a built-in function for doing the required calculation. The function is called \texttt{expm}. If you type \texttt{expm(A)} MATLAB will return \( e^A \). Use the MATLAB function \texttt{expm} to calculate the exponential of each of the matrices in this question. Was your estimate of the error in your approximate matrix exponential (from \texttt{matexp}) a value good in each case?

8.3.3 Exercise on MATLAB logic

Write a MATLAB script that prompts the user for three numbers \( a, b \) and \( c \). If \( b^2 - 4ac \geq 0 \) the script should compute the two solutions of the quadratic equation

\[ ax^2 + bx + c = 0 \]

which are (of course)

\[ x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad x = \frac{-b - \sqrt{b^2 - 4ac}}{2a}. \]

(here the MATLAB function \texttt{sqrt} may be useful). If no real solution exists (ie. \( b^2 - 4ac < 0 \)) the program should print out a suitable informative message. The script should continue prompting for numbers until the user inputs all three numbers as zero in which case it should stop. You might want to think what happens if \( a = 0 \) (or if \( a = b = 0 \)).

Run your script and input the data

\[
\begin{align*}
& a = 1, \ b = -3, \ c = 2 \\
& a = 1, \ b = 0, \ c = 1 \\
& a = 2, \ b = -4, \ c = 2 \\
& a = 0, \ b = 1, \ c = 3
\end{align*}
\]

Hand in a diary file of the output and a printout of your script file.

8.4 Hand-In Check list

There are three questions in this assignment:

- Exercise 8.3.1. Hand in a diary file for your MATLAB loop and for the output.
- Exercise 8.3.3. Hand in a printout of the script file and a diary file showing the requested computations.
- Exercise 8.3.2. Hand in a printout of your function file and a diary file showing the use of \texttt{matexp} and \texttt{expm} on each example. Write clearly your predicted error and also the actual error in each case.
Chapter 9

The matrix inverse (explicitly) and the determinant

9.1 Computing the Inverse of a Matrix

It is rare in practice that you need to compute the inverse of a matrix. If you do need to compute $A^{-1}$ then the MATLAB command \texttt{inv(A)} will compute the inverse of a square $(n \times n)$ matrix $A$ provided the matrix $A$ is non-singular (at least to the accuracy MATLAB is capable of calculating).

9.1.1 Exercise (not for credit)

Compute $A^{-1}$ if

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ -1 & 2 & 1 \end{pmatrix}.$$ 

Compute $A \cdot \text{inv}(A)$ to see if MATLAB works(!). What do the entries -0.0000 in the matrix mean?

9.2 Determinant of a matrix

MATLAB will compute the determinant of a square matrix $A$ by the command \texttt{det(A)}.

9.2.1 Exercise (not for credit)

Let

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

1. Compute $\text{det}(A)$ and $\text{inv}(A)$ in MATLAB.

2. Compute $\text{det}(A)$ by hand and verify the determinant calculated by MATLAB. Use MATLAB to check that $\text{inv}(A)$ is the inverse of $A$ by computing $\text{inv}(A) \cdot A$ and $A \cdot \text{inv}(A)$.

3. Create two $5 \times 5$ matrices $A$ and $B$ with random entries between $-20$ and $20$ by typing

$$A = \text{round}(40 \cdot (\text{rand}(5) - 0.5))$$
$$B = \text{round}(40 \cdot (\text{rand}(5) - 0.5))$$
We will use these matrices to try to find relationships between determinants.

Now compute the following quantities using MATLAB. Record the computed values in the table provided:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>result for $5 \times 5$ random matrices</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\det(A)$</td>
<td></td>
</tr>
<tr>
<td>$\det(B)$</td>
<td></td>
</tr>
<tr>
<td>$\det(3A)$</td>
<td></td>
</tr>
<tr>
<td>$3\det(A)$</td>
<td></td>
</tr>
<tr>
<td>$3^2 \det(A)$</td>
<td></td>
</tr>
<tr>
<td>$\det(A^T)$</td>
<td></td>
</tr>
<tr>
<td>$\det(B^T)$</td>
<td></td>
</tr>
<tr>
<td>$\det(A + B)$</td>
<td></td>
</tr>
<tr>
<td>$\det(A) + \det(B)$</td>
<td></td>
</tr>
<tr>
<td>$\det(AB)$</td>
<td></td>
</tr>
<tr>
<td>$\det(BA)$</td>
<td></td>
</tr>
<tr>
<td>$\det(A^T B^T)$</td>
<td></td>
</tr>
<tr>
<td>$\det(B^T A^T)$</td>
<td></td>
</tr>
<tr>
<td>$\det(A^{-1})$</td>
<td></td>
</tr>
<tr>
<td>$1/\det(A)$</td>
<td></td>
</tr>
<tr>
<td>$\det(A^{-1}B)$</td>
<td></td>
</tr>
<tr>
<td>$\det(A)/\det(B)$</td>
<td></td>
</tr>
<tr>
<td>$\det(B)/\det(A)$</td>
<td></td>
</tr>
<tr>
<td>$\det(B^{-1}AB)$</td>
<td></td>
</tr>
<tr>
<td>$\det(B^{-1}B)$</td>
<td></td>
</tr>
<tr>
<td>$\det(2B^{-1}B)$</td>
<td></td>
</tr>
</tbody>
</table>

Look down your list and find any pairs of expressions that seem to have equal values (for example, does $\det(A) = \det(A^T)$?). Do you think that the expressions you found out to be equal will be equal for ALL matrices? Try to find theorems in the book that confirm your observations, or try to derive them from theorems in your book. Is it true that $\det(A + B) = \det(A) + \det(B)$.

### 9.3 MATLAB commands (continued)

Here are two more useful features of MATLAB.

#### 9.3.1 Continuation

It is sometimes necessary to type a command to MATLAB that would take more than one line. To do so you need to tell MATLAB to continue to the next line. If you end a line with ... MATLAB will assume that the following line is a continuation of the previous one. For example, in MATLAB, type

```plaintext
A= ... 
[ ... 
1, ... 
3,4;5 ... 
6,7]
```

This is equivalent to typing `A=[1,3,4;5,6,7]`. Obviously it was not necessary to type this command as a multi-line command.
9.4 Toeplitz matrices

MATLAB has lots of commands for creating matrices of different types. One very useful one in practice is the command for creating a Toeplitz matrix. This type of matrix is one in which each diagonal is formed by repeating a single fixed number down that diagonal (but different numbers for different diagonals). Thus the general form of a Toeplitz matrix is

$$
\begin{bmatrix}
    c_1 & r_2 & r_3 & \cdots & r_n \\
    c_2 & c_1 & r_2 & \cdots & r_{n-1} \\
    c_3 & c_2 & c_1 & \ddots & \vdots \\
    \vdots & \ddots & \ddots & \ddots & r_{n-2} \\
    c_4 & c_3 & c_2 & \cdots & r_{n-3} \\
\end{bmatrix}
$$

If \(c\) is a vector of length \(m\) and \(r\) is a vector of length \(n\) then the MATLAB command \texttt{toeplitz}(c,r) creates and \(m \times n\) Toeplitz matrix by using the entries of \(r\) and \(c\) as in the above matrix. Note that \(r_1\) is not used, but MATLAB will warn you if \(r_1 \neq c_1\). Try typing the following commands into MATLAB (or type \texttt{toep} to run a script that does the same thing):

\[
\begin{align*}
&\texttt{r=[1,2,3,4]} \\
&\texttt{c=[2,-2,-1,5]} \\
&\texttt{toeplitz(c,r)} \\
&\texttt{toeplitz(r)}
\end{align*}
\]

Note that the last command \texttt{toeplitz(r)} produces a symmetric Toeplitz matrix formed by taking \(c_2 = r_2\) etc.

9.4.1 Exercise (not for credit)

Create the following matrix using the \texttt{toeplitz} command:

\[
A = \begin{pmatrix}
-2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2
\end{pmatrix}
\]

Compute \(A^{-1}\) and \(\det(A)\).
Chapter 10

Eigenvalues

10.1 Computation of Eigenvalues

Construct a symmetric matrix $A$ by using the following MATLAB commands:

```matlab
A=round(5*rand(6))/5
A=A+A'
```

1. MATLAB is not very good at handling polynomials, but it has some functions for dealing with the characteristic polynomial.
   
   (a) Compute the coefficients of the characteristic polynomial of $A$ by typing $p=poly(A)$.
   
   (b) Find the roots of the characteristic polynomial to compute the eigenvalues of $A$ by typing $r=roots(p)$. The vector $r$ should contain the eigenvalues of $A$.

2. A better way to calculate the eigenvalues (and eigenvectors if desired) of a matrix is to use the MATLAB command `eig`. Type $e=eig(A)$. Now $e$ should contain the eigenvalues of $A$. Compare $e$ and $r$. Do they agree?

3. Now compute the following quantities
   
   (a) The determinant of $A$: $\det(A)$.
   
   (b) The product of the eigenvalues of $A$: $\text{prod}(e)$.
   
   (c) The trace of $A$: $\text{trace}(A)$.
   
   (d) The sum of the eigenvalues of $A$: $\text{sum}(e)$.

   Which of these quantities are equal?

4. Compute

```matlab
S=round(10*rand(6))
B=inv(S)*A*S
```

What is the relationship between the eigenvalues of $A$ and of $B$? Compute $eb=eig(B)$ and check your prediction.