Learning MATLAB with Linear Algebra

Soomin Jeon and Chang-Ock Lee

Department of Mathematical Sciences
KAIST

2017

Contents

Chapter 1

Matlab Basics and Programming

Matlab is a commercial program[1] which provides an integrated environment for "Matrix Laboratory". Matlab is one of many softwares used most ubiquitously in areas of mathematics and engineering. Matlab is very compatible with areas of fields which require numerical simulations because the built-in functions and m-files are based on the standard library LIN-PACK and EISPACK[2].

The filenames of Matlab files and script files always end in ".m". The program is very convenient to use because almost all of the structures are composed of matrices. In addition, the graphic processing is set so that the numerical analysis results are expressed conveniently.

One can get explanation regarding Help by typing

≫ `help`

in the Matlab command window, and one can get explanations regarding every available Matlab functions. For information regarding a specific command, type

≫ `help command_name`

For instance,

≫ `help fft`

gives explanation for `fft` command. To see the explanation in document form with Hypertext structure, use `doc` instead of `help`.

This chapter deals with basic usage of Matlab and programming.

[1] The homepage is www.mathworks.com.
[2] One can obtain the source files from www.netlib.org

1.1 Basic Command

1.1.1 Matrix

Matlab has many different types of built-in matrices. For instance, let us try to make a 7×7 matrix with random numbers in each entries.

```
≫ rand(7)
```

We can also make a random matrix with a different number of columns and rows by, for example,

```
≫ rand(2,5)
```

For more explanations regarding the function rand, use help or doc command. Another specific matrix, Hilbert matrix, is a typical example used in numerical analysis.

```
≫ hilb(5)
```

```
≫ help hilb
```

5×5 magic square can be made with the following command.

```
≫ magic(5)
```

```
≫ help magic
```

Magic square is a square matrix with the summation of the entries in the row, column, and diagonal all equal. This property of magic square will be explored in subsection 1.3 using matrix multiplication. Many types of matrices used in numerical analysis can be made using built-in functions.

```
≫ eye(6)
```

```
≫ zeros(4,7)
```

```
≫ ones(5)
```

Not only limited to these built-in matrices, one can make different matrices in any form.

```
≫ [1 2 3 5 7 9]
```

```
≫ [1, 2, 3; 4, 5, 6; 7, 8, 9]
```

```
≫ [1 2 RET 3 4 RET 5 6]
```

Here `RET` means one must press the "return key". Matlab grammatical system allows for easy usage of block matrices.

```
≫ [eye(2); zeros(2)]
```

```
≫ [eye(2); zeros(3)]
```

```
≫ [eye(2), ones(2,3)]
```

The second example above gives an error. Why is that?

1.1.2 Variable

Matlab has built-in variables, such as `pi`, `eps`, and `ans`.

```
≫ pi
≫ eps
≫ help eps
```

`who` is a command which tells us which variables are currently being used. In addition, one can see which variables are being used in the Workspace.

```
≫ who
≫ help who
```

`ans` is a variable which has the value of the last calculated results that was not assigned a variable.

```
≫ magic(6)
≫ ans
≫ x = ans
≫ x = [x, eye(6)]
≫ x
≫ who
```

Since a new variable x has been created, x is a variable being used. In order to delete a variable, use the following command.

```
≫ clear x
≫ x
≫ who
```

In order to erase all variables, use

```
≫ clear
```

or

```
≫ clear all
```

Use `help` or `doc` to figure out the difference between the two.

1.1.3 Functions

Let us try the following command.

```
≫ a = magic(4)
```

Now let us find the transpose of a

```
≫ a'
```

If `a` was a complex matrix, then Matlab would calculate the conjugate transpose, not simply a transpose.

Let us explore other arithmetic operations.

```
≫ 3*a
```

```
≫ -a
```

```
≫ a + (-a)
```

```
≫ b = max(a)
```

```
≫ max(b)
```

Some Matlab functions may have output with more than one values. If one uses `max` on matrix, then the output is maximum of each column and the indices of the row that contains that maximum. For vector case, the maximum value and the index of that value is presented.

```
≫ [m, i] = max(b)
```

```
≫ [m, i] = min(a)
```

Let us try matrix multiplication in order to verify the "magic" of magic square.

```
≫ A = magic(5)
```

```
≫ b = ones(5,1)
```

```
≫ A*b
```

```
≫ v = ones(1,5)
```

```
≫ v*A
```

In Matlab, dot in front of an operation means entry-by-entry operation. In matrix multiplication, `a.*b` is different from usual matrix multiplication in that the multiplication is done entry-by-entry.

```
≫ b = 2*ones(4)
```

```
≫ a.*b
```

```
≫ a*a
```

```
≫ a^2
≫ a.^2
```

The followings are many different arithmetic operations related to matrix.

```
≫ triu(a)
≫ tril(a)
≫ diag(a)
≫ diag(diag(a))
≫ c = rand(4,5)
≫ size(c)
≫ [m,n] = size(c)
≫ m
≫ d = .5-c
```

Typically, Matlab commands are used for scalar, but there are many functions that can be applied for both scalars and matrices.

```
≫ sin(d)
≫ exp(d)
≫ log(d)
≫ abs(d)
```

There are functions which translates decimal valued numbers into integers in Matlab. `round`, `fix`, `ceil`, and `floor` are some of these. For instance,

```
≫ f = [-.5 .1 .5]
≫ round(f)
≫ fix(f)
≫ ceil(f)
≫ floor(f)
```

1.1.4 Logic operation

Let us think of 1 as "true", and 0 as "false" in this subsection. &, |, and $\sim$ are logic operations which mean "and", "or", and "not", respectively. == is a logic operation which means equal.

```
≫ a = [1 0 1 0]
```

```
>> b = [1 1 0 0]
>> a == b
>> a <= b
>> ~a
>> a $ b
>> a $ ~a
>> a | b
>> a | ~a
```

There is a comand named `any` that checks whether the matrix has at least one non-zero entry or not. Not only that, there is also a command named `all` that checks whether all the entries in the matrix are non-zero or not.

```
>> a
>> any(a)
>> c = zeros(1,4)
>> d = ones(1,4)
>> any(c)
>> all(a)
>> all(d)
>> e = [a', b', c', d']
>> any(e)
>> all(e)
>> any(all(e))
```

1.1.5 Colon

Matlab provides a useful command for producing and dividng a matrix.

```
>> x = -2:1
>> length(x)
>> -2:.5:1
>> -2:.2:1
>> a = magic(5)
```

```
≫ a(2,3)
```

Now let us try using colon to select specific rows and columns of `a`.

```
≫ a(2,:)
≫ a(:,3)
≫ a(2:4,:)
≫ a(:,3:5)
≫ a(2:4,3:5)
≫ a(1:2:5,:)
```

In addition, one can freely use row or column vectors in a matrix.

```
≫ a(:,[1 2 5])
≫ a([2 5],[2 4 5])
```

And one can use assignment statements using vectors and matrices.

```
≫ b = rand(5)
≫ b([1 2],:)  = a([1 2],:)
≫ a(:,[1 2]) = b(:,[3 5])
≫ a(:,[1 5]) = a(:,[5 1])
≫ a = a(:,5:-1:1)
```

All of these are simple Matlab functions and examples of matrix multiplication. More functions can be found in the appendix **??**.

1.1.6 Other features

The default setting for Matlab is that the decimals are expressed up to 4 decimal digits. Even though the actual calculation is done up to 16 decimal digits, it is rounded and then expressed. The command

```
≫ format long
```

changes so that all 16 decimal digits are displayed. And,

```
≫ format short
```

changes back to the default setting. Of course, one can display scientific constants long or short, however one wishes, by using the following command

```
≫ format short e
≫ format long e
```

It is not necessary to always display all the calculated values on the screen. If one attaches semicolon(;) at the end of the command, then calculation will proceed, but the values will not be displayed, just like the colon in Maple.

Sometimes, a lot of time is spent on making matrices in Matlab session, and one might need to use these matrices next time. One can use Matlab command

≫ `save filename`

to make a file named `filename.mat` that saves all the variable values made in the current session. If one does not wish to save all the variable values, then one can use

≫ `save filename x y z` to save only variable `x,y,z` in `filename.mat`. These saved variables can be reused next time using the command

≫ `load filename`

There are times when one must record all the keyboard inputs and results in Matlab session. The following command allows one to record all the process except for graphs.

≫ `diary filename`

produces a file named `filename` and starts the record.

≫ `diary off`

stops the record,

≫ `diary on`

restarts the record. These processes are saved in text files, so one can edit them.

1.2 Programming in Matlab

Matlab is a language, with which one is able to program, just like Maple. A person who wants to write the files can easily do so by using .m files toy write a program and execute it. If one wrote `myfile.m`, one can execute `myfile.m` by using the command `myfile`, just like other Matlab commands. Matlab is an interpreter language, so one can execute a written program without compiling.

1.2.1 Assignment Statement

Assignment means one can assign a value to a variable. In other words, `x=a` means that one is assigning the value `a` to the variable `x`. Let us see the following simple program which uses assignment statement.

```
function r=mymod(a,d)

% r=mymod(a,d).  If a and d are integers, then
% r is the integer remainder of a after
% division by d.  If a and d are integer matrices,
% then r is the matrix of remainders after division
% by corresponding entries.  Compare with REM.

r=a-d.*floor(a./d);
```

Make the above `mymod.m` file, and assign an integer value to `a` and `d`. Then, if one uses

≫ `mymod(a,d)`

it is executed as if it were a built-in Matlab command. Let us enter the following command.

≫ `help mymod`

Then the 5-lined information above that start with % will show up. % means that the statements that come after it are ignored when a program is executed. When help command is executed in Matlab, the information on the uppermost part of the announced part of the function is displayed. Using this method, the help command provides a help function that let us know the properties of the function quickly. Let us enter the following.

≫ `type mymod`

This command displays the entire details of the file on the screen for convenient reading. Now let us examine the details of `mymod.m`. The first row corresponds to "function announcement statement". In here, name of the file(always the same as the file name without m), input variable(In this case, `a` and `d`), and output variable(`r`) are announced. The next is the 'help" aforementioned. Last, the middle part of the program is displayed on the screen. The variable `r` is assigned the value `a-d.*floor(a./d)`. Out of the operations on the right hand side, "." means entry-by-entry operation. Last, ";" means that until the last part of the execution, the result is blocked from being printed on the screen. Try executing the program after deleting ";". A quite different result will be printed.

1.2.2 Conditional Statement

Conditional statement has the following structure.

```
if <condition>, <program> end
```

<condition> above is a MATLAB function, but it is not a must to have values only 0 and 1. In the conditional statement, <program> is executed only when <condition> has a non-zero value, and proceeds to the next. Let us not forget that a==b and a<=b are functions perceived as having values 0 or 1. Often, conditional statements have the following form.

```
if <condition1>, <program1> else <program2> end
```

In this case, if <condition1> has the value 0, then <program2> is performed. There is another form

```
if <condition1>, <program1>
elseif <condition2>, <program2>
end
```

In this case, if <condition1> is non-zero, then <program1> is performed, if <condition1> is 0 and <condition2> is non-zero, then <program2> is performed. In other cases, the program exits the conditional statement and proceeds to the next. The following is a simple program that uses conditional statement.

```
function b=even(n)

% b=even(n).  If n is an even integer, then b=1
% otherwise b=0

if mymod(n,2) == 0
  b=1;
  else b=0;
end
```

1.2.3 For Loop

For loop has the following structure.

```
for i=1:n, <program>, end
```

Depending on the value of i, <program> is repeatedly executed each time. Let us introduce some simple program. The first is matrix multiplication.

```
function c=add(a,b)

% c=add(a,b).  This is the function which adds
% the matrices a and b.  It duplicates the MATLAB
% function a+b.

[m,n] = size(a);
[k,l] = size(b);
if m~=k | n~=l,
  r='ERROR using add:  matrices are not the same size',
  return,
end
c=zeros(m,n);
for i=1:m,
  for j=1:n,
    c(i,j)=a(i,j)+b(i,j);
  end
end
```

The next is a program related to matrix multiplication.

```
  function c=mult(a,b)

  % c=mult(a,b).  This is the matrix product of
  % the matrices a and b.  It duplicates the MATLAB
  % function c=a*b.

  [m,n]=size(a);
  [k,l]=size(b);
  if n~=k
    r='ERROR using mult:  matrices are not compatible
      for multiplication',
    return,
  end
  c=zeros(m,l);
  for i=1:m,
    for j=1:l,
      for p=1:n,
        c(i,j)=c(i,j)+a(i,p)*b(p,j);
      end
    end
  end
```

Let us look carefully at the conditional statement after the `size` statement in both programs. It is there to print error message. In `add` case, adding matrices with different dimensions results in printing error message, and in `mult` case, if the dimension of the column of the left matrix and the dimension of the row of the right column do not match, error message prints. If there is an error even when there is no such error messages, then MATLAB will print a strange calculation result. Observe the single quotation marks in the error message part. The sentence indicated by the quotation marks is regarded as text and is displayed as a value of the variable `r`. After the error message is the return command. This is an instruction statement which tells us to return back to the function or prompt that called `add` or `mult`. Return command is very useful in error message statement.

`i` in the next loop statement

```
for i=1:n, <program>, end
```

can be handled in many different ways in the program. There is no problem in writing a vector in place of `1:n` in MATLAB. In the case of loop statement

```
for i=[2,4,5,6,10], <program>, end
```

the program will be executed 5 times repeatedly, with i having the values $2, 4, 5, 6, 10$ each time. The developers of MATLAB went further. It's possible to use vector, but what about matrix? Thus the loop statement like the following

```
for i=magic(7), <program>, end
```

is also possible. This program will be executed 7 times (the dimension of the column), with the variable `i` being the column of `magic(7)` each time.

1.2.4 While Loop Statement

While loop statement takes the following form

```
while <condition>, <program>, end
```

`<condition>` becomes MATLAB function, just like it did in conditional statement. The program keeps executing as long as `<condition>` has non-zero value. However, there is a risk in using while statement because there is no way to forcefully terminate the while statement. Next is a simple program using a while statement.

```
function l=twolog(n)

% l=twolog(n).  l is the floor of the base 2
% logarithm of n.

l=0;
m=2;
while m<=n
  l=l+1;
  m=2*m;
end
```

1.2.5 Recursion

Recursion means that the function calls upon itself. Next is a simple example that uses recursion.

```
function y=twoexp(n)

% y=twoexp(n).  This is a recursive program for
%    computing
% y=2^n.  The program halts only if n is a nonnegative
%    integer.

if n==0, y=1;
  else y=2*twoexp(n-1);
end
```

Many recursive programs consist of conditional statement just like this program. The condition `n==0` is the basic part of recursion, and it is the only way for the program to restrict itself from calling itself up. The "else" part is the part that shows the recursion. Let us take note on how `twoexp(n-1)` is executed in a program that calculates `twoexp(n)`. The principle is calling upon smaller number `n-1`, and continuing this until n=0 is called upon. Successful recursion means continuously calling upon smaller number.

However, there are many dangers in using recursion. First, just like while loop statement, the function can continuously call itself up. Second, although it can be stopped, it might calculate unneccesarily, wasting time, and third, while the recursive program is executing, extra memory is required. In massive calculation, the memory storage is crucial, and it must not be unnecessarily wasted. Then, with all these negative sides, why is recursion used? Actually,

users who are familiar with using recursive programs can utilize its merits while avoiding these problems. When using recurisve function, one can program easier than when one does not use recursive function.

1.2.6 Other Variety of Programs

One can use matrix-valued functions as conditional statement or conditions of loop statement. In other words, conditions can have matrices such as `ones(2)`, `zeros(2)`, or `eye(2)` in them. If `<condition> = eye(2)` in the following program, how will this program execute?

```
if <condition>, <program1>,
else <program2>, end
```

Here, `<program1>` is executed when all the entries in `<condition>` is non-zero. Hence, `<program1>` is executed if `<condition>` is `magic(2)`, and `<program2>` is executed if `<condition>` is `eye(2)`.

Now let us predict how the following program will be executed.

```
if A ~= B, <program>, end
```

Here, `<program>` will only be executed when the entries of `A` and `B` are all different. If we wanted to execute `<program>` even when only one entry of `A` and `B` are different, then how can we do this? There are many ways, but first, we can do

```
if A == B, else <program>, end
```

With this program, the "else" part will be executed when at least one entry of `A` and `B` is different. For a different way, we can use `A==B` as `all(all(A==B))` to transform it into binary function. The inner `all` will produce a binary vector with all the entries as 1 if the `i`-th column of `A` and the `i`-th column of `B` are the same. If all the entries of this vector is 1, then the outer `all` will produce 1. Therefore, if at least one entry of `A` and `B` is different, then `all(all(A==B))=0`. Therefore, the following program

```
if ~all(all(A==B)), <program>, end
```

will be executed as we want.

Very similar method is also used in while loop statement.

```
while <condition>, <program>, end
```

With this program, when `<condition>` takes a matrix value, the program is continuously executed if all the entries are non-zero, and the program exits the loop statement if at least one of the entry is 0.

The following program uses many conditional statements at the same time. Let us predict how this program is executed.

```
if <condition1> & <codition2>,
<program>, end
```

Of course, the program is executed when both `<condition1>` and `<condition2>` are non-zero. But, does this program always work without any problems? In reality, there are times when `<condition1> = 0` and `<condition2>` causes an error during the execution of a program. For instance, this happens in the following program.

```
if i <= m & A(i,j)==0, <program>, end
```

Here, `m` is the dimension of the column of `A`. If `i>m`, then we would like the program to proceed on. However, the error message will be displayed on the screen because of `A(i,j)` part. To avoid this, one can change the conditional statement as follows.

```
if i<=m,
  if A(i,j)==0,
    <program>
  end
end
```

1.2.7 Script

Script is m-file without function declaration statement and executes differently from m-file defined by function. Let us assume `x` is one of the variable in use this session. If we write a program using a function(-defined) file, and we use `x` in the program even if `x` is not an input variable of the function, then the program does not use the value of `x` that was defined during the session, but it takes on the value that was locally allocated in the program. Furthermore, the program does not change the value of `x` in the session. Therefore, during the execution of function file, it is very convenient in that one does not need to take heed of the variable declared in the session, and this is because of the function that was declared. If there is no part about the declaration, then that script is regarded as a continuous part of the session. Hence, if a used variable is changed in the script during the session, then the variable is changed for the entire session.

1.2.8 Proposal for Better Programming

Pay special attention to the following details when programming with MATLAB. Of course, these details can be applied with not only Matlab, but also for programming in Maple as well.

1. Just like the examples until now, it is recommended to use indent while programming. This way, it is easier to read, easier to find grammatical errors, and easier to think of programming in group.

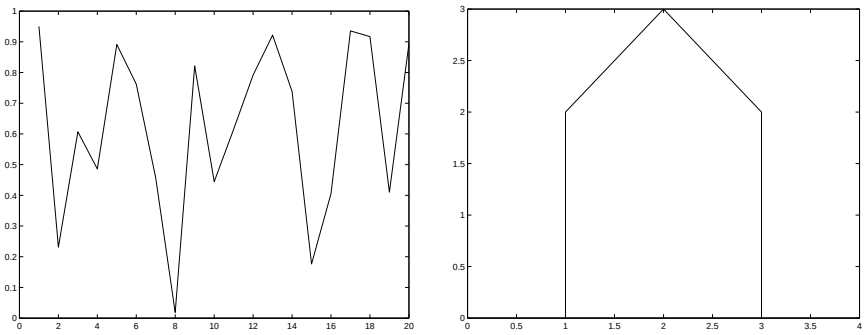
2. Be precise with writing footnote. This is because there can be times when one cannot understand the program that he or she has written. If one uses % in a sentence, then everything after % in that line becomes footnote.

3. Insert the error message just like the examples above. These error messages are very helpful when debugging.

4. Always structuralize the output as the same as the form of input of other functions. For example, if the form of output of a program is "yes-no", then change it to 1 or 0 instead of "yes" and "no". This way, they can be used as conditions for conditional statement or loop statement.

5. Try to avoid loop statement in MATLAB as much as possible and use optimized built-in MATLAB functions. For example, try to see how much faster `A*B` is compared to `mult(A,B)`. One will probably be surprised by the difference in speed.

6. If there is a difficult in writing a program, then try taking out the suspicious part and execute it separately. Then come back to the original program and then fix it.



(a) (b)

Figure 1.1: `plot`

1.3 Graphics

A well drawn picture is better than 100-words long explanation. Matlab has powerful graphics processing system that is convenient to use and that expresses given data very well. (The pictures used in this chapter are all drawn using Matlab.) This section will explain the high-level 2-D and 3-D graphic fuction of Matlab. The low-level factors like graphic objects are not dealt with here. This section deals with only showing a little bit of the power of Matlab graphics. For more details, use `help` or `doc`, or reference **Matlab User's Guide** or **Reference Guide**.

1.3.1 2-D Graph

2-D graph is drawn using the `plot` function. In the simplest form, one can display graphs that has only one vector value like `plot(y)`. In this case, the elements of `y` are drawn in the order of index. In other words, `plot(rand(1,20))` graphs 20 random numbers in the order of $1 \sim 20$, connects all the consecutive points in lines, and prints the picture on the screen just like picture 1.1(a). If `y` was a matrix, each elements of columns are drawn in order, drawing as many as the number of columns of the matrix. The axis are adjusted with the maximum and minimum values of the data and printed on the screen.

The typical form of `plot` is `plot(x,y)`. Here, `x` and `y` are vectors with same dimensions. Let us execute the following command.

```
≫ x = 0:pi/40:4*pi;
≫ plot(x, sin(x))
```

In this graph, the coordinate of `i`-th point is $(\mathtt{x_i}, \mathtt{y_i})$.

A line is drawing using the coordinates of `x` and `y` vectors. For instance, to

draw a line connecting $(0, 1)$ and $(4, 3)$, type

```
≫ plot([0,4], [1,3])
```

Here, `[0,4]` mean `x` coordinates of two points, and `[1,3]` represent the associated `y` coordinates.

Practice question: Draw a line connecting the following each points: $(0, 1), (4, 3), (2, 0), (5, -2)$

Practice question: Draw a house similar to the picture 1.1(b).

Label

One can attach label to the graph using the following commands.

`gtext('text')`

After executing this command, when one moves the mouse on the graph window, a cross-shaped thing is shown on the graph and waits for mouse click. After choosing the location by moving the mouse, the sentence `'text'` pops up on the graph window upon clicking.

`text(x, y, 'text')`

Prints `'text'` on `(x,y)` location on the graph window. If `x` and `y` are vectors, `'text'` is printed on each points.

`title('text')`

Prints the title `'text` at the very top part of the graph window.

`xlabel('text')`

Puts an explanation on the x-axis.

`ylabel('text')`

Puts an explanation on the y-axis.

Drawing Many Graphs on the Same Axis

There are at least 3 ways to draw many graphs on the same axis. However, if the recent graph contains data with bigger range than the previous data, then the graph can be rescaled.

1. The easiest way is to the command `hold`, which holds the current graph on the graphic window. The graphs that are drawn after that are overlapped until the `hold` status is turned off. The command that turns off `hold` is `hold off`.

2. The second way is to use the `plot` command the following way.

   ```
   plot(x1, y1, x2, y2, x3, y3, ...)
   ```

The merit of this way is that each vector pair is represented with different color/line forms.

3. The third way is using `plot(x,y)`. Here, `x` and `y` are either both matrices or one is vector and the other is matrix. If one of them is matrix and the other vector, then the column or row that matches the vector is printed with different color/line form. The column or row is chosen to match the vector. In case of square matrix, the column is selected. If both `x` and `y` are matrices that have same dimensions, the column of `x` is matched with the column of `y` and printed. If `x` is not predetermined (just like `plot(y)`), then the column of `y` is matched with the index of the row and then drawn.

Form of Line, Output Symbol, and Color

The form of line and output symbol can be selected by adding sentences to the `plot` command.

`plot(x,y, '--')`

`y` is plotted in point-line with respect to `x`.

`plot(x,y, 'o')`

Instead of connecting each points with line, the points are displayed in circle.

`plot(x,y, '--o')`

Each data is connected with point-line and the points are displayed in circle.

`plot(x,y, '--m')`

Displays magenta colored point-line.

For diverse usage, refer to `help plot`.

Axis Scale

In Matlab, the axis are automatically adjusted when drawing the graph. One can use command

```
axis([xmin, xmax, ymin, ymax])
```

to adjust the axis. One can return to the default setting using `axis('auto')`. The next command

```
v = axis
```

adjusts the current axis scale to the vector `v`. One can maintain the current axis scale using `axis(axis)`. Then, executing `hold`, the next graph is printed with the current scale.

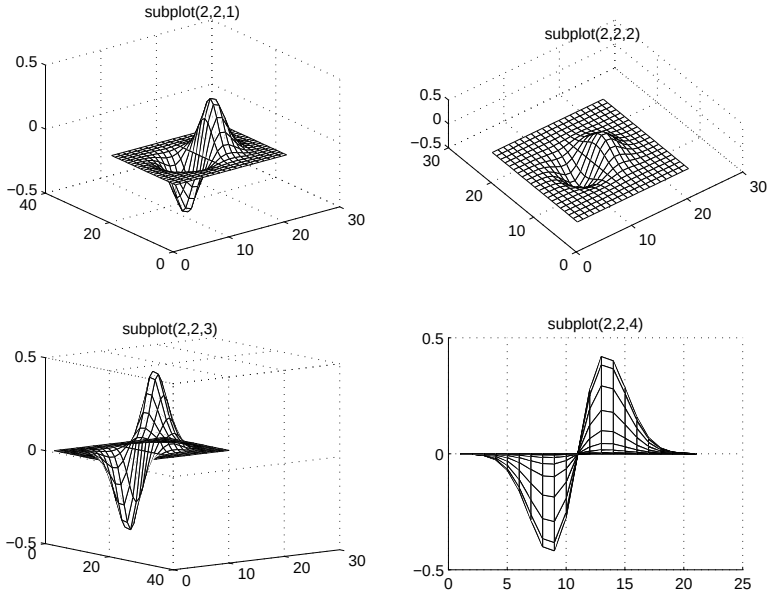


Figure 1.2: 4 subplot: 3

Let us draw the following circle.

≫ `x = 0:pi/40:2*pi;`

≫ `plot(cos(x), sin(x))`

However, the thing that is shown in the screen does not look like circle. Why? The default setting for the length of each axis in Matlab is different. The command `axis('equal')` equalizes the visual scale of `x` and `y` on the screen so that the circle shape will be printed. Use `axis('normal')` to turn off this setting. Use the command `axis('on')` and `axis('off')` to set/erase the axis marking.

Drawing Many Axes Using Subplot

One can draw graphs with many different axes in one graphic window using `subplot`. The command `subplot(m, n, p)` divides the current graphic window into $m \times n$ small axis, (starting from top left following the row) sets the `p`-th graph as the current graph. For example, the following program makes four axes just like the picture 1.2.

```
% test_subplot.m

[x,y] = meshgrid(-3:0.3:3, -3:0.3,3);
z = x.* exp(-x.^2-y.^2);
subplot(2,2,1)
mesh(z), title('subplot(2,2,1)')
subplot(2,2,2)
mesh(z)
view(-37.5, 70), title('subplot(2,2,2)')
subplot(2,2,3)
mesh(z)
view(-37.5, -10), title('subplot(2,2,3)')
subplot(2,2,4)
mesh(z)
view(0,0), title('subplot(2,2,4)')
```

The command `subplot(1,1,1)` sets the graphic axis back to one.

figure, clf, cla

Command `figure` produces a new figure window.

`figure(N)`

Produces `N`-th figure window. Commands related to graphic after this will be executed in this window.

`clf`

Everything except the window of the current figure window will be deleted. Thus, the properties of the current window will also be deleted.

`cla`

Deletets all the lines, symbols, texts except the axis and axis markings in the current figure window.

Inputs related to graphics

The command

```
[x, y] = ginput
```

saves all the points that are inputted by the mouse on the current window. Cross shape is shown on the screen, and saves the points the mouse clicks. `Enter` finishes this command. The command

```
[x, y] = ginput(n)
```

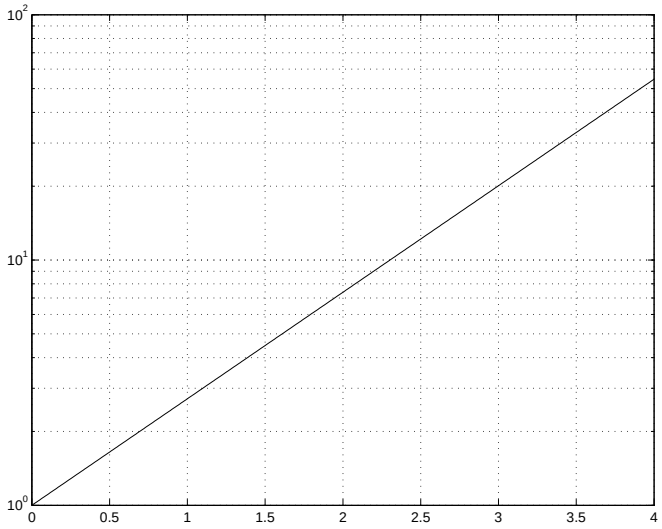


Figure 1.3: Logarithmic plot

is exactly the same as `ginput` except it only saves `n` points. Use `help` or `doc` to earn more information..

Logarithmic plot

Command `semilogy(x, y)` displays the graph `y` with $\log_{10}$ scale and `x` with linear scale. For instance,

```
≫ x = 0:0.01:4;
≫ semilogy(x, exp(x)), grid
```

draws the graph like picture 1.3. The increase of equidistant interval of `y`-axis is expressed in exponent of 10. In addition, the marking in the `y`-axis are drawn to show 1, 2, 3, ..., 10, 20, 30, ..., 100, ... starting from bottom. There are also similar commands like `semilogx` and `loglog`. `x` and `y` can be vector or matrix just like they were with `plot`.

Practice question: Draw graph of $x^2, x^3, x^4, \exp x^2$ with $0 \leq x \leq 10$ using `semilogy`.

Polar Coordinate

The command `polar(theta, r)` uses angle θ and size r to show the position of the point. For instance,

```
≫ x = 0:pi/40:2*pi;
```

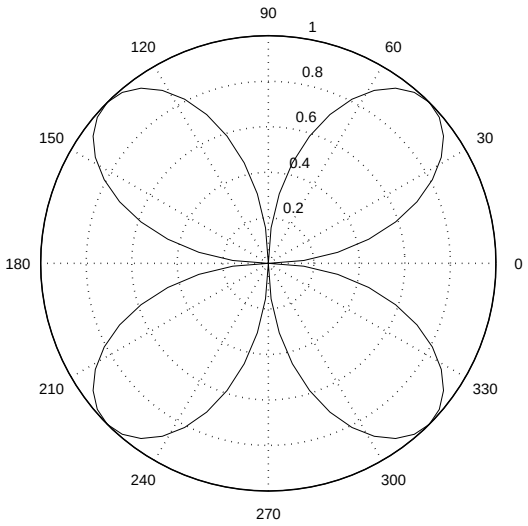


Figure 1.4: Polar plot: $r = \sin 2\theta$

```
≫ polar(x, sin(2*x)), grid
```

produces graph like picture 1.4.

Drawing a Graph of a Function that Changes Quickly

Until now, the graphs were drawn with data that has x-axis all distributed equally, like the example `x = 0:0.01:4`. If a function to be drawn rapidly changes in a certain domain, then the distribution of x-axis will be inefficient, and the graph will not be drawn properly. For instance,

```
≫ x = 0.01:0.001:0.1;
```

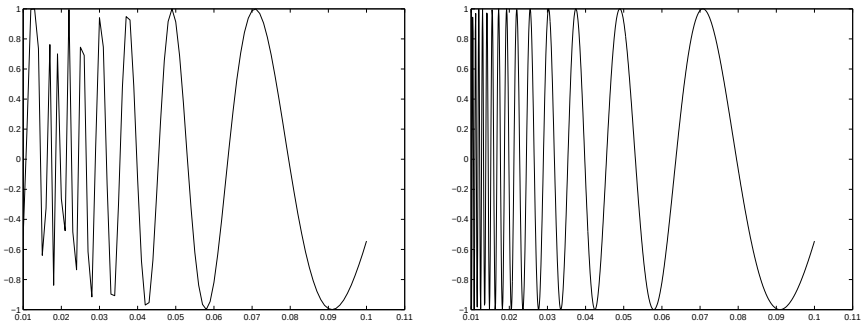
```
≫ plot(x, sin(1./x))
```

will draw the graph like picture 1.5(a). However, if the increment of x is reduced to 0.0001, then the graph like picture 1.5(b) will be drawn. The two graphs are clearly different in the domain $x < 0.04$.

Matlab provides `fplot`, which is a more efficient function. When it comes to drawing a function like $\sin(1/x)$, `fplot` calculates rapid changing places more frequently. However, the command `fplot` has a demerit, which is it must use function file.

Many Commands related to 2 Dimensional Graphs

Matlab provides many commands that express functions into graphs. Here, we state some examples, but we wish for the reader to use `help` or `doc` to get more



(a) (b)

Figure 1.5: $y = \sin(1/x)$

detailed information.

`bar`

Draws bar graph.

`compass`

Displays the vector with entries size and direction of complex number with an arrow starting from the origin.

`errorbar`

Displays error bar.

`hist`

Draws histogram.

`quiver`

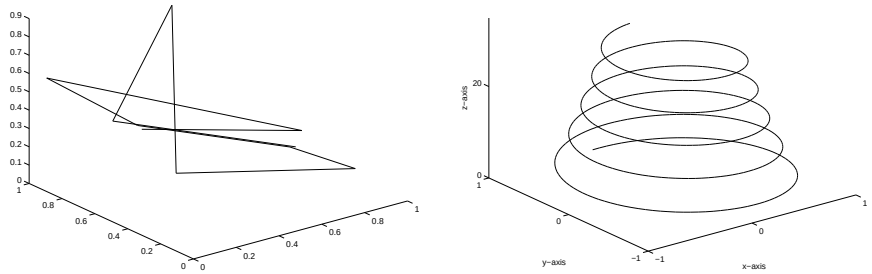
Draws many different types of vector fields(for instance, gradient) using little arrows.

`fill`

Draws polygon and fills in with given color.

1.3.2 3 Dimensional Graph

Matlab provides many functions that can express 3 dimensional graphs. This susbsection will be a brief introduction to these functions.



(a) (b)

Figure 1.6: example of command plot3

Plot3

The command `plot3` is a 3 dimensional version of `plot`, and if we write

```
≫ plot3(x, y, z)
```

then a line connecting the points (x_i, y_i, z_i) will be drawn in 3 dimension. For instance,

```
≫ plot3(rand(1,10), rand(1,10), rand(1,10))
```

uses 10 random points to draw a line in 3 dimension, like picture 1.6(a). And another example,

```
≫ t = 0:pi/50:10*pi;

≫ plot3(exp(-0.02*t).*sin(t), exp(-0.02*t).*cos(t), t), ...
xlabel('x-axis'), ylabel('y-axis'), zlabel('z-axis')
```

draws a dwindling spiral like picture 1.6(b). Be careful on the direction of x-axis, y-axis, z-axis, and pay attention to the fact that label was marked for each axis.

Mesh Surface

The following is an example regarding mesh surface.

```
% Mexicanhat.m

[x y] = meshgrid(-7.5:0.5:7.5, -7.5:0.5:7.5);
r = sqrt(x.^2 + y.^2) + eps;
z = sin(r)./r;
mesh(z);
```

To know how these surfaces are drawn, let us study a simple example like $z = x^2 - y^2$. We want a graph that shows the change in z value when there is a change in values in x-y plane. Let us think only in the domain $0 \leq x \leq 5, 0 \leq y \leq 5$ for this example. First use Matlab command `meshgrid` to produce grid on the x-y plane where the surface will be drawn.

```
≫ [x y] = meshgrid(0:5, 0:5)
```

This command produces two matrices `x, y` like the following.

`x` =

0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5
0	1	2	3	4	5

`y` =

0	0	0	0	0	0
1	1	1	1	1	1
2	2	2	2	2	2
3	3	3	3	3	3
4	4	4	4	4	4
5	5	5	5	5	5

As we can see from above, matrix `x` represents each grid of x-axis, and matrix `y` represents each grid of y-axis. If the grid of x-direction and that of y-direction are of same shape, then we can write in the following short form.

```
≫ [x y] = meshgrid(0:5)
```

And as can be predicted with the Matlab matrix operation, the command `z = x.^2-y.^2` produces the following matrix.

`z` =

0	1	4	9	16	25
−1	0	3	8	15	24
−4	−3	0	5	12	21
−9	−8	−5	0	7	16
−16	−15	−12	−7	0	9
−25	−24	−21	−16	−9	0

For instance, at the point $(5, 2)$, z takes the value $5^2 - 2^2 = 21$. Fortunately, one does not need to be concerned with the precise relationship between the coordinate system of the grid and the index of the matrix. This is automatically adjusted by `meshgrid`.

The command `mesh(z)` produces graph with lattice-like surface, where the points on the grid are raised to the surface and then connected to form the lattice. In other words, `mesh` draws a 'wire mesh'-like surface. If one does not want color, then one can type

```
≫ mesh(z,'EdgeColor','black')
```

In addition, another command `surf` draws a lattice-like surface composed of small colored tiles. Use `help` or `doc` to learn more about `mesh` and `surf`.

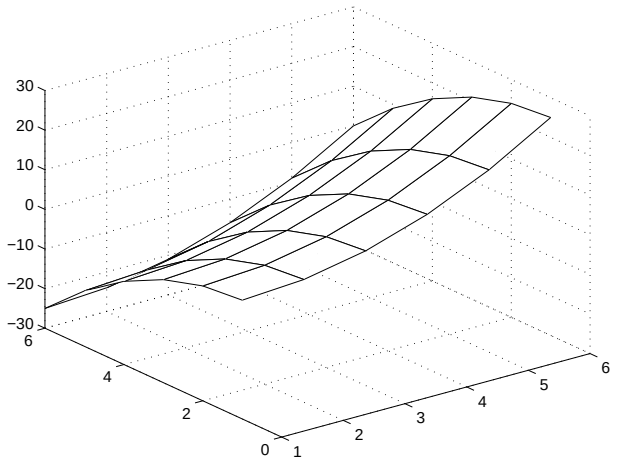


Figure 1.7: curved surface $z = x^2 - y^2$

If one is using Matlab student edition, then one must know that there is a limit to grid size when using `meshgrid`. The limit is that the size of the row or column of matrix must be at most 32, and the size of matrix must not exceed 8192.

Practice question: Use the command

```
[x y] = meshgrid(0:0.25:5);
```

to draw a denser mesh than picture 1.7.

Practice question: The distribution of temperature on the iron plate is as follows.

$$u(x, y) = 80y^2 e^{-x^2 - 0.3y^2}$$

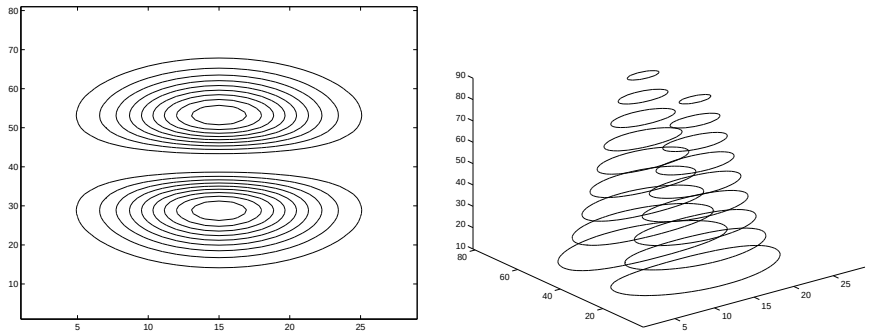
With the domain $-2.1 \leq x \leq 2.1, -6 \leq y \leq 6$, draw the curved surface u with the grid size of each direction as 0.15.

Drawing Contour

After solving the practice questions above, execute the following command.

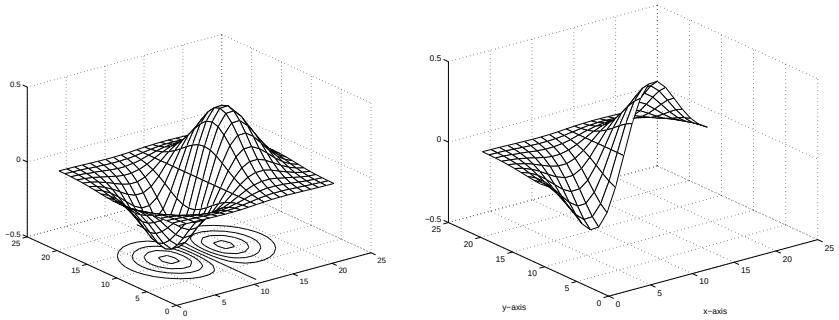
```
≫ contour(u)
```

Then, one can earn a contour(isothermal line) about the distribution of temperature like picture 1.8(a). The command `contour` can take second input variable. For this second variable, one inputs how many lines the contour will draw or a vector with specific values for drawing contour. Use command `contour3` to



(a) (b)

Figure 1.8: Contour plot



(a) (b)

Figure 1.9: (a) meshc (b) Erasing part of curved surface

draw a 3 dimensional contour like picture 1.8(b). One can make contour label with the command `clabel`.

To display both contour and mesh together, one can use `meshc` or `surfc`. For instance, the following program

```
≫ [x y] = meshgrid(-2:0.2:2);
≫  z = x.*exp(-x.^2 - y.^2);
≫ meshc(z);
```

draws a graph like picture 1.9(a).

Deletion of Curved Surface Due to NaN(Not a Number)

If the matrix that holds information on the curved surface contains NaN, then this value does not appear in the graph, and because of this, a part of the curved

surface will be omitted. Let us study the following example.

```
% cropping.m

[x y] = meshgrid(-2:.2:2);
z = x.*exp(-x.^2 - y.^2);
c = z;    % preserve the original surface
c(1:11, 1:21) = nan;
mesh(c), xlabel('x-axis'), ylabel('y-axis')
```

The above program will display the graph like picture 1.9(b).

quiver

The command `quiver` draws a vector that starts at 2 dimensional point. Although it is drawn in 2 dimensional graph, it is occasionally used with `contour`, which helps understanding changes in 3 dimensional curved surfaces. For instance, let us think about $V = x^2 + y$, which is a scalar function with 2 variables for input. The gradient of V is defined as the following vector field.

$$\begin{aligned} \nabla V &= \left(\frac{\partial V}{\partial x}, \frac{\partial V}{\partial y} \right) \\ &= (2x, 1) \end{aligned}$$

The following program draws the direction of ∇V for each points in x-y plane(refer to picture 1.10).

```
% test_quiver.m

[x y] = meshgrid(-2:.2:2);
V = x.^2 + y;
dx = 2*x;
dy = ones(size(y));
axis equal
contour(x, y, V), hold on
quiver(x, y, dx, dy), hold off
```

'Contour' is a series of `level surface`. Gradient of a random point is perpendicular to the level surface that passes through that point. When drawing a contour, the vectors `x` and `y` are required for labeling the axes. What will happen if take this out and just use `contour(V)` and execute the above `test_quiver.m`? Let us try to predict the result before we execute it.

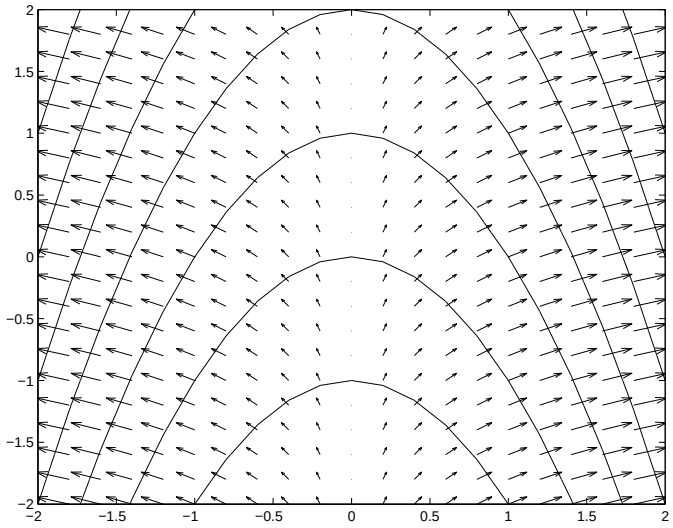


Figure 1.10: Gradient and contour

Another option regarding `quiver` is that one can change the size of the arrows. See `help` or `doc`.

If it is not possible to differentiate the vector V or if one does not wish to differentiate it, then one can use the command `gradient` to calculate the derivative.

```
≫ [dx dy] = gradient(V, 0.2, 0.2);
```

0.2 means the increment with respect to x and y directions for approximate calculation.

Pseudocolor

The following program

```
≫ [x, y] = meshgrid(-2:.2:2);

≫ z = x.*exp(-x.^2 - y.^2);

≫ pcolor(z), shading flat, colormap(hot)
```

draws a contour that expresses height using mixture of red, orange, and yellow. The command `shading flat` eliminates the grid line. `pcolor` means pseudo-color. Each element of the matrix `z` is used as index of color map(in this case hot) that determines the color which will express the element. If one wants a cool color, then try `colormap(cool)`.^^ Does it feel cool? There is yet another color map, `colormap(hsv)`, where `hsv` stands for huge-saturation-value.

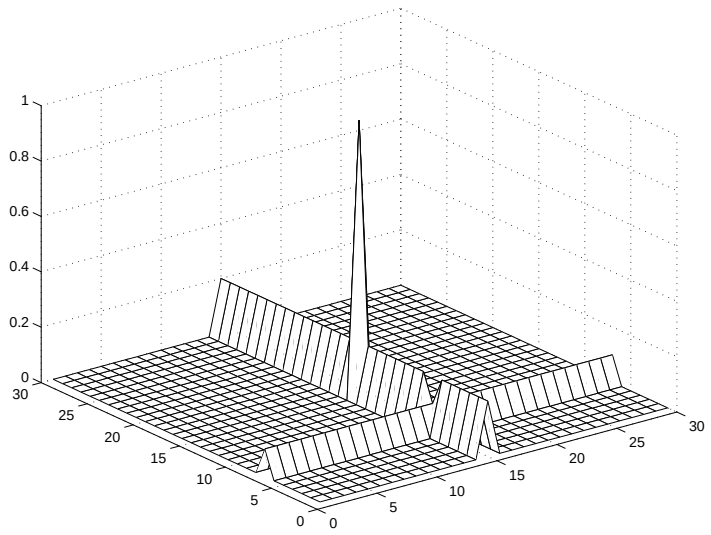


Figure 1.11: visualization of matrix

Visualization of matrix

The command `mesh` can 'visualize' the matrix. The following program displays the graph like picture 1.11.

```
% visual_mat.m

a = zeros(30);
a(:,15) = 0.2*ones(30,1);
a(7,:)  = 0.1*ones(1,30);
a(15,15) = 1;
mesh(a)
```

The size of the matrix `a` is 30×30. The middle element `a(15,15)` is 1, and every element of the 7-th row is 0.1, and the remaining elements of the 15-th row is 0.2. `mesh(a)` cognizes every rows and columns of matrix a as coordinates of x-y. In other words, the value of `a(i,j)` is the height of the curved surface mesh at point `(i,j)`.

Rotating 3 Dimensional Graph

`view` is a command which designates observation point when viewing a 3 dimensional graph. To see how this works, let us execute the following program which rotates the visualized matrix picture 1.11.

```
% rotation.m

a = zeros(30);
a(:,15) = 0.2*ones(30,1);
a(7,:)  = 0.1*ones(1,30);
a(15,15) = 1;
el = 30;
for az = -37.5:15:-37.5+360
   mesh(a), view(az, el)
   pause(0.5)
end
```

The command `view` requires two angles. The first, as can be seen in the example, is azimuth `az` on the x-y plane that has degree as its unit. `az` rotates the observation point about z-axis - in other words, the 'sharp point' at $(15, 15)$ in picture 1.11 - counterclockwise. The default value for `az` is $-37.5°$. Therefore, the above program rotates the observation point about z-axis $15°$ each time from the default value. The second angle of `view` is `el` which expresses altitude with degree as its unit. This means the angle between the z-axis and x-y plane. For instance, $90°$ represents 2 dimensional graphic, in other words, looking down from above. If the value of altitude is positive, then the observer is above the x-y plane, and if negative, then is below the plane. The default value is $30°$.

The command `pause(n)` stops the execution for `n` seconds.

Practice question: Fix the above program so that the value of `az` is fixed as the default value and and the value of `el` is gradually changing.

Lighting

One can materialize lighting and shadow effect using the command `surfl`. Try the following.

```
≫ [x, y] = meshgrid(-2:0.05:2);

≫ z = x.*exp(-x.^2 - y.^2);

≫ surfl(z, [-20 50]), colormap(gray), shading flat
```

The location of the source of light is determined by the second variable of `surfl`, with the first value being azimuth and the second being altitude. To make a natural reflection light, one must set the grid compact so that the grid is not visualizable. (In this case, 81×81)

Chapter 2

Systems of Linear Equations

2.1 Introduction to Systems of Linear Equatios

No MATLAB problems in this section.

2.2 Solving Linear Ssytems by Row Reduction

Exercise 2.1. (*Reduced Row Echelon Form with Pivot Columns and Ranks*) In MATLAB, there are several useful commands for matrices such as *rref* command which produces the reduced row echelon form together with the pivot columns, and *rank* command which gives the number of the leading 1's without finding its row echelon form. Find the reduced row echelon form, the pivot columns, and the rank of the matrix A, where

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

Solution.

```
% Construct the matrix A.
A=[2 -3 1 0 4; 1 1 2 2 0; 3 0 -1 4 5; 1 6 5 6 -4];

% Display the format of each entry as a rational form
format rat;

% Find the reduced row echelon form
% and the pivot columns of the matrix A.
[rref_A pivotcols] = rref(A);
```

```
% Find the rank of the matrix A.
rank_A = rank(A);

disp('The reduced row echelon form is'); disp(rref_A);
disp('The pivot columns are'); disp(pivotcols);
disp('The number of the leading 1 is'); disp(rank_A);
```

MATLAB results.

```
The reduced row echelon form is
       1    0    0   17/13   3/2
       0    1    0   11/13  -1/2
       0    0    1   -1/13  -1/2
       0    0    0      0     0

The pivot columns are
       1    2    3

The number of leading 1 is
       3
```

Exercise 2.2. (*Linear Combinations*) Use the MATLAB command *rref* to express the vector $\mathbf{b} = (-21, -60, -3, 108, 84)$ as a linear combination of $\mathbf{v_1}$, $\mathbf{v_2}$, and $\mathbf{v_3}$ where $\mathbf{v_1} = (1, -1, 3, 11, 20)$, $\mathbf{v_2} = (10, 5, 15, 20, 11)$, and $\mathbf{v_3} = (3, 3, 4, 4, 9)$.

Solution.

```
% Construct b as a column vector.
b = [-21 -60 -3 108 84]';
% Set v1, v2, v3 as column vectors.
v1 = [1 -1 3 11 20]';
v2 = [10 5 15 20 11]';
v3 = [3 3 4 4 9]';
% Set a matrix A with column vectors v1, v2 and v3.
A = [v1 v2 v3];
% Augmented matrix [A | b].
augA = [A b];
% Reduced row echelon form of augA.
rref_augA = rref(augA);
% Solution vector from rref_augA.
x = rref_augA(1:3, 4);

% Display the result as an integer form.
format rat;
disp('b is a linear combination of x(1)*v1+x(2)*v2+x(3)*v3, where');
disp('x(1) ='); disp(x(1)); disp('x(2) ='); disp(x(2));
disp('x(3) ='); disp(x(3));
```

MATLAB results.

```
b is a linear combination of x(1)*v1+x(2)*v2+x(3)*v3, where
x(1) =
      12

x(2) =
       3

x(3) =
     -21
```

Chapter 3

Matrices and Matrix Algebra

3.1 Operations on Matrices

No MATLAB problems in this section.

3.2 Inverses; Algebraic Properties of Matrices

Exercise 3.1. In this problem, we compute $A^5 - 3A^3 + 7A - 4I$ for the matrix A, where

$$A = \begin{bmatrix} 1 & 2 & -3 & 0 \\ 1 & 1 & -2 & 1 \\ 2 & 1 & 3 & 4 \\ -3 & 2 & 2 & -8 \end{bmatrix}.$$

(a) Using the syntax $A\hat{}k$ which produces the k-th power of a square matrix and the command *eye* for the identity matrix, compute the above matrix polynomial.

(b) Using the command *polyvalm*, compute the above matrix polynomial.

(c) Tell what happens if you type the syntax $A.\hat{}k$.

Solution.

```
% Construct the matrix A.
A = [1 2 -3 0; 1 1 -2 1; 2 1 3 4; -3 2 2 -8];

% (a)
result_a = A^5 + (-3)*A^3 + 7*A + (-4)*eye(4);
```

```
% Display the matrix polynomial.
disp('The result of the matrix polynomial is');
disp(result_a)

% (b)
% Coefficient of the matrix polynomial.
coeff_poly = [1 0 -3 0 7 -4];

% Evaluate the matrix polynomial of coefficient
% with coeff_poly vector with the input matrix A.
result_b = polyvalm(coeff_poly, A);

% Display the matrix polynomial.
disp('The result of the matrix polynomial is');
disp(result_b);

% (c)
disp('The result of A.^2 is'); disp(A.^2);
disp('The result of A.^3 is'); disp(A.^3);
disp('The result of A.^4 is'); disp(A.^4);
```

MATLAB results.

```
The result of the matrix polynomial is
         874       -1272         -39        3021
        2580       -2306        -723        7536
        5191       -4121       -2444       14563
      -16852       12539        5649      -46917

The result of the matrix polynomial is
         874       -1272         -39        3021
        2580       -2306        -723        7536
        5191       -4121       -2444       14563
      -16852       12539        5649      -46917

The result of A.^2 is
     1     4     9     0
     1     1     4     1
     4     1     9    16
     9     4     4    64

The result of A.^3 is
     1     8   -27     0
     1     1    -8     1
     8     1    27    64
   -27     8     8  -512
```

```
The result of A.^4 is
           1          16          81           0
           1           1          16           1
          16           1          81         256
          81          16          16        4096
```

From the results, we can see that the syntax $A.\hat{}\,k$ produces the entrywise k-th powers of the matrix A.

3.3 Elementary Matrices; A Method for Finding A^{-1}

Exercise 3.2. In this problem, we solve the linear system $A\mathbf{x} = \mathbf{b}$ by using matrix inversion, where

$$A = \begin{bmatrix} 3 & 3 & -4 & -3 \\ 0 & 6 & 1 & 1 \\ 5 & 4 & 2 & 1 \\ 2 & 3 & 3 & 2 \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} -2 \\ 3 \\ 5 \\ 1 \end{bmatrix}.$$

(a) Use the MATLAB command *inv* or the syntax $A\hat{}\,(-1)$ to find the inverse of A.

(b) Display the output matrix as a rational form, NOT decimally. You may use the command *format*.

(c) Using the result of (a), compute the solution of the linear system $A\mathbf{x} = \mathbf{b}$ by taking $\mathbf{x} = A^{-1}\mathbf{b}$.

Solution.

```
% Construct the matrix A and the right-hand-side vector b.
A = [3 3 -4 -3; 0 6 1 1; 5 4 2 1; 2 3 3 2];
b = [-2 3 5 1]';

% (a)
% Use the command inv.
Inv_A1 = inv(A);

% Use the syntax A^(-1).
Inv_A2 = A^(-1);

% (b)
format rat;
disp('The result of the command inv is'); disp(Inv_A1);
disp('The result of the syntax A^(-1) is'); disp(Inv_A2);
```

```
% (c)
% Since A is invertible, the solution to Ax=b is x=A^(-1)*b.
x = Inv_A1 * b;
disp('The solution to Ax=b is x = A^(-1)*b'); disp(x');
```

MATLAB results.

```
The result of the command inv is
 -7    5   12  -19
  3   -2   -5    8
 41  -30  -69  111
-59   43   99 -159

The result of the syntax A^(-1) is
 -7    5   12  -19
  3   -2   -5    8
 41  -30  -69  111
-59   43   99 -159

The solution to Ax=b is x = A^(-1)*b
    70      -29     -406      583
```

3.4 Subspaces and Linear Independence

Exercise 3.3. (*Sigma notation*)
Compute the linear combination

$$\mathbf{v} = \Sigma_{j=1}^{25} c_j \mathbf{v}_j$$

for $c_j = 1/j$ and $\mathbf{v}_j = (\sin j, \cos j)$.

Solution.

```
v=zeros(1,2);
for i=1:25
    v=v+(1/i)*[sin(i), cos(i)];
end
disp(v);
```

MATLAB results.

```
1.0322    0.0553
```

Exercise 3.4. Let $\mathbf{v_1} = (4, 3, 2, 1)$, $\mathbf{v_2} = (5, 1, 2, 4)$, $\mathbf{v_3} = (7, 1, 5, 3)$, $\mathbf{x} = (16, 5, 9, 8)$, and $\mathbf{y} = (3, 1, 2, 7)$. Determine whether $\mathbf{x}$ and $\mathbf{y}$ lie in span$\{\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}\}$.

Solution.

```
% Construct v1, v2, v3, x, y
v1=[4 3 2 1]'; v2=[5 1 2 4]'; v3=[7 1 5 3]';
x=[16 5 9 8]'; y=[3 1 2 7]';
```

```
% Augmented matrices [v1|v2|v3|x] and [v1|v2|v3|y]
X=[v1 v2 v3 x];
Y=[v1 v2 v3 y];

disp('Reduced row echelon form of [v1 v2 v3 x] is');
disp(rref(X));
disp('Reduced row echelon form of [v1 v2 v3 y] is');
disp(rref(Y));
```

MATLAB results.

```
Reduced row echelon form of [v1 v2 v3 x] is
       1              0              0              1
       0              1              0              1
       0              0              1              1
       0              0              0              0

Reduced row echelon form of [v1 v2 v3 y] is
       1              0              0              0
       0              1              0              0
       0              0              1              0
       0              0              0              1
```

Therefore, $\mathbf{x}$ lies in span$\{\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}\}$ and $\mathbf{y}$ does not lie in span$\{\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}\}$.

3.5 The Geometry of Linear Systems

No MATLAB problems in this section.

3.6 Matrices with Special Forms

Exercise 3.5. (*Inverting* $(I - A)$)

(a) (*Inverting* $(I - A)$ *when* A *is nilpotent*) Using MATLAB, show that the matrix

$$A = \begin{bmatrix} 2 & 11 & 3 \\ -2 & -11 & -3 \\ 8 & 35 & 9 \end{bmatrix}$$

is nilpotent, and then use Theorem 3.6.6 in the text book to compute $(I - A)^{-1}$. Check your answer by computing the inverse directly in MATLAB.

(b) (*Approximating* $(I - A)^{-1}$ *by a power series*) Using MATLAB, confirm that the matrix

$$A = \begin{bmatrix} 0 & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{4} & \frac{1}{8} & \frac{1}{10} \\ \frac{1}{8} & \frac{1}{10} & \frac{1}{10} \end{bmatrix}$$

satisfies the condition in Theorem 3.6.7 of the text book. You may use the command *sum*. Since A satisfies that condition, $(I - A)$ is invertible and can be expressed by the series in Formula (18) in Section 3.6 of the text book. Compute the approximation

$$(I - A)^{-1} \approx I + A + A^2 + A^3 + \cdots + A^{10},$$

and compare it with the inverse of $I - A$ produced directly by MATLAB. To how many decimal places do the results agree? You may use the command *format* to display the output with long digits.

Solution.

(a)
```
% (a)-i
A = [ 2 11 3 ; -2 -11 -3; 8 35 9];  % Construct the matrix A.
% Compute the A^2, A^3, ... , and display.
disp('A^2 is'); disp(A^2);
disp('A^3 is'); disp(A^3);

% (a)-ii Comparing two result

% By Theorem 3.6.6, (I-A)^(-1)=I+A+A^2.
result1=eye(3)+A+A^2;

% Compute the inverse of (I-A) directly.
result2=inv(eye(3)-A);
disp('I+A+A^2 is'); disp(result1);
```

```
disp('(I-A)^(-1) is'); disp(result2);

% Display as a rational form.
format rat;
disp('Rational form of (I-A)^(-1) is');disp(result2);
```

MATLAB results.

```
A^2 is
     6     6     0
    -6    -6     0
    18    18     0

A^3 is
     0     0     0
     0     0     0
     0     0     0

I+A+A^2 is
     9    17     3
    -8   -16    -3
    26    53    10

(I-A)^(-1) is
    9.0000   17.0000    3.0000
   -8.0000  -16.0000   -3.0000
   26.0000   53.0000   10.0000

Rational form of (I-A)^(-1) is
       9            17             3
      -8           -16            -3
      26            53            10
```

Since $A^3 = \mathbf{0}$, A is nilpotent. By the Theorem 3.6.6, since $A^3 = \mathbf{0}$, $I - A$ is invertible and $(I - A)^{-1} = I + A + A^2$. To check answer by computing the inverse directly in MATLAB, we implement as in the next page.

(b)

```
% Construct the matrix A.
A=[0 1/4 1/8; 1/4 1/8 1/10; 1/8 1/10 1/10];

% Check that the condition in Theorem 3.6.7
% of the text book is satisfied for matrix A.
column_sum=sum(abs(A),1);   % column-wise sum
row_sum=sum(abs(A),2);  % row-wise sum
disp('The sum of the absolute values of the entries in each column is');
disp(column_sum);
disp('The sum of the absolute values of the entries in each row is');
disp(row_sum);
```

```
result3=eye(size(A))+A+A^2+A^3+A^4+A^5+A^6+A^7+A^8+A^9+A^10;
result4=inv(eye(3)-A);

format long; % Display the result with long digits
disp('With format long');
disp('Approximated inv(I-A) is'); disp(result3);
disp('Exact inv(I-A) is'); disp(result4);
```

MATLAB results.

```
The sum of the absolute values of the entries in each column is
       3/8           19/40          13/40

The sum of the absolute values of the entries in each row is
       3/8
      19/40
      13/40

With format long
Approximated inv(I-A) is
   1.108587459181130   0.338615080927493   0.191581699462210
   0.338615080927493   1.260966638806045   0.187122081247432
   0.191581699462210   0.187122081247432   1.158500720998029

Exact inv(I-A) is
   1.108610894508188   0.338643199287067   0.191600757491367
   0.338643199287067   1.261000334187368   0.187144925921800
   0.191600757491367   0.187144925921800   1.158516208087334
```

The approximation result agrees with the exact result to 2 decimal places.

3.7 Matrix Factorizations; LU-Decomposition

Exercise 3.6. (*LU-decompositions*) In this problem, we find an LU-decomposition of A, where A is given in the Example 2 of the Section 3.7.

(a) Find an LU-decomposition of A by following the procedure given in the Example 2.

(b) Solve the linear system $A\mathbf{x} = \mathbf{b}$ by using the LU-decomposition of A obtained in (a), where $\mathbf{b} = \begin{bmatrix} 0 \\ -2 \\ 1 \end{bmatrix}$.

(c) Tell what happens if you use the MATLAB command *lu* of A. Explain why this result differs from the result in (a).

Solution.

```
%(a)
A = [6 -2 0; 9 -1 1; 3 7 5]; % Set the matrix A.

format rat; % Display results as a rational form.

% Initialization of U and L.
U = A; L = eye(3);

% Multiply the first row by 1/6.
U(1,:)=(1/6)*U(1,:);
% L(1,1) is the inverse of 1/6.
L(1,1)=(1/6)^(-1);

% Add (-9) times the first to the second.
U(2,:)=((-9)*U(1,:))+U(2,:);
% L(2,1) is the negative of (-9).
L(2,1)=-(-9);

% Add (-3) times the first to the third.
U(3,:)=((-3)*U(1,:))+U(3,:);
% L(3,1) is the negative of (-3).
L(3,1)=-(-3);

% Multiply the second row by 1/2.
U(2,:)=(1/2)*U(2,:);
% L(2,2) is the inverse of 1/2.
L(2,2)=(1/2)^(-1);

% Add (-8) times the second to the third.
U(3,:)=((-8)*U(2,:))+U(3,:);
% L(3,2) is the negative of (-8).
L(3,2)=-(-8);

disp('A is'); disp(A);
disp('The Lower Triangular part L is'); disp(L);
disp('The Upper Triangular part U is'); disp(U);
disp('The product L*U is'); disp(L*U);

%(b)
% Solve the linear system Ax=b
% by using the LU-decomposition obtained in (a).

% First, let us solve L*y = b by forward substitution.
 % Set the right-hand-side vector b.
b = [0 -2 1]';
```

```
 % Initialization of the solution vector y.
y = zeros(3, 1);
y(1) = b(1) / L(1, 1);
y(2) = (b(2) - (L(2, 1)*y(1))) / L(2, 2);
y(3) = (b(3) - (L(3, 1)*y(1)) - (L(3, 2)*y(2))) / L(3, 3);

% Next, let us solve U*x = y by backward substitution.
x = zeros(3, 1); % Initialization of the solution vector x.
x(3) = y(3) / U(3, 3);
x(2) = (y(2) - (U(2, 3)*x(3))) / U(2, 2);
x(1) = (y(1) - (U(1, 3)*x(3)) - (U(1, 2)*x(2))) / U(1, 1);

disp('The solution to Ax=b by the LU-decomposition is'); disp(x');

% (c)
fprintf('Using MATLAB command lu\n');
% LU decomposition of A with a permutation matrix.
[L U P] = lu(A);

disp('Lower triangular part L is'); disp(L);
disp('Upper triangular part U is'); disp(U);
disp('The permutation matrix P is'); disp(P);
disp('PA='); disp(P*A); disp('LU='); disp(L*U);
```

MATLAB results.

```
A is
       6             -2             0
       9             -1             1
       3              7             5

The Lower Triangular part L is
       6              0             0
       9              2             0
       3              8             1

The Upper Triangular part U is
       1             -1/3           0
       0              1             1/2
       0              0             1

The product L*U is
       6             -2             0
       9             -1             1
       3              7             5
```

```
The solution to Ax=b by the LU-decomposition is
     -11/6          -11/2            9

Using MATLAB command lu

Lower triangular part L is
       1              0              0
       1/3            1              0
       2/3           -2/11           1
Upper triangular part U is
       9             -1              1
       0             22/3           14/3
       0              0              2/11
The permutation matrix P is
       0              1              0
       0              0              1
       1              0              0
PA=
       9             -1              1
       3              7              5
       6             -2              0
LU=
       9             -1              1
       3              7              5
       6             -2              0
```

Since the permutation matrix P is not the identity matrix, the MATLAB command *lu* gave us an LU-decomposition after multiplying A by the permutation matrix P, hence, this decomposition is a PLU-decomposition of A because $PA = LU$. Since at least one row interchange of A occurred in the process of LU-decomposition, this result is different from the previous decomposition obtained in (a).

Exercise 3.7. (*LU-decomposition*)

(a) The MATLAB command *lu* is used to find the LU-decomposition of a matrix A. Tell what happens if you use the command *lu* for A, where A is given in the Example 2 of the Section 3.7. Explain why this result differs from the result in the textbook.

(b) Using MATLAB, observe what happens when you try to find an LU-decomposition of a singular matrix.

Solution.

```
% (a)
```

```
% Construct the matrix A.
A=[6 -2 0; 9 -1 1; 3 7 5];

% LU decomposition of A.
[L U P]=lu(A);
disp('[L U P]=lu(A)');
disp('L'); disp(L); disp('U'); disp(U); disp('P'); disp(P);

% (b)
% Construct the some singular matrices.
A1=[1 0 0; -2 0 0; 4 6 1];
A2=[1 -2 7; -4 8 5; 2 -4 3];
A3=[1 0 0; -2 0 0; 4 6 1];

% LU decompositions of them.
[L1 U1 P1]=lu(A1); [L2 U2 P2]=lu(A2); [L3 U3 P3]=lu(A3);
disp('[L1 U1 P1]=lu(A1)'); disp('L1');disp(L1);disp('U1');disp(U1);
disp('[L2 U2 P2]=lu(A2)'); disp('L2');disp(L2); disp('U2');disp(U2);
disp('[L3 U3 P3]=lu(A3)'); disp('L3');disp(L3); disp('U3');disp(U3);
```

MATLAB results.

```
[L U P]=lu(A)
L
    1.0000         0         0
    0.3333    1.0000         0
    0.6667   -0.1818    1.0000

U
    9.0000   -1.0000    1.0000
         0    7.3333    4.6667
         0         0    0.1818

P
     0     1     0
     0     0     1
     1     0     0


[L1 U1 P1]=lu(A1)
L1
    1.0000         0         0
   -0.5000    1.0000         0
    0.2500   -0.5000    1.0000
U1
    4.0000    6.0000    1.0000
         0    3.0000    0.5000
```

```
         0         0         0

[L2 U2 P2]=lu(A2)
L2
    1.0000         0         0
   -0.2500    1.0000         0
   -0.5000         0    1.0000
U2
   -4.0000    8.0000    5.0000
         0         0    8.2500
         0         0    5.5000

[L3 U3 P3]=lu(A3)
L3
    1.0000         0         0
   -0.5000    1.0000         0
    0.2500   -0.5000    1.0000
U3
    4.0000    6.0000    1.0000
         0    3.0000    0.5000
         0         0         0
```

Remark on (a). Since the permutation matrix P is not the identity matrix, the MATLAB command *lu* gave us an LU-decomposition after multiplying A by the permutation matrix P, hence, this decomposition is a PLU-decomposition of A because $PA = LU$. Since at least one row interchange of A occurred in the process of LU-decomposition, this result is different from the decomposition result in the textbook.

Remark on (b). When we try LU-decomposition of the sigular matrices using the MATLAB command *lu*, the resulting upper triangular matrices are singular.

Chapter 4

Determinants

4.1 Determinants; cofactor Expansion

Exercise 4.1. Compute the determinants of the matrix A:

$$A = \begin{bmatrix} -4 & 1 & 1 & 1 & 1 \\ 1 & -4 & 1 & 1 & 1 \\ 1 & 1 & -4 & 1 & 1 \\ 1 & 1 & 1 & -4 & 1 \\ 1 & 1 & 1 & 1 & -4 \end{bmatrix}.$$

How can you construct A brilliantly?

Solution.

```
A=ones(5)-5*eye(5);
disp('A is'); disp(A);
disp('Determinant of A is'); disp(det(A));
```

MATLAB results.

```
A is
    -4     1     1     1     1
     1    -4     1     1     1
     1     1    -4     1     1
     1     1     1    -4     1
     1     1     1     1    -4

Determinant of A is
  -5.5511e-14
```

Exercise 4.2. Show that

$$\det\left(\begin{bmatrix} a & b & c & d \\ -b & a & d & -c \\ -c & -d & a & b \\ -d & c & -b & a \end{bmatrix}\right) = (a^2+b^2+c^2+d^2)^2.$$

Solution.

```
syms a b c d;

A=[a b c d; -b a d -c; -c -d a b; -d c -b a];

disp('Given matrix is'); disp(A);
disp('Determinant of the given matrix is');
disp(simplify(det(A)));
```

MATLAB results.

```
Given matrix is
[  a,  b,  c,  d]
[ -b,  a,  d, -c]
[ -c, -d,  a,  b]
[ -d,  c, -b,  a]

Determinant of the given matrix is
(a^2 + b^2 + c^2 + d^2)^2
```

Exercise 4.3. The nth-order **Fibonacci matrix** [named for the Italian mathematician (circa 1170 - 1250)] is the $n \times n$ matrix F_n that has 1's on the main diagonal, 1's along the diagonal immediately above the main diagonal, -1's along the diagonal immediately below the main diagonal, and zeros everywhere else. Construct the sequence

$$\det(F_1), \det(F_2), \det(F_3), \cdots, \det(F_7).$$

Make a conjecture about the relationship between a term in the sequence and its two immediate predecessors, and then use your conjecture to make a guess at $\det(F_8)$. Check your guess by calculating this number.

Solution.

```
% Construct the 10x10 Fibonacci matrix F.
N=10; nOnes=ones(N, 1);
F=diag(nOnes)+diag(nOnes(1:N-1),1)-diag(nOnes(1:N-1),-1);

for n=1:7 % n is from 1 to 7
    Fn=F(1:n,1:n); % nxn Fibonacci matrix is selected from F.
    disp(det(Fn));
end
```

MATLAB results.

```
     1
     2
     3
     5
     8
    13
    21
```

The constructed sequence satisfies the relationship $\det(F_n) = \det(F_{n-1}) + \det(F_{n-2})$, for $\det(F_1) = 1$ and $\det(F_2) = 2$. From that, we may guess that $\det(F_8) = 34$. MATLAB gives us the same output value 34 as our guess.

Exercise 4.4. Let A_n be the $n \times n$ matrix that has 2's along the main diagonal, 1's along the diagonals immediately above and below the main diagonal, and zeros everywhere else. Make a conjecture about the relationship between n and $\det(A_n)$.

Solution.

```
format rat;
% Construct the 10x10 matrix A satisfying given conditions.
n=10; nOnes=ones(n, 1);
A=2*diag(nOnes)+diag(nOnes(1:n-1),1)+diag(nOnes(1:n-1),-1);

for i=1:10 % i is from 1 to 10
    Ai=A(1:i,1:i); % A_i matrix is selected from A.
    disp(det(Ai));
end
```

MATLAB results.

```
       2
       3
       4
       5
       6
       7
       8
       9
      10
      11
```

From the outputs, we make a conjecture about the relationship between n and $\det(A_n)$ as follows:

$$\det(A_n) = n + 1.$$

4.2 Properties of Determinants

Exercise 4.5. (*Determinants with LU-decomposition*) In this problem, we find the determinant of the matrix A by using the LU-decomposition of A, where

$$A = \begin{bmatrix} -2 & 2 & -4 & -6 \\ -3 & 6 & 3 & -15 \\ 5 & -8 & -1 & 17 \\ 1 & 1 & 11 & 7 \end{bmatrix}.$$

(a) Compute the determinant of A directly by using the MATLAB command *det* for A.

(b) Compute the determinant of A by using the MATLAB command *lu* for A. Confirm that you get the same results.

Solution.

```
%(a)
A = [-2 2 -4 -6; -3 6 3 -15; 5 -8 -1 17; 1 1 11 7];

det_A = det(A); % Find the determinant of A by using the command det.

disp('The determinant of A by direct use of the command det is');
disp(det_A);

%(b)
[L U P] = lu(A); % We have a PLU-decomposition of A. (i.e., PA=LU ).

% Since the determinant of a triangular matrix is
% just a product of diagonal entries,

det_L = prod(diag(L)); % The product of diagonal entries of L.
% Or, you may use the command det for L, directly. (i.e., det_L = det(L)).

det_U = prod(diag(U)); % The product of diagonal entries of U.
% Or, you may use the command det for U, directly. (i.e., det_U = det(U)).

% If you observe the permutation matrix P, you can see that
% P is an odd permutation. Thus, we have det(P) = -1.
det_P = -1;
% Or, you may use the command det for P, directly. (i.e., det_P = det(P)).

% Since PA = LU, det(P)*det(A) = det(L)*det(U).
det_A = det_P * det_L * det_U;

disp('The determinant of A by using the LU-decomposition is'); disp(det_A);
```

MATLAB results.

```
The determinant of A by direct use of the command det is
   24.0000

The determinant of A by using the PLU-decomposition is
   24.0000
```

Exercise 4.6. (*Effects of Elementary Row Operations on the Determinant*)
Using the MATLAB command *det*, confirm the formulas (a)-(c) in Theorem 4.2.2 of Section 4.2 for the matrix A given in the problem 31 of Exercise set 4.1.

Solution.

```
A = [3 3 0 5; 2 2 0 -2; 4 1 -3 0; 2 10 3 2];

% (a). Multiply the second row of A by 2 and call it A2.
% Initialize the matrix A2 as A.
A2 = A;
% Multiply the second row of A by 2.
A2(2,:) = 2*A(2,:);
disp('The determinant of A2 is'); disp(det(A2));
disp('2*det(A) = '); disp(2*det(A));

% (b). Interchange the rows 2 and 4 of A and call it A24.
% Initialize the matrix A24 as A.
A24 = A;
% Interchange the rows 2 and 4 of A.
A24(2, :) = A(4, :) ; A24(4, :) = A(2, :);
disp('The determinant of A24 is'); disp(det(A24));
disp('-det(A) = '); disp(-det(A));

% (c). Add 2 times row 3 to row 4 of A and call it A234.
% Initialize the matrix A234 as A.
A234 = A;
% Add 2 times row 3 of A to row 4.
A234(4, :) = 2 * A(3, :) + A(4, :);
disp('The determinant of A234 is'); disp(det(A234));
disp('det(A) = '); disp(det(A));
```

MATLAB results.

```
The determinant of A2 is
  -480
2*det(A) =
 -480.0000

The determinant of A24 is
  240.0000
-det(A) =
  240.0000

The determinant of A234 is
 -240.0000
det(A) =
 -240.0000
```

Exercise 4.7. Use a determinant to show that if a, b, c, and d are not all zeros,

then the vectors

$$\begin{aligned} \mathbf{v}_1 &= (a,\, b,\, c,\, d) \\ \mathbf{v}_2 &= (-b, a, d, -c) \\ \mathbf{v}_3 &= (-c, -d, a, b) \\ \mathbf{v}_4 &= (-d, c, -b, a) \end{aligned}$$

are linearly independent.

Solution.

```
syms a b c d;
v1=[a b c d];
v2=[-b a d -c];
v3=[-c -d a b];
v4=[-d c -b a];

V=[v1; v2; v3; v4];
disp('det(V) is'); disp(simplify(det(V)));
```

MATLAB results.

```
det(V) is
(a^2 + b^2 + c^2 + d^2)^2
```

4.3 Cramer's Rule; Formula for A^{-1}; Applications

No MATLAB problems in this section.

4.4 A First Look at Eigenvalues and Eigenvectors

Exercise 4.8. (*Eigenvalues and Eigenvectors*)
Use the MATLAB command *eig* to find the eigenvalues and the associated eigenvectors of the matrix A, where

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

Display the results with long digits.

Solution.

```
% Construct the matrix A.
A=[2 -3 1 0; 1 1 2 2; 3 0 -1 4; 1 6 5 6];

% Find the eigenvalues and eigenvectors of A by using eig.
% This command gives AQ = QD.
[Q D] = eig(A);
lambda1 = D(1,1); lambda2 = D(2,2);
lambda3 = D(3,3); lambda4 = D(4,4);

% Extract each column vector as an eigenvector of A.
x1 = Q(:,1); x2 = Q(:,2); x3 = Q(:,3); x4 = Q(:,4);

% Display the result with long digits.
format long;
disp('lambda1 is'); disp(lambda1);
disp('The eigenvector corresponding to lambda1 is'); disp(x1');
disp('lambda2 is'); disp(lambda2);
disp('The eigenvector corresponding to lambda2 is'); disp(x2');
disp('lambda3 is'); disp(lambda3);
disp('The eigenvector corresponding to lambda3 is'); disp(x3');
disp('lambda4 is'); disp(lambda4);
disp('The eigenvector corresponding to lambda4 is'); disp(x4');
```

MATLAB results.

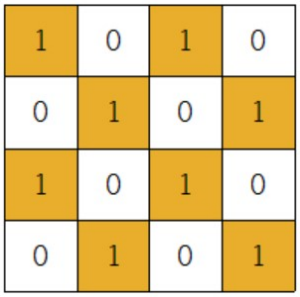
```
lambda1 is
  9.561855032395805
The eigenvector corresponding to lambda1 is
 -0.067716707308095  0.278176502030497  0.322465582156500  0.902246213399589
lambda2 is
 -3.364648937746373
The eigenvector corresponding to lambda2 is
  0.275562522991092  0.197508356444458 -0.885771126913498  0.316962546342283
lambda3 is
```

```
  1.802793905350564
The eigenvector corresponding to lambda3 is
 -0.833621905475750 -0.103812731179200 -0.147042873144503  0.522183711938150
lambda4 is
 -3.860931435448914e-16
The eigenvector corresponding to lambda4 is
 -0.705886578756789 -0.456750139195570  0.041522739926871  0.539795619049310
```

Remark. In fact, if we compute λ_4 by hand, we can obtain that $\lambda_4 = 0$. However, from the result, we see that the resulting value of λ_4 seems to be nonzero even though it is small enough. This is due to roundoff errors in arithmetic operations. Please refer to the help command of *eps*, then you can see that $eps = 2.220446049250313e-016$ is floating-point relative accuracy, which means that *eps* value is the allowable tolerance when we do numerical computations with rounding floating-point number off. (*i.e.*, *eps* is an upper bound on the relative error due to rounding in floating point arithmetic.) Therefore, we can regard the resulting value of λ_4 as zero.

Exercise 4.9. (*Eigenvalues and Eigenvectors*)

Define an nth-order checkboard matrix C_n to be a matrix that has a 1 in the upper left corner and alternates between 1 and 0 along rows and columns (see the figure below). Find the eigenvalues of $C_1, C_2, \cdots$ to make a conjecture about the eigenvalues of C_n. What can you say about the eigenvalues of C_n?



Solution.

```
format short;
n=10;   % Set the size of the large check board

% Construct your checkboard
CheckBoard=zeros(n);
CheckBoard(1:2:n, 1:2:n)=1;
CheckBoard(2:2:n, 2:2:n)=1;
for i=1:n
    Cn=CheckBoard(1:i, 1:i);
    [Qn Dn]=eig(Cn);    % Eigenvectors and eigenvalues
    fprintf('The size of the checkboard is %d \n',i);
```

```
    disp(diag(Dn)');
end
```

MATLAB results.

```
The size of the checkboard is 1
     1
The size of the checkboard is 2
     1 1
The size of the checkboard is 3
     0 1 2
The size of the checkboard is 4
     0 0 2 2
The size of the checkboard is 5
   -0.0000 -0.0000  0.0000  2.0000  3.0000
The size of the checkboard is 6
   -0.0000  -0.0000  -0.0000  -0.0000  3.0000  3.0000
The size of the checkboard is 7
   -0.0000  -0.0000   0.0000   0.0000  0.0000  3.0000  4.0000
The size of the checkboard is 8
   -0.0000  -0.0000  -0.0000  0.0000  0.0000  0.0000  4.0000  4.0000
The size of the checkboard is 9
   -0.0000  -0.0000  -0.0000 -0.0000   0   0.0000  0.0000  4.0000  5.0000
The size of the checkboard is 10
   -0.0000  -0.0000  -0.0000  0  0.0000  0.0000  0.0000  0.0000  5.0000  5.0000
```

We may conclude that the eigenvalues of C_n are given as follows:

$$\begin{cases} 1 & \text{if } n = 1, \\ k,\ k,\ \underbrace{0,\ 0,\ \cdots,\ 0}_{(n-2)} & \text{if } n = 2k, \\ k,\ k+1,\ \underbrace{0,\ 0,\ \cdots,\ 0}_{(n-2)} & \text{if } n = 2k+1, \end{cases}$$

where k is a positive integer.

Chapter 5

Matrix Models

No MATLAB problems in this chapter.

Chapter 6

Linear Transformations

6.1 Matrices as Transformations

Exercise 6.1. (*Linear Transformation: Rotation*)
Make a function file with a function name *reflect_pt* to find the reflection of the point (a, b) about the line through the origin of the xy-plane that makes an angle of θ° with the positive x-axis. Make a, b, and θ the inputs to the function and the reflection point (x, y) the output. Using this function, find the result of this function for $(a, b) = (1, 3)$ and $\theta = 12$ by the following commands:

```
>> [x, y]=reflect_pt(1, 3, 12)
```

Solution.

```
%--- The following is the function file 'reflect_pt.m'. ---%
function [x, y]=reflect_pt(a, b, ang)
  theta=ang*(pi/180);
  T = [cos(2*theta) sin(2*theta); sin(2*theta) -cos(2*theta)];

  result=T*[a b]';
  x=result(1); y=result(2);
end
```

MATLAB results.

```
x =
    2.1338

y =
   -2.3339
```

Exercise 6.2. (*Linear Transformation: Projection*)
Consider successive rotations of $\mathbb{R}^3$ by an angle θ_1 degree about the x-axis, then by an angle θ_2 degree about the y-axis, and then by an angle θ_3 degree about

the z-axis. Make a function file named *comp_Rot* to find an appropriate axis and angle of rotation that achieves the same result in one rotation. Make θ_1, θ_2, and θ_3 degrees the inputs to the function and the axis and angle of rotation the outputs. Give the output axis as a unit vector. You may make the standard matrix for the composition of the rotations by multiplication of the standard matrices for the rotations about the position x-, y-, and z-axes, respectively. Using the function *comp_Rot*, check the result of the following commands:

```
>> [L ang]=comp_Rot(45, 45, 45)
```

Solution.

```
%--- The following is the function file 'comp_Rot.m'. ---%
function [L, ang] = comp_Rot(ang1, ang2, ang3)

  % Angle as a degree.
  angle=[ang1 ang2 ang3];

  % Convert angles from degrees to radians.
  theta=angle*(pi/180);

  % Rotation about x-axis, y-axis, and z-axis.
  Rx = [1 0 0;
        0 cos(theta(1)) -sin(theta(1));
        0 sin(theta(1)) cos(theta(1))];
  Ry = [cos(theta(2)) 0 sin(theta(2));
        0 1 0;
        -sin(theta(2)) 0 cos(theta(2))];
  Rz = [cos(theta(3)) -sin(theta(3)) 0;
        sin(theta(3)) cos(theta(3)) 0;
        0 0 1];

  % Composition of the three rotation matrices R = Ry*Rx*Rz.
  R = Rz*Ry*Rx;

  % Find the axis of rotation of R,
  % by finding the eigenvector of R
  % corresponding to the eigenvalue lambda=1.

  % Find a nonzero vector satisfying Rx = x.
  L = null(eye(3) - R);

  % Make the axis of rotation unit vector.
  L = L/norm(L);

  % Find the angle of rotation of R.
  % Note that w=(-x2,x1,0) is
```

```
  % orthogonal to the axis of rotation x=(x1,x2,x3).
  w = [-L(2) L(1) 0]';
  rot_theta = acos((dot(w, R*w))/((norm(w)*norm(R*w))));
  ang = ((rot_theta)*(180/pi));
end
```

MATLAB results.

```
L =
   -0.3574
   -0.8629
   -0.3574

ang =
   64.7368
```

6.2 Geometry of Linear Operators

Exercise 6.3. (*Rotation as an Orthogonal Transformation*)
Let

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

Show that A represents a rotation, and use Formulas (16) and (17) in Section 6.2 to find the axis and angle of rotation.

Solution.

```
A = [-3/7 -2/7 -6/7; 6/7 -3/7 -2/7; -2/7 -6/7 3/7]; format short;

% Check that A*A' is the identity matrix.
A*A'

% Although the off diagonal entries are not exactly all zeros,
% the scaling factor suggests that roundoff error prevents
% the computed matrix from being the identity matrix.
% You can see that the product is exactly the identity matrix,
% when the symbolic computation is used.

% Check that det(A)=1 to conclude that A is orthogonal.
det(A)

% Since A*A'=I and det(A) = 1, A represents a rotation.

% By (16) of Section 6.2,
% find the angle of rotation
% and convert the angle from radians to degrees.
theta = acos((trace(A)-1)/2); ang = ((theta)*(180/pi));
disp('The angle of rotation in degrees is'); disp(ang);

% By (17) of Section 6.2, find the axis of rotation.
% The initial point at the origin.
e1 = [1 0 0]';

% v is along the axis of rotation.
v = (A+A'+((1-trace(A))*eye(3))) * e1;
disp('The axis of rotation passes through the point'); disp(v');
```

MATLAB results.

```
The angle of rotation in degrees is
```

```
  135.5847
The axis of rotation passes through the point
    0.5714    0.5714   -1.1429
```

6.3 Kernel and Range

Exercise 6.4. (*Invertible Matrix as a Bijective Linear Transformation*) Consider the matrix

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

Referring to the Theorem 6.3.15 in Section 6.3, show that $T_A : \mathbb{R}^4 \to \mathbb{R}^4$ is onto in at least four different ways. You may use several related MATLAB commands.

Solution.

```
A = [3 -5 -2 2; -4 7 4 4; 4 -9 -3 7; 2 -6 -3 2];
format short;

% By (a) in Theorem 6.3.15, use the command rref of A.
rref(A)

% Since the reduced row echelon form of A is the identity matrix,
% the linear transformation T is onto.

% By (d) in Theorem 6.3.15, use the command null of A.

% Find a basis for the null space of A.
null(A, 'r')

% Since the null space contains only the zero vector,
% the linear transformation T is onto.

% By (g) in Theorem 6.3.15, use the command rank of A.

% Find the number of linearly independent columns of A.
rank(A)

% Since the column vectors of A are linearly independent,
% the linear transformation T is onto.

% By (i) in Theorem 6.3.15, use the command det of A.
```

```
% Find the determinant of A.
det(A)

% Since det(A) is nonzero,
% the linear transformation T is onto.

% By (j) in Theorem 6.3.15, use the command eig of A.

% Find the eigenvalues A.
eig(A)

% Since 0 is not an eigenvalue of A,
% the linear transformation T is onto.
```

6.4 Composition and Invertibility of Linear Transformations

Exercise 6.5. (*Compositions of Linear Transformations*)
Consider successive rotations of $\mathbb{R}^3$ by $30\,^\circ$ about the z-axis, then by $60\,^\circ$ about the x-axis, and then by $45\,^\circ$ about the y-axis. If it is desired to execute the three rotations by a single rotation about an appropriate axis, what axis and angle should be used?

Solution.

```
% Angle as a degree.
ang1=30; ang2=60; ang3=45;

% Convert angles from degrees to radians.
theta1=((ang1)*(pi/180));
theta2=((ang2)*(pi/180));
theta3=((ang3)*(pi/180));

format short;

% Rotation about z-axis with the angle 30.
Rz = [cos(theta1) -sin(theta1) 0; sin(theta1) cos(theta1) 0; 0 0 1];
% Rotation about x-axis with the angle 60.
Rx = [1 0 0; 0 cos(theta2) -sin(theta2); 0 sin(theta2) cos(theta2)];
% Rotation about y-axis with the angle 45.
Ry = [cos(theta3) 0 sin(theta3); 0 1 0; -sin(theta3) 0 cos(theta3)];

% Composition of the three rotation matrices R = Ry*Rx*Rz.
R = Ry*Rx*Rz;

% Find the axis of rotation of R,
```

```
% by finding the eigenvector of R corresponding to the eigenvalue lambda=1.

% Find a nonzero vector satisfying Rx = x.
x = null(eye(3) - R);

% Find the angle of rotation of R.
% Note that w=(-x2,x1,0) is orthogonal to the axis of rotation x=(x1,x2,x3).
w = [-x(2) x(1) 0]';
theta = acos((dot(w, R*w))/((norm(w)*norm(R*w))));

% Convert the angle from radians to degrees.
ang = ((theta)*(180/pi));

disp('The angle of rotation in degrees is'); disp(ang);
disp('The axis of rotation is'); disp(x');
```

MATLAB results.

```
The angle of rotation in degrees is
   69.3559

The axis of rotation is
   -0.9350   -0.3525   -0.0391
```

Chapter 7

Dimension and Structure

7.1 Basis and Dimension

Exercise 7.1. (*Linear Combination and Independence*)
Are any of the vectors in the set

$$S = \{(2, 6, 3, 4, 2),\ (3, 1, 5, 8, 3),\ (5, 1, 2, 6, 7),\ (8, 4, 3, 2, 6),\ (5, 5, 6, 3, 4)\}$$

linear combinations of predecessors? Justify your answer.

Solution. One strategy is to form a matrix V of the column vectors $\mathbf{v}_k$ mentioned above and decide whether the system $V\mathbf{x} = \mathbf{0}$ has nontrivial solutions. If so, then at least one column is a linear combination of previous ones. Otherwise, the columns are linearly independent.

```
v1 = [2 6 3 4 2]'; v2 = [3 1 5 8 3]'; v3 = [5 1 2 6 7]';
v4 = [8 4 3 2 6]'; v5 = [5 5 6 3 4]';

% Construct V of the column vectors v1,v2,v3,v4 and v5.
V = [v1 v2 v3 v4 v5];

format short;

% Find the reduced row echelon form of V.
rref_V = rref(V);

disp('The reduced row echelon form of A is'); disp(rref_V);
```

MATLAB results.

```
The reduced row echelon form of A is
     1     0     0     0     0
     0     1     0     0     0
```

```
0     0     1     0     0
0     0     0     1     0
0     0     0     0     1
```

Since the reduced row echelon form of V has 5 pivots, the columns of V are linearly independent. Hence, no column of V can be a linear combination of any other columns.

7.2 Properties of Bases

Exercise 7.2. In this problem, we make a function file `CheckBasis.m` to check that the vectors $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ and $\mathbf{v}_4$ form a basis of $\mathbb{R}^4$ using the equivalent statements (a), (g), (h), and (o) of Theorem 7.2.7 in the textbook.

(a) Complete the shadow part (//////) of the m-file given below referring to the comments and the execution results.

```
%--- your function file ---%
function [Result]=CheckBasis(v1, v2, v3, v4, case_num)
  %  if case_num=1, check the statement (a),
  %  if case_num=2, check the statement (g),
  %  if case_num=3, check the statement (h).

  % Construct the matrix V.
  ////////////////////////

  % Use the switch statement to check
  % whether one of the statements (a), (g), and (h) holds.
  switch case_num
    case 1
      fprintf('* You enter %d: statement (a) *\n', case_num);
      ////////////////////////
      if ////////////////////////
        disp('Given vectors form a basis of 4 dimensional space.');
      else
        disp('Given vectors do not form a basis of 4 dimensional space.');
      end
    case 2
      fprintf('* You enter %d: statement (g) *\n', case_num);
      Result=det(V);
      if Result~=0
        disp('Given vectors form a basis of 4 dimensional space.');
      else
        disp('Given vectors do not form a basis of 4 dimensional space.');
```

```
      end
    ////////////////////////  % check statement (h)
        ////////////////////////
        ////////////////////////
        ////////////////////////
        ////////////////////////
        ////////////////////////
        ////////////////////////
        ////////////////////////
  end
end
```

The execution results will be as follows:

```
>> v1=[1 0 0 0]'; v2=[0 2 0 0]'; v3=[0 0 4 5]'; v4=[0 0 0 -1]'; v5=[0 0 0 1]';
>> C=CheckBasis(v1, v2, v3, v4,3)
* You enter 3: statement (h) *
  Given vectors are basis of 4 dimensional space.

C =

     1
     2
    -1
     4

>> CheckBasis(v1, v2, v4, v5, 1);
* You enter 1: statement (a) *
  Given vectors do not form a basis of 4 dimensional space.

>> determinant=CheckBasis(v1, v2, v3, v5, 2)
* You enter 2: statement (g) *
  Given vectors form a basis of 4 dimensional space.

determinant =
     8
```

(b) Using `CheckBasis.m` from (a), check whether

i. $\mathbf{v}_1 = (-1, 0, 1, 0)^T$, $\mathbf{v}_2 = (2, 3, -2, 6)^T$, $\mathbf{v}_3 = (0, -1, 2, 0)^T$ and $\mathbf{v}_4 = (0, 0, 1, 5)^T$ form a basis of $\mathbb{R}^4$.

ii. $\mathbf{v}_1 = (a, b, c, d)^T$, $\mathbf{v}_2 = (-b, a, d, -c)^T$, $\mathbf{v}_3 = (-c, -d, a, b)^T$ and $\mathbf{v}_4 = (-d, c, -b, a)^T$ form a basis of $\mathbb{R}^4$. (Do not use the statement (h). Guess why not?)

Solution.

(a)
```
% ----- your function file ----- %
function [Result]=CheckBasis(v1, v2, v3, v4, case_num)
  %  if case_num=1, check the statement (a),
  %  if case_num=2, check the statement (g),
  %  if case_num=3, check the statement (h).

 % Construct the matrix V.
 V=[v1 v2 v3 v4];

  % Use the switch statement to check
  % whether one of the statements (a), (g), and (h) holds.
  switch case_num
    case 1
      fprintf('* You enter %d: statement (a) *', case_num);
      Result=rref(V)
      if det(Result)~=0
        disp('Given vectors form a basis of 4 dimensional space.');
      else
        disp('Given vectors do not form a basis of 4 dimensional space.');
      end
    case 2
      fprintf('* You enter %d: statement (g) *', case_num);
      Result=det(V);
      if Result~=0
        disp('Given vectors form a basis of 4 dimensional space.');
      else
        disp('Given vectors do not form a basis of 4 dimensional space.');
      end
    case 3  % check statement (h)
      [Q D]=eig(V);
      Result=diag(R);
      if det(R)==0
        disp('Given vectors form a basis of 4 dimensional space.');
      else
        disp('Given vectors do not form a basis of 4 dimensional
space.');
      end
    end
```

(b)-i.
```
>> v1=[-1 0 1 0]'; v2=[2 3 -2 6]';  v3=[0 -1 2 0]'; v4 = [0 0 1 5]';
>> CheckBasis(v1, v2, v3, v4, 1);
>> CheckBasis(v1, v2, v3, v4, 2);
```

```
>> CheckBasis(v1, v2, v3, v4, 3);
```

MATLAB results.

```
* You enter 1: statement (a) *
  Given vectors form a basis of 4 dimensional space.
* You enter 2: statement (g) *
  Given vectors form a basis of 4 dimensional space.
* You enter 3: statement (h) *
  Given vectors form a basis of 4 dimensional space.
```

(b)-ii.

```
>> syms a b c d;
>> v1=[a;b;c;d]; v2=[-b;a;d;-c]; v3=[-c;-d;a;b]; v4 = [-d;c;-b;a];
>> CheckBasis(v1, v2, v3, v4, 1);
>> CheckBasis(v1, v2, v3, v4, 2);
```

MATLAB results.

```
* You enter 1: statement (a) *
  Given vectors form a basis of 4 dimensional space.
* You enter 2: statement (g) *
  Given vectors form a basis of 4 dimensional space.
```

7.3 The Fundamental Spaces of a Matrix

Exercise 7.3. In this problem, we make a function file `getFSinfo.m` to get the dimension and basis of the fundamental spaces of a given matrix. For example, we execute the followings:

```
>> A=[1 0 0 0 2; -2 1 -3 -2 -4; 0 5 -14 -9 0; 2 10 -28 -18 4];
>> getFSinfo(A);
```

Then, the Command Window displays the results as follows:

```
Given matrix is:
     1     0     0     0     2
    -2     1    -3    -2    -4
     0     5   -14    -9     0
     2    10   -28   -18     4

== Dimension of the fundamental spaces of a given matrix ==
dim(row(A))=dim(col(A)): 3,   dim(null(A)): 2,   dim(null(A_trans)): 1

== Basis of the fundamental spaces of a given matrix (in row vectors) ==
 row(A)
     1     0     0     0     2
     0     1     0     1     0
     0     0     1     1     0

 col(A)
```

```
     1     0     0     2
     0     1     0     0
     0     0     1     2

 null(A)
     0    -1    -1     1     0
    -2     0     0     0     1

 null(A_trans)
    -2     0    -2     1

**************************************************
```

(a) Complete the missing parts of the m-file `getFSinfo` given as follows:

```
%--- function file 'getFSinfo.m' ---%
function [info]=getFSinfo(A)
  % row(A): basis and dimension
  /////// missing part ///////

  % col(A): basis and dimension
  /////// missing part ///////

  % null(A): basis and dimension
  /////// missing part ///////

  % null(A'): basis and dimension
  /////// missing part ///////

  disp('Given matrix is:'); disp(A);
  fprintf('== Dimension of the fundamental spaces of given matrix == \n');
  fprintf('dim(row(A))=dim(col(A)): %d,', rank_A);
  fprintf('\t dim(null(A)): %d,\t dim(null(A_trans)): %d \n\n', nullity, nulli
  fprintf('== Basis of the fundamental spaces of given matrix (in row vectors)
  disp(' row(A)'); disp(double(rowA_basis));
  disp(' col(A)'); disp(double(colA_basis));
  disp(' null(A)'); disp(nullA_basis);
  disp(' null(A_trans)'); disp(nullAtrans_basis);
  fprintf('\n**************************************************\n');
end
```

You may use the MATLAB commands *rank, colspace, rref, null* and so on.

(b) Using your function file `getFSinfo.m`, find the dimension and basis of the fundamental spaces of

$$A=\begin{bmatrix}3&2&1&3&5\\6&4&3&5&7\\9&6&5&7&9\\3&2&0&4&8\end{bmatrix}, B=\begin{bmatrix}3&-1&3&2&5\\5&-3&2&3&4\\1&-3&-5&0&-7\\7&-5&1&4&1\end{bmatrix}, C=\begin{bmatrix}1&3&2&1\\-2&-6&0&-6\\3&9&1&8\\-1&-3&-3&-6\\1&3&2&1\\4&12&1&11\end{bmatrix}.$$

Solution.

(a)
```
% ----- function file 'getFSinfo.m' ----- %
function [info]=getFSinfo(A)
  [m,n]=size(A)
  % row(A): basis and dimension
    rank_A=rank(A);    % rank of A;
    rowA=colspace(sym(A')); % find the row basis
    rowA_basis=rowA(:, 1:rank_A)';  % basis of row(A)
  % col(A): basis and dimension
    colA=colspace(sym(A)); % find the column basis
    colA_basis=colA(:, 1:rank_A)';  % basis of col(A)
  % null(A): basis and dimension
    nullA=null(A, 'r');
    nullity=n-rank_A;   % using Dimension theorem
    nullA_basis=nullA(:, 1:nullity)';
  % null(A'): basis and dimension
    nullAtrans=null(A', 'r');
    nullity_T=m-rank_A;
    nullAtrans_basis=nullAtrans(:,1:nullity_T)';

  disp('Given matrix is:'); disp(A);
  fprintf('== Dimension of the fundamental spaces of given matrix == \n');
  fprintf('dim(row(A))=dim(col(A)): %d,', rank_A);
  fprintf('\t dim(null(A)): %d,', nullity);
  fprintf('\t dim(null(A_trans)): %d \n\n', nullity_T);
  fprintf('== Basis of the fundamental spaces ');
  fprintf('of given matrix (in row vectors) == \n');
  disp(' row(A)'); disp(double(rowA_basis));
  disp(' col(A)'); disp(double(colA_basis));
  disp(' null(A)'); disp(nullA_basis);
  disp(' null(A_trans)'); disp(nullAtrans_basis);
  fprintf('\n*******************************************************\n');
end
```

(b)
```
A=[3 2 1 3 5; 6 4 3 5 7; 9 6 5 7 9; 3 2 0 4 8];
B=[3 -1 3 2 5; 5 -3 2 3 4; 1 -3 -5 0 -7; 7 -5 1 4 1];
C=[1 3 2 1; -2 -6 0 -6 ;3 9 1 8; -1 -3 -3 -6; 1 3 2 1; 4 12 1 11];
getFSinfo(A);
getFSinfo(B);
getFSinfo(C);
```

MATLAB results.

```
Given matrix is:
     3     2     1     3     5
     6     4     3     5     7
     9     6     5     7     9
     3     2     0     4     8
== Dimension of the fundamental spaces of given matrix ==
dim(row(A))=dim(col(A)): 2, dim(null(A)): 3, dim(null(A_trans)): 2
== Basis of the fundamental spaces of given matrix (in row vectors) ==
 row(A)
    1.0000    0.6667         0    1.3333    2.6667
         0         0    1.0000   -1.0000   -3.0000
 col(A)
     1     0    -1     3
     0     1     2    -1
 null(A)
   -0.6667    1.0000         0         0         0
   -1.3333         0    1.0000    1.0000         0
   -2.6667         0    3.0000         0    1.0000
 null(A_trans)
     1    -2     1     0
    -3     1     0     1
******************************************************

Given matrix is:
     3    -1     3     2     5
     5    -3     2     3     4
     1    -3    -5     0    -7
     7    -5     1     4     1
== Dimension of the fundamental spaces of given matrix ==
dim(row(A))=dim(col(A)): 3, dim(null(A)): 2, dim(null(A_trans)): 1
== Basis of the fundamental spaces of given matrix (in row vectors) ==
 row(A)
    1.0000         0    1.7500    0.7500         0
         0    1.0000    2.2500    0.2500         0
         0         0         0         0    1.0000
 col(A)
     1     0    -3     0
     0     1     2     0
```

```
     0     0     0     1
 null(A)
   -1.7500   -2.2500    1.0000         0         0
   -0.7500   -0.2500         0    1.0000         0
 null(A_trans)
     3    -2     1     0
****************************************************

Given matrix is:
     1     3     2     1
    -2    -6     0    -6
     3     9     1     8
    -1    -3    -3    -6
     1     3     2     1
     4    12     1    11
== Dimension of the fundamental spaces of given matrix ==
dim(row(A))=dim(col(A)): 3, dim(null(A)): 1, dim(null(A_trans)): 3
== Basis of the fundamental spaces of given matrix (in row vectors) ==
 row(A)
     1     3     0     0
     0     0     1     0
     0     0     0     1
 col(A)
    1.0000         0    0.5000         0    1.0000    0.5000
         0    1.0000   -1.2500         0         0   -1.7500
         0         0         0    1.0000         0         0
 null(A)
    -3     1     0     0
 null(A_trans)
   -0.5000    1.2500    1.0000         0         0         0
   -1.0000         0         0         0    1.0000         0
   -0.5000    1.7500         0         0         0    1.0000
****************************************************
```

Exercise 7.4. (*Bases for the Fundamental Spaces*)

(a) Use the MATLAB commands *sym* and *colspace* to find a basis for the column space of the matrix

$$A = \begin{bmatrix} 2 & -1 & 3 & 5 \\ 4 & -3 & 1 & 3 \\ 3 & -2 & 3 & 4 \\ 4 & -1 & 15 & 17 \\ 7 & -6 & -7 & 0 \end{bmatrix}.$$

(b) Use the same MATLAB commands in (a) to find a basis for the row space of A.

(c) Confirm that the basis obtained in (b) is consistent with the basis obtained from the reduced row echelon form of A.

(d) Tell what happens if you use the MATLAB command *orth*?

Solution.

(a)
```
% Set a matrix A whose entries are symbolic objects.
A = sym([2 -1 3 5; 4 -3 1 3; 3 -2 3 4; 4 -1 15 17; 7 -6 -7 0]);

% Find a basis for the column space of A.
col_basis = colspace(A);

disp('A basis for the column space of A is');
disp(col_basis(:,1)'); disp(col_basis(:,2)'); disp(col_basis(:,3)');
```
MATLAB results.
```
A basis for the column space of A is
[ 1, 0, 0, 2, 1]

[ 0, 1, 0, -3, 5]

[ 0, 0, 1, 4, -5]
```

(b)
```
% Set a matrix A_transpose whose entries are symbolic objects.
A_transpose = sym([2 -1 3 5; 4 -3 1 3; 3 -2 3 4; 4 -1 15 17; 7 -6 -7 0]');

% Finding a basis for the row space of A is equivalent to
% finding a basis for the column space of A_transpose.
rowbasis = colspace(A_transpose);

disp('A basis for the row space of A is');
disp(rowbasis(:,1)'); disp(rowbasis(:,2)'); disp(rowbasis(:,3)');
```
MATLAB results.
```
A basis for the row space of A is
[ 1, 0, 0, 6]

[ 0, 1, 0, 7]

[ 0, 0, 1, 0]
```

(c)
```
% Set a matrix A.
A = [2 -1 3 5; 4 -3 1 3; 3 -2 3 4; 4 -1 15 17; 7 -6 -7 0];

% Find the reduced row echelon form of A.
```

```
rref_A = rref(A);

% The nonzero rows of the reduced row echelon form of A
% form a basis for the row space of A.

disp('A basis for the row space of A is');
disp(rref_A(1,:)); disp(rref_A(2,:)); disp(rref_A(3,:));
```

MATLAB results.

```
A basis for the row space of A is
     1     0     0     6

     0     1     0     7

     0     0     1     0
```

(d)
```
% Set A.
A = [2 -1 3 5; 4 -3 1 3; 3 -2 3 4; 4 -1 15 17; 7 -6 -7 0];

% The command orth gives an orthonormal basis for the column space of A.
B = orth(A);

disp('An orthonormal basis for the column space of A is');
disp('q1='); disp(B(:,1)');
disp('q2='); disp(B(:,2)');
disp('q3='); disp(B(:,3)');
```

MATLAB results.

```
An orthonormal basis for the column space of A is
q1=
   -0.2427   -0.1508   -0.2229   -0.9246    0.1177

q2=
   -0.1189   -0.3624   -0.2060    0.0253   -0.9008

q3=
    0.3760   -0.6016   -0.5930    0.1848    0.3331
```

7.4 The Dimension Theorem and Its Implications

Exercise 7.5. (*Rank and Nullity*)

(a) Use the MATLAB command *rank* and the Formula (2) in Section 7.4 to find the nullity of the matrix

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

(b) Confirm that the result obtained in (a) is consistent with the number of basis vectors which are obtained by using the MATLAB command *null*.

Solution.

(a)
```
% Set A.
A = [3 2 1 3 5; 6 4 3 5 7; 9 6 5 7 9; 3 2 0 4 8];

% Find the rank of A by using the command rank.
rank_A = rank(A);

% Size of the matrix A.
[ m n ] = size(A);
% m = the number of rows of A, n = the number of columns of A.

 % By (2) in section 7.4, rankA + nullA = n.
null_A = n - rank_A;

disp('The nullity of A is'); disp(null_A);
```

MATLAB results.
```
The nullity of A is
     3
```

(b)
```
% Set A.
A = [3 2 1 3 5; 6 4 3 5 7; 9 6 5 7 9; 3 2 0 4 8];

% Find a basis for the null space of A.
nullA = null(A,'r');
% null(A,'r') returns a matrix
% whose columns are a basis for the null space of A.

[m n] = size(nullA);
% Since the number of columns of nullA is n,
% thus, n = the number of basis vectors of the null space of A.

disp('The nullity of A is'); disp(n);
```

MATLAB results.
```
The nullity of A is
     3
```

7.5 The Rank Theorem and Its Implications

Exercise 7.6. Note that the rank of a nonzero matrix A is equal to the order of the largest square submatrix of A (formed by deleting rows and columns of A) whose determinant is nonzero. In this problem, we make a function file `CheckRank.m` to find the rank of the given matrix using this fact. We want to obtain the execution results as follows:

```
>> A=[1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16];
>> rankA=CheckRank(A)
rankA =
     2
```

For this, you may start with the largest square matrices to be found in A and a search is started for the first submatrix with a nonzero determinant. Use the MATLAB command *nchoosek* to select all the combinations of rows and columns needed in the search process and you may use the several MATLAB commands if you need. Complete the m-file below and check the determinant of the matrices A, B, and C given in the Exercise 7.3 (b). Also, compare the results using the MATLAB command *rank*.

Solution.

```
%--- function file 'CheckRank.m' ---%
function [rank_A]= CheckRank(A)

    [m,n]=size(A);  % size of given matrix
    flg=1;  % flag for while loop
    if m>n  % if (# of row) > (# of col)
        A=A';
    end
    A=sym(A);   % Set A as a symbolic object

    K = min(m,n);   N = max(m,n);   % k : row number, N: col number
    k=K;    % from the largest size of submatrix
    while flg == 1
        comb_row=nchoosek(1:K, k);  % combinations of row
        comb_col=nchoosek(1:N, k);  % combinations of columns
        for ii=1:size(comb_row) %
            selected_A=A(comb_row(ii,:),:); % selected row index
            for jj=1:size(comb_col)
                sub_A=selected_A(:,comb_col(jj, :));   % selected col index
                if det(sub_A)~=0    % if non zeros determinant appears
                    rank_A=k;   % the size at that time <- rank
                    flg=0;  % stop the while loop.
                end
            end
        end
        k=k-1;  % if all submatrices of size k have a zero determinant,
```

```
                % reduce the size of submatrix.
    end

end
```

To check the determinant of the matrices A, B, and C given in the Exercise 7.3, you execute the followings:

```
A=[3 2 1 3 5; 6 4 3 5 7; 9 6 5 7 9; 3 2 0 4 8];
B=[3 -1 3 2 5; 5 -3 2 3 4; 1 -3 -5 0 -7; 7 -5 1 4 1];
C=[1 3 2 1; -2 -6 0 -6 ;3 9 1 8; -1 -3 -3 -6; 1 3 2 1; 4 12 1 11];

fprintf('my rank(A): %.5f, MATLAB rank(A): %.5f \n', CheckRank(A), rank(A));
fprintf('my rank(B): %.5f, MATLAB rank(B): %.5f \n', CheckRank(B), rank(B));
fprintf('my rank(C): %.5f, MATLAB rank(C): %.5f \n', CheckRank(C), rank(C));
```

MATLAB results.

```
my rank(A): 2.00000, MATLAB rank(A): 2.00000
my rank(B): 3.00000, MATLAB rank(B): 3.00000
my rank(C): 3.00000, MATLAB rank(C): 3.00000
```

Those are the same results as given in Exercise7.3.

7.6 The Pivot Theorem and Its Implications

Exercise 7.7. (*Finding a Basis with the Pivot Theorem*)
Consider the vectors

$$\begin{aligned}
\mathbf{v}_1 &= (1,\ 2,\ 4,\ -6,\ 11,\ 23,\ -14,\ 0,\ 2,\ 2),\\
\mathbf{v}_2 &= (3,\ 1,\ -1,\ 7,\ 9,\ 13,\ -12,\ 8,\ 6,\ -30),\\
\mathbf{v}_3 &= (5,\ 5,\ 7,\ -5,\ 31,\ 59,\ -40,\ 8,\ 10,\ -26),\\
\mathbf{v}_4 &= (5,\ 0,\ -6,\ 20,\ 7,\ 3,\ -10,\ 16,\ 10,\ -62).
\end{aligned}$$

Use Algorithm 1 in Section 7.6 to find a subset of these vectors that forms a basis for span$\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4\}$, and express those vectors not in the basis as linear combinations of basis vectors.

Solution.

```
v1 = [1 2 4 -6 11 23 -14 0 2 2]';
v2 = [3 1 -1 7 9 13 -12 8 6 -30]';
v3 = [5 5 7 -5 31 59 -40 8 10 -26]';
v4 = [5 0 -6 20 7 3 -10 16 10 -62]';

% Construct A whose column space is W=span(v1,v2,v3,v4).
A = [v1 v2 v3 v4];

% Find the reduced row echelon form R of A and the pivot columns of A.
```

```
[R, pivotcols] = rref(A);

format short;

disp('The pivot columns of the reduced row echelon form of A are');
disp(pivotcols);

% From the result, the leading 1's in R occur in columns 1 and 2.
% (i.e., the pivot columns of A are 1 and 2.)
% Hence, the basis vectors for W are v1 and v2.

disp('The reduced row echelon form R of A is'); disp(R);

% Furthermore, from the reduced row echelon form R of A,
% we can see that v3 = 2*v1 + v2, and v4 = -v1 + 2*v2.
```

MATLAB results.

```
The pivot columns of the reduced row echelon form of A are
     1     2

The reduced row echelon form R of A is
     1     0     2    -1
     0     1     1     2
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

Exercise 7.8. (*Finding Bases for the Fundamental Spaces*)
Consider the matrix

$$A = \begin{bmatrix} 1 & 3 & 2 & 1 \\ -2 & -6 & 0 & -6 \\ 3 & 9 & 1 & 8 \\ -1 & -3 & -3 & -6 \\ 1 & 3 & 2 & 1 \\ 4 & 12 & 1 & 11 \end{bmatrix}.$$

(a) Use Algorithm 1 in Section 7.6 to find a subset of the column vectors of A that forms a basis for the column space of A, and express each column vector of A that is not in that basis as a linear combination of the basis vectors.

(b) Use Algorithm 2 in Section 7.6 to find a basis for the null space of the matrix A^T.

Solution.

```
A = [1 3 2 1; -2 -6 0 -6; 3 9 1 8; -1 -3 -3 -6; 1 3 2 1; 4 12 1 11];

% Find the reduced row echelon form R of A and the pivot columns of A.
[R, pivotcols] = rref(A);

format short;

disp('The pivot columns of the reduced row echelon form of A are');
disp(pivotcols);
% From the result, the leading 1's in R occur in columns 1, 3, and 4.
% (i.e., the pivot columns of A are 1, 3, and 4.)
% Hence, the columns 1, 3, and 4 of A are a basis for the column space of A.

disp('The reduced row echelon form R of A is'); disp(R);
% Furthermore, from the reduced row echelon form R of A,
% we can see that v2 = 3*v1, where v1 = A(:, 1), and v2 = A(:, 2).
```

MATLAB results.

```
The pivot columns of the reduced row echelon form of A are
     1     3     4

The reduced row echelon form R of A is
     1     3     0     0
     0     0     1     0
     0     0     0     1
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

7.7 The Projection Theorem and Its Implications

Exercise 7.9. (*Standard Matrix for an Orthogonal Projection*)
One way to find the standard matrix for the orthogonal projection onto a subspace W spanned by a set of vectors $\{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_k\}$ is first to find a basis for W, then create a matrix A that has the basis vectors as columns, and then use the Formula (27) in the Section 7.7.

(a) Find the standard matrix for the orthogonal projection of $\mathbb{R}^4$ onto the subspace W spanned by

$$\mathbf{v}_1 = (1,\ 2,\ 3,\ -4), \qquad \mathbf{v}_2 = (2,\ 3,\ -4,\ 1),$$
$$\mathbf{v}_3 = (2,\ -5,\ 8,\ -3), \qquad \mathbf{v}_4 = (5,\ 26,\ -9,\ -12),$$
$$\mathbf{v}_5 = (3,\ -4,\ 1,\ 2).$$

(b) Use the matrix obtained in part (a) to find $\text{proj}_W \mathbf{x}$, where $\mathbf{x} = (1,\ 0,\ -3,\ 7)$.

(c) Find $\text{proj}_{W^\perp} \mathbf{x}$ for the vector in part (b).

Solution.

```
v1 = [1 2 3 -4]'; v2 = [2 3 -4 1]'; v3 = [2 -5 8 -3]';
v4 = [5 26 -9 -12]'; v5 = [3 -4 1 2]';

% Set A that has v1,v2,v3,v4 and v5, as column vectors.
A = [v1 v2 v3 v4 v5];

% Find the reduced row echelon form R of A and the pivot columns of A.
[R, pivotcols] = rref(A);

% M is the matrix whose columns are a basis for the column space of A.
M = A(:, pivotcols);

% By (27) in section 7.7, find the standard matrix.
P = M * inv(M'* M) * M';

format short;
disp('The standard matrix for the orthogonal projection of R^4 onto W=col(A) is');
disp(P);

x = [1 0 -3 7]';
xproj = P*x;
xperp = x - xproj;
disp('The projection of x onto W=col(A) is'); disp(xproj');

disp('The projection of x onto the orthogonal complement of W=col(A) is');
disp(xperp');

% As a check, the dot product of the two projections should be zero.
disp('The dot product of the two projections is'); disp(dot(xproj, xperp));
```

MATLAB results.

```
The standard matrix for the orthogonal projection of R^4 onto W=col(A) is
    0.9992   -0.0144   -0.0161   -0.0195
   -0.0144    0.7551   -0.2737   -0.3314
   -0.0161   -0.2737    0.6941   -0.3703
```

```
   -0.0195   -0.3314   -0.3703    0.5517

The projection of x onto W=col(A) is
    0.9110   -1.5127   -4.6907    4.9534

The projection of x onto the orthogonal complement of W=col(A) is
    0.0890    1.5127    1.6907    2.0466

The dot product of the two projections is
 -5.3291e-015
```

7.8 Best Approximation and Least Squares

Exercise 7.10. Make a function file `LinearSolver.m` to find a least squares solution of $A\mathbf{x} = \mathbf{b}$ where A has full column rank. Complete the missing part referring to the comments. Using this function file, solve the linear system

$$\begin{cases} x - y = 4 \\ 3x + 2y = 1 \\ -2x + 4y = 3 \end{cases}$$

and compare the output with the result of the MATLAB syntax $A\backslash\mathbf{b}$.

Solution.

```
%--- This is a function file 'LinearSolver.m' ---%
function [rank_A sol]=LinearSolver(A, b)
  [m,n]=size(A);
  rank_A=rank(A);

  % Check that A has full column rank.
  if rank_A<n
    fprintf('rank(A)=%d < %d  -> Not full column rank\n', rank_A, n);
    return; % If A does not have full column rank, then return.
  else
    fprintf('rank(A)=%d = %d -> Full column rank\n', rank_A, n);
  end

  % From the reduced row echelon form of [A'*A |A'*b],
  % find a solution to the normal equation A'Ax=A'b.

  Aug=[A'*A A'*b];
  rref_Aug=rref(Aug);
  sol=rref_Aug(:,n+1);

  fprintf('The least squares solution is');disp(sol');
end
```

You execute the followings:

```
A=[1 -1; 3 2; -2 4];
b=[4; 1; 3];
LinearSolver(A, b);
A\b
```

MATLAB results.

```
rank(A)=2 = 2 -> Full column rank
The least squares solution is    0.1789    0.5018
ans =
    0.1789
    0.5018
```

Exercise 7.11. The least squares method can be used to estimate the center (h, k) of a circle $(x-h)^2 + (y-k)^2 = r^2$ using measured data points on its circumference. Suppose that the data points are

$$(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$$

and rewrite the equation of the circle in the form

$$2xh + 2yk + s = x^2 + y^2 \tag{7.1}$$

where

$$s = r^2 - h^2 - k^2 \tag{7.2}$$

Substituting the data points in (7.1) yields a linear system in the unknowns h, k, and s, which can be solved by least squares to estimate their values. Equation (7.2) can then be used to estimate r. Use this method to approximate the center and radius of a circle from the measured data points on the circumference given in the accompanying table.

Table 7.1: Data points of Problem 7(b)

x	19.880	20.919	21.735	23.375	24.361	25.375	25.979
y	68.874	67.676	66.692	64.385	62.908	61.292	60.277

Graph the circle you obtained and plot the data points with red circles in the same figure.

Solution. You execute the followings:

```
format short;

% given data
x=[19.880 20.919 21.735 23.375 24.361 25.375 25.979];
y=[68.874 67.676 66.692 64.385 62.908 61.292 60.277];
```

```
% number of data points.
[m,n]=size(x);

% construct the system matrix
A=[2*x' 2*y' ones(n,1)]; b=x.^2+y.^2;

% solve the normal equation
hks=inv(A'*A)*A'*b'
h=hks(1); k=hks(2); s=hks(3);

% compute the radius
r=sqrt(s+h^2+k^2);

figure;
theta=0:0.01:2*pi;
xx=h+r*cos(theta);
yy=k+r*sin(theta);

% plot the obtained circle
plot(xx,yy);
hold on;

% plot the data points
plot(x, y, 'o');
```

MATLAB results.

```
hks =
  -18.3534
   35.4513
  986.5129
```

7.9 Orthonormal Bases and the Gram Schmidt Process

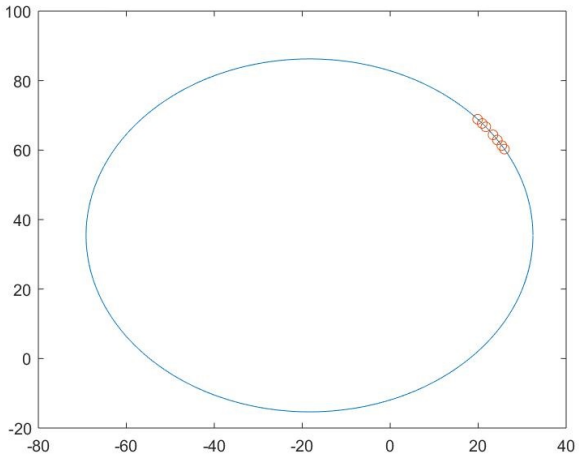
Exercise 7.12. (*Gram-Schmidt Process*)
Perform the Gram-Schmidt process to transform the vectors given in the Example 9 of the Section 7.9 to obtain an orthonormal basis for $\mathbb{R}^3$.

In this problem, use a nested loop and the MATLAB command *norm*.

Solution.

```
w1 = [1 1 1]'; w2 = [0 1 1]'; w3 = [0 0 1]';
A = [w1 w2 w3]; % Construct a matrix A whose columns are w1, w2, and w3.
format short; [m, n] = size(A);
Q = zeros(m, n); % Initialize the matrix Q as an m*n zero matrix.

% Find an orthonormal basis for the column space of A.
```




```
for j = 1 : n
    v = A(:, j); % v begins as jth column of A.
    for i = 1 : (j-1)
        temp = Q(:, i)' * A(:, j);
        % Subtract each component of orthogonal projection of v
        % onto the subspace spanned by the vector Q(:, i).
        v = v - temp * Q(:, i);
    end
    Q(:, j) = v / norm(v); % Normalize v by its 2-norm.
end
disp('The orthonormal basis {q1,q2,q3} for R^3 from {w1,w2,w3} are as follows:')
disp('q1='); disp(Q(:,1)'); disp('q2='); disp(Q(:,2)'); disp('q3='); disp(Q(:,3)');
```

MATLAB results.

```
The orthonormal basis {q1,q2,q3} for R^3 from {w1,w2,w3} are as follows:
q1=
    0.5774    0.5774    0.5774

q2=
   -0.8165    0.4082    0.4082

q3=
   -0.0000   -0.7071    0.7071
```

Exercise 7.13. (*Orthonormal Bases for the Four Fundamental Spaces*)
Find orthonormal bases for the four fundamental spaces of the matrix

$$A = \begin{bmatrix} 2 & -1 & 3 & 5 \\ 4 & -3 & 1 & 3 \\ 3 & -2 & 3 & 4 \\ 4 & -1 & 15 & 17 \\ 7 & -6 & -7 & 0 \end{bmatrix}.$$

Solution.

```
%--- The following is the function file 'GramSchmidt.m'. ---%

% Find an orthonormal basis for col(A) when A has full column rank.
function Q = GramSchmidt(A)

  [m, n] = size(A);

  % Initialize the matrix Q as an m*n zero matrix.
  Q = zeros(m, n);

  for j = 1 : n
    % v begins as jth column of A.
    v = A(:, j);
    for i = 1 : (j-1)
      temp = Q(:, i)' * A(:, j);
      % Subtract each component of orthogonal projection of v
      % onto the subspace spanned by the vector Q(:, i).
      v = v - temp * Q(:, i);
    end
    Q(:, j) = v / norm(v); % Normalize v by its 2-norm.
  end
end
% Q is an m*n matrix whose columns form an orthonormal basis for col(A).
```

The following commands are performed in the command window of MATLAB.

```
A = [2 -1 3 5; 4 -3 1 3; 3 -2 3 4; 4 -1 15 17; 7 -6 -7 0];
format short;

% Find the reduced row echelon form of A.
rref_A = rref(A);

% (1). Find an orthonormal basis for the row space of A.
% From the result of rref_A, the first three nonzero rows in rref_A form
% a basis for the row space of A.

% Construct a matrix R_A whose columns are a basis for the row space of A.
R_A = rref_A(1:3, :)';
```

```
% Find an orthonormal basis for the column space of R_A by Gram-Schmidt process,
% which is the same as finding an orthonormal basis for the row space of A.
Orth_R_A = GramSchmidt(R_A);

a1 = Orth_R_A(:, 1); a2 = Orth_R_A(:, 2); a3 = Orth_R_A(:, 3);

disp('An orthonormal basis {a1, a2, a3} for the row space of A is');
disp('a1 = '); disp(a1'); disp('a2 = '); disp(a2'); disp('a3 = '); disp(a3');

% (2). Find an orthonormal basis for the column space of A.
% From the result of rref_A, the first three columns of A are the pivot columns
% which form a basis for the column space of A.

% Construct a matrix C_A whose columns are a basis for the column space of A.
C_A = A(:, 1:3);

% Find an orthonormal basis for the column space of C_A by Gram-Schmidt process,
% which is the same as finding an orthonormal basis for the column space of A.
Orth_C_A = GramSchmidt(C_A);

b1 = Orth_C_A(:, 1); b2 = Orth_C_A(:, 2); b3 = Orth_C_A(:, 3);

disp('An orthonormal basis {b1, b2, b3} for the column space of A is');
disp('b1 = '); disp(b1'); disp('b2 = '); disp(b2'); disp('b3 = '); disp(b3');


% (3). Find an orthonormal basis for the null space of A.
% In addition, from the result of rref_A,
% we can easily see that {[-6 -7 0 1]'} is a basis for N(A).

% Construct a matrix N_A whose columns are a basis for the null space of A.
N_A = [-6 -7 0 1]';

% Find an orthonormal basis for the column space of N_A by Gram-Schmidt process,
% which is the same as finding an orthonormal basis for the null space of A.
Orth_N_A = GramSchmidt(N_A);

c1 = Orth_N_A(:, 1);
disp('An orthonormal basis {c1} for the null space of A is');
disp('c1 = '); disp(c1');

% (4). Find an orthonormal basis for the null space of A transpose.
[L U P] = lu(A);
temp = [0 0 0 0 1]';
\% Make L a square matrix of order 5.
L = [L temp];
```

```
% Make U have the same size of A.
U(5, :) = 0;

% Then, we have P*A = L*U, which is the same result as above.
% Note that L^(-1)*P*A = U, where U is an upper triangular matrix.

E = L^(-1)*P;

% Since E = L^(-1)*P is a product of elementary matrices s.t. E*A=U,
% E represents a set of elementary row operations
% that makes A become a row echelon form U.

% ref_par_A is the resulting partitioned matrix [U E].
ref_par_A = [U E];

% From the result of ref_par_A, we can see that ref_par_A([4:5], [1:4]) = 0.
% Thus, the row vectors of E2 form a basis for null(A'),
% where E2 = ref_par_A([4:5], [5:9]).

% Construct a matrix N_Atrans whose columns are a basis for
% the null space of A transpose.
N_Atrans = ref_par_A(4:5, 5:9)';

% Find an orthonormal basis for the column space of N_Atrans by Gram-Schmidt proces
% which is the same as finding an orthonormal basis for the null space of A transpo
Orth_N_Atrans = GramSchmidt(N_Atrans);

d1 = Orth_N_Atrans(:, 1); d2 = Orth_N_Atrans(:, 2);
disp('An orthonormal basis {d1, d2} for the null space of the transpose of A is');
disp('d1 = '); disp(d1'); disp('d2 = '); disp(d2');
```

MATLAB results.

```
An orthonormal basis {a1, a2, a3} for the row space of A is
a1 =
    0.1644         0         0    0.9864

a2 =
   -0.7446    0.6559         0    0.1241

a3 =
     0     0     1     0

An orthonormal basis {b1, b2, b3} for the column space of A is
b1 =
    0.2063    0.4126    0.3094    0.4126    0.7220
```

```
b2 =
    0.1873   -0.0887    0.0493    0.8378   -0.5027

b3 =
   -0.3699    0.5812    0.5878   -0.1321   -0.4029

An orthonormal basis {c1} for the null space of A is
c1 =
   -0.6470   -0.7548         0    0.1078

An orthonormal basis {d1, d2} for the null space of the transpose of A is
d1 =
    0.8649    0.3089         0   -0.3089   -0.2471

d2 =
    0.1936   -0.6234    0.7458   -0.1224    0.0512
```

7.10 $QR-$Decomposition; Householder Transformations

Exercise 7.14. (*$QR-$Decomposition*)

(a) Make a function file `myQR.m` to find a QR-decomposition of a given matrix. You may use your function file `GS_process.m` from the **Exercise** 7.13.

(b)
$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 2 \\ 0 & 1 & 2 \end{bmatrix}.$$
Compare your result with the output produced by the MATLAB command *qr*.

Solution.

```
%(a)
%--- This is a function file myQR.m ---%

function [Q R]=myQR(A)
 Q=GS_process(A);
 R=Q'*A;
end

%(b)
A=[1 1 1; 1 0 2; 0 1 2];
```

```
[Q1 R1]=myQR(A); [Q R]=qr(A);

disp('my QR result'); disp('Q');disp(Q1); disp('R');disp(R1);
disp('MATLAB QR result'); disp('Q');disp(Q); disp('R');disp(R);
```

MATLAB results.

```
my QR result
Q
    0.7071    0.4082   -0.5774
    0.7071   -0.4082    0.5774
         0    0.8165    0.5774
R
    1.4142    0.7071    2.1213
    0.0000    1.2247    1.2247
    0.0000   -0.0000    1.7321

MATLAB QR result
Q
   -0.7071    0.4082   -0.5774
   -0.7071   -0.4082    0.5774
         0    0.8165    0.5774
R
   -1.4142   -0.7071   -2.1213
         0    1.2247    1.2247
         0         0    1.7321
```

The results are the same.

7.11 Coordinates with Respect to a Basis

Exercise 7.15. (*Transition Matrices between Two Different Bases*)

(a) Confirm that $B_1 = \{\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4, \mathbf{u}_5\}$ and $B_2 = \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_5\}$ are bases for $\mathbb{R}^5$, and find the transition matrices $P_{B_1 \to B_2}$ and $P_{B_2 \to B_1}$, where

$$\begin{aligned} \mathbf{u}_1 &= (3,\ 1,\ 3,\ 2,\ 6) & \mathbf{v}_1 &= (2,\ 6,\ 3,\ 4,\ 2) \\ \mathbf{u}_2 &= (4,\ 5,\ 7,\ 2,\ 4) & \mathbf{v}_2 &= (3,\ 1,\ 5,\ 8,\ 3) \\ \mathbf{u}_3 &= (3,\ 2,\ 1,\ 5,\ 4) & \mathbf{v}_3 &= (5,\ 1,\ 2,\ 6,\ 7) \\ \mathbf{u}_4 &= (2,\ 9,\ 1,\ 4,\ 4) & \mathbf{v}_4 &= (8,\ 4,\ 3,\ 2,\ 6) \\ \mathbf{u}_5 &= (3,\ 3,\ 6,\ 6,\ 7) & \mathbf{v}_5 &= (5,\ 5,\ 6,\ 3,\ 4) \end{aligned}$$

(b) Find the coordinate matrices with respect to B_1 and B_2 of $\mathbf{w} = (1,\ 1,\ 1,\ 1,\ 1)$.

Solution.

```
u1 = [3 1 3 2 6]'; v1 = [2 6 3 4 2]';
```

```
u2 = [4 5 7 2 4]'; v2 = [3 1 5 8 3]';
u3 = [3 2 1 5 4]'; v3 = [5 1 2 6 7]';
u4 = [2 9 1 4 4]'; v4 = [8 4 3 2 6]';
u5 = [3 3 6 6 7]'; v5 = [5 5 6 3 4]';

U = [u1 u2 u3 u4 u5];
V = [v1 v2 v3 v4 v5];

format short;

% Initialization.
P_B1B2 = zeros(5);
P_B2B1 = zeros(5);

for j = 1:5
  % Find the coordinate vector of U(:, j) in B1 with respect to B2.
  P_B1B2(:, j) = V\U(:, j);
  % Find the coordinate vector of V(:, j) in B2 with respect to B1.
  P_B2B1(:, j) = U\V(:, j);
end

disp('The transition matrix from B1 to B2 is'); disp(P_B1B2);
disp('The transition matrix from B2 to B1 is'); disp(P_B2B1);

w = [1 1 1 1 1]';

% Find the coordinate matrix of w with respect to B1.
w_B1 = U\w;

% Find the coordinate matrix of w with respect to B2.
w_B2 = P_B1B2 * w_B1;

disp('The coordinate matrix of w with respect to B1 is'); disp(w_B1');
disp('The coordinate matrix of w with respect to B2 is'); disp(w_B2');
```

MATLAB results.

```
The transition matrix from B1 to B2 is
   -0.4992   -0.2531    0.4843    1.8286   -0.2123
   -0.7830   -0.3679    0.1604   -0.8019   -0.5849
    1.3019    0.2925    0.4623    0.6887    1.4906
   -0.9096   -0.6116    0.1918   -0.2091   -1.4104
    1.4230    1.8082   -0.4591   -0.2044    1.8019

The transition matrix from B2 to B1 is
   -0.6889   -1.3556    0.6222    1.2667   -0.0444
    0.4067    0.3591    0.0278    1.2083    1.0873
```

```
    0.3151    1.2675    1.3444    1.6833    0.6540
    0.3615   -0.5433   -0.2056   -0.1417   -0.0746
    0.2571    0.9714   -0.2000   -1.8000   -0.3429

The coordinate matrix of w with respect to B1 is
   -0.0222    0.1508    0.1841   -0.0016   -0.0286

The coordinate matrix of w with respect to B2 is
    0.0653    0.0094    0.0566    0.0039    0.1053
```

Chapter 8

Diagonalization

8.1 Matrix Representations of Linear Transformations

Exercise 8.1. Let $T : \mathbb{R}^5 \to \mathbb{R}^3$ be the linear operator given by the formula

$$T(x_1, x_2, x_3,\ x_4, x_5) = (7x_1+12x_2-5x_3,\ 3x_1+10x_2+13x_4+x_5,\ -9x_1-x_3-3x_5)$$

and let $B = \{\mathbf{v}_1,\ \mathbf{v}_2,\ \mathbf{v}_3,\ \mathbf{v}_4,\ \mathbf{v}_5\}$ and $B' = \{\mathbf{v}'_1,\ \mathbf{v}'_2,\ \mathbf{v}'_3\}$ be the bases for $\mathbb{R}^5$ and $\mathbb{R}^3$, respectively, in which $\mathbf{v}_1 = (1,\ 1,\ 0,\ 0,\ 0)$, $\mathbf{v}_2 = (0,\ 1,\ 1,\ 0,\ 0)$, $\mathbf{v}_3 = (0,\ 0,\ 1,\ 1,\ 0)$, $\mathbf{v}_4 = (0,\ 0,\ 0,\ 1,\ 1)$, $\mathbf{v}_5 = (1,\ 0,\ 0,\ 0,\ 1)$, $\mathbf{v}'_1 = (1,\ 2,\ -1)$, $\mathbf{v}'_2 = (2,\ 1,\ 3)$, and $\mathbf{v}'_3 = (1,\ 1,\ 1)$.

(a) Find the matrix $[T]_{B',B}$.

(b) For the vector $\mathbf{x} = (3,\ 7,\ -4,\ 5,\ 1)$, find $[\mathbf{x}]_B$ and use the matrix obtained in part (a) to compute $[T(\mathbf{x})]_{B'}$.

(c) Find the factorization of $[T]$ which is the standard matrix for the linear transformation T using Formula (28) in Section 8.1.

Solution.

(a)
```
v1 = [1 1 0 0 0]'; v2 = [0 1 1 0 0]'; v3 = [0 0 1 1 0]';
v4 = [0 0 0 1 1]'; v5 = [1 0 0 0 1]';
nv1 = [1 2 -1]'; nv2 = [2 1 3]'; nv3 = [1 1 1]';

T = [7 12 -5 0 0; 3 10 0 13 1; -9 0 -1 0 -3];
B1 = [v1 v2 v3 v4 v5]; B2 = [nv1 nv2 nv3];
format short;
```

```
% Find the matrix representation with respect to the bases B1 and B2.
TB = T*B1;
TB1B2 = B2\TB;

disp('The matrix representation of T with respect to the basis B1 and B2 is');
disp(TB1B2);
```

MATLAB results.

```
The matrix representation of T with respect to the basis B1 and B2 is
   34.0000    5.0000  -22.0000  -11.0000   22.0000
   40.0000    2.0000  -40.0000  -25.0000   25.0000
  -95.0000   -2.0000   97.0000   61.0000  -65.0000
```

(b)
```
% Find the coordinate vector of x with respect to the basis B1.
x = [3 7 -4 5 1]';  x_B1 = B1\x;
disp('The coordinate vector of x with respect to the basis B is');
disp(x_B1');

% Find the coordinate vector of T(x) with respect to the basis B2.
Tx_B2 = TB1B2 * x_B1;
disp('The coordinate vector of T(x) with respect to the basis B'' is');
disp(Tx_B2');
```

MATLAB results.

```
The coordinate vector of x with respect to the basis B is
     9    -2    -2     7    -6

The coordinate vector of T(x) with respect to the basis B' is
  131.0000  111.0000 -228.0000
```

(c)
```
% Transition matrix from B to the standard basis for R^n.
U=B1;

% Transition matrix from B' to the standard basis for R^m.
V=B2;

T=[7 12 -5 0 0 ; 3 10 0 13 1; -9 0 -1 0 -3];

disp('V'); disp(V);
disp('TB1B2'); disp(TB1B2);
disp('inv(U)'); disp(inv(U));
disp('V*TB1B2*inv(U)');disp(V*TB1B2*inv(U));
disp('T'); disp(T);
```

MATLAB results.

```
V
     1     2     1
     2     1     1
    -1     3     1
```

```
TB1B2
   34.0000    5.0000  -22.0000  -11.0000   22.0000
   40.0000    2.0000  -40.0000  -25.0000   25.0000
  -95.0000   -2.0000   97.0000   61.0000  -65.0000

inv(U)
    0.5000    0.5000   -0.5000    0.5000   -0.5000
   -0.5000    0.5000    0.5000   -0.5000    0.5000
    0.5000   -0.5000    0.5000    0.5000   -0.5000
   -0.5000    0.5000   -0.5000    0.5000    0.5000
    0.5000   -0.5000    0.5000   -0.5000    0.5000

V*TB1B2*inv(U)
    7.0000   12.0000   -5.0000         0         0
    3.0000   10.0000         0   13.0000    1.0000
   -9.0000   -0.0000   -1.0000   -0.0000   -3.0000

T
     7    12    -5     0     0
     3    10     0    13     1
    -9     0    -1     0    -3
```

8.2 Similarity and Diagonalizability

Exercise 8.2. (a) Show that the matrix

$$A = \begin{bmatrix} -13 & -60 & -60 \\ 10 & 42 & 40 \\ -5 & -20 & -18 \end{bmatrix}$$

is diagonalizable by finding the nullity of $\lambda I - A$ for each eigenvalue λ with the use of Theorem 8.2.11 in the Section 8.2.

(b) Find a basis for $\mathbb{R}^3$ consisting of eigenvectors of A.

Solution.

(a)
```
% For the exact computation of the eigenvalues,
% we use symbolic computation.

% Set A as a symbolic matrix.
A = sym([-13 -60 -60; 10 42 40; -5 -20 -18]);

n = length(A);

% Find the eigenvalues of A by using the command eig.
```

```
eigenvalues = eig(A);

for j = 1 : n
    fprintf('The eigenvalue lambda is '); disp(eigenvalues(j));

    % nullity(lambda*I - A) = n - rank(lambda*I - A);
    nullity = n - rank((eigenvalues(j) * eye(n)) - A);

    fprintf('The nullity of (lambda*I - A) is '); disp(nullity);
end

% Since the geometric multiplicity of each eigenvalue of A
% is the same as the algebraic multiplicity,
% by the Theorem 8.2.11, A is diagonalizable.
```

MATLAB results.

```
The eigenvalue lambda is 2
The nullity of (lambda*I - A) is      2
The eigenvalue lambda is 2
The nullity of (lambda*I - A) is      2
The eigenvalue lambda is 7
The nullity of (lambda*I - A) is      1
```

(b)

```
% Since the eigenvalue = 2 of A has the multiplicity = 2,
% find two linearly independent eigenvectors of A corresponding to lambda = 2.

%Find a basis for the null space of (2*I-A).
eigvec12=null((2 * eye(n)) - A);

% Since the eigenvalue = 7 of A has the multiplicity = 1,
% find an eigenvector of A corresponding to lambda = 7.

%Find a basis for the null space of (7*I-A).
eigvec3=null((7 * eye(n)) - A);

p1 = eigvec12(:, 1); p2 = eigvec12(:, 2); p3 = eigvec3(:, 1);

% By the Theorem 8.2.7, since the eigenvectors corresponding to
% distinct eigenvalues are linearly independent,
% the three obtained eigenvectors {p1, p2, p3} form a basis for R^{3}.

disp('A basis {p1, p2, p3} for R^{3} consisting of the eigenvectors of A is');
fprintf('p1 ='); disp(p1');
fprintf('p2 ='); disp(p2');
fprintf('p3 ='); disp(p3');
```

MATLAB results.

```
A basis {p1, p2, p3} for R^{3} consisting of the eigenvectors of A is
```

```
p1 =[ -4, 1, 0]
p2 =[ -4, 0, 1]
p3 =[ 3, -2, 1]
```

8.3 Orthogonal Diagonalizability; Functions of a Matrix

Exercise 8.3. Let

$$A = \begin{bmatrix} \frac{1}{2} & 0 & \frac{3}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{3}{2} \\ \frac{3}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{3}{2} & 0 & \frac{1}{2} \end{bmatrix}.$$

(a) Find a matrix P that orthogonally diagonalizes the matrix A. You may use the MATLAB command *eig* and perform the Gram-Schmidt process. Use your result to find a diagonal matrix D satisfying $A = PDP^T$.

(b) Confirm that the matrix A satisfies its characteristic equation, in accordance with the Cayley-Hamilton theorem. You may use the symbolic object to find the characteristic polynomial and use the MATLAB command *coeffs* to find the coefficient of obtained characteristic polynomial.

(c) Find the spectral decomposition of A.

Solution.

(a)
```
A=[1/2 0 3/2 0; 0 1/2 0 3/2; 3/2 0 1/2 0; 0 3/2 0 1/2];

% V: eigen vector, D: eigen value
[V D]=eig(A);

% Gram-Schmidt process
P=GS_process(V);
disp('P is'); disp(P);
disp('D is'); disp(D);
disp('P_transpose is'); disp(P');
disp('P*D*P_transpose is'); disp(P*D*P');
disp('A is'); disp(A);
```

MATLAB results.

```
P is
   -0.7071         0         0   -0.7071
         0    0.7071    0.7071         0
    0.7071         0         0   -0.7071
         0   -0.7071    0.7071         0
```

```
D is
    -1     0     0     0
     0    -1     0     0
     0     0     2     0
     0     0     0     2

P_transpose is
   -0.7071         0    0.7071         0
         0    0.7071         0   -0.7071
         0    0.7071         0    0.7071
   -0.7071         0   -0.7071         0

P*D*P_transpose is
    0.5000         0    1.5000         0
         0    0.5000         0    1.5000
    1.5000         0    0.5000         0
         0    1.5000         0    0.5000

A is
    0.5000         0    1.5000         0
         0    0.5000         0    1.5000
    1.5000         0    0.5000         0
         0    1.5000         0    0.5000
```

(b)
```
% Symbolic variable lambda
syms lambda;

% Characteristic polynomial
char_poly=det(lambda*eye(size(A))-A);

% Expand the characteristic polynomial cf. simplify
polynomial=expand(char_poly);

% Coefficients extraction
coeff=coeffs(polynomial);

% According to the descending order of lambda degree
coefficient=coeff(end:-1:1);

 % Compute the matrix polynomial
poly_A=polyvalm(double(coefficient), A);

disp('Coefficients of the matrix characteristic polynomial is');
disp(double(coefficient));
disp('Matrix characteristic polynomial is'); disp(poly_A);
```

MATLAB results.

```
Coefficients of the matrix characteristic polynomial is
     1    -2    -3     4     4

Matrix characteristic polynomial is
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

(c)
```
[V D]=eig(A);
sum_A=0;
for i=1:size(A,1);
  % spectral decomposition
  sum_A=sum_A+D(i,i)*V(:,i)*V(:,i)';

  fprintf('lambda_%d is %f \n', i, D(i,i));
  fprintf('corresponding u_%d is \n', i);
  disp(V(:,i));
end

disp('spectral decomposition of A is'); disp(sum_A);
disp('A is'); disp(A);
```

MATLAB results.
```
lambda_1 is -1.000000
corresponding u_1 is
   -0.7071
         0
    0.7071
         0

lambda_2 is -1.000000
corresponding u_2 is
         0
    0.7071
         0
   -0.7071

lambda_3 is 2.000000
corresponding u_3 is
         0
    0.7071
         0
    0.7071

lambda_4 is 2.000000
corresponding u_4 is
```

```
   -0.7071
         0
   -0.7071
         0

spectral decomposition of A is
    0.5000         0    1.5000         0
         0    0.5000         0    1.5000
    1.5000         0    0.5000         0
         0    1.5000         0    0.5000

A is
    0.5000         0    1.5000         0
         0    0.5000         0    1.5000
    1.5000         0    0.5000         0
         0    1.5000         0    0.5000
```

8.4 Quadratic Forms

Exercise 8.4. (*Cholesky Factorization*)
In this problem, we find a Cholesky factorization of the Hilbert matrix

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

To generate the Hilbert matrix, you may use the MATLAB command *hilb*.

(a) Show that A is positive definite symmetric matrix by finding its eigenvalues and the MATLAB command *issymmetric*.

(b) Make a function file `ludecomp.m` to find the LU-decomposition of an invertible $n \times n$ matrix A such that A can be reduced to row echelon form by Gaussian elimination without row interchanges. You may refer to the four steps given in Page 157. Check your result by applying this function for the matrix given in the Example 2 of the Section 3.7.

(c) Referring to the Section 3.7, find the LDU-factorization of A from an LU-factorization of A by `ludecomp.m`.

(d) From the LDU-factorization of A obtained in (c), find a Cholesky factorization $A = R^T R$, where R is upper triangular.

(e) Use the MATLAB command *chol* to find a Cholesky factorization of A. Compare it with the result obtained in (d).

Solution.

(a)
```
format rat;
A=hilb(4);

eig_val=eig(A);
if all(eig_val>0) && issymmetric(A)==1
    disp('Given matrix A is'); disp(A);
    disp('A is symmetric and positive definite matrix.');
    fprintf('eigen value of A is '); disp(eig_val');
end
```

MATLAB results.
```
Given matrix A is
       1              1/2            1/3            1/4
       1/2            1/3            1/4            1/5
       1/3            1/4            1/5            1/6
       1/4            1/5            1/6            1/7

A is symmetric and positive definite matrix.
eigen value of A is    66/682507   101/14989   262/1549   3500/2333
```

(b)
```
%--- This is a function file 'ludecomp.m' ---%
function [L, U] = ludecomp(A)

% The number of rows and columns of the matrix A.
[nrow, ncol] = size(A);

% Initialization of U and L.
U = A; L = eye(ncol);

% Forward Elimination %
for i=1:nrow
    % Find the first nonzero entry of the ith row.
    for k=i:ncol
        if U(i,k) ~= 0
            break % Terminates the execution of the loop.
        end
    end
    temp1 = U(i,k); % Save U(i,k) in temp.
    U(i,:) = (1/temp1) * U(i,:);
    % Normalize the pivot (i,k)-entry by 1 to the ith row.
    L(i,i) = (1/temp1)^(-1);
    % Place the reciprocal of the multiplier in that position in U.
        if i ~= nrow
        for j=(i+1):nrow
            temp2 = U(j,k); % Save U(j,k) in temp2.
            U(j,:) = ((-temp2) * U(i,:)) + U(j,:);
```

```
            % Add minus (j,k)-entry times the ith row to the jth row
            L(j,i) = -(-temp2);
            % Place the negative of the multiplier in that position in U.
        end
    end
end
```

(c)
```
% LU-factorization of A without row interchanges
[L, U] = ludecomp(A);

% From an LU-factorization of A, we can find the LDU-factorization of A,
% by appropriate normalization of L.
D = diag(diag(L));

for i = 1:4
    L(:, i) = L(:, i)./L(i, i);
end

disp('The LDU-factorization of A is');
disp('L = '); disp(L); disp('D = '); disp(D); disp('U = '); disp(U);
```

MATLAB results.

```
The LDU-factorization of A is
L =
       1              0              0              0
       1/2            1              0              0
       1/3            1              1              0
       1/4            9/10           3/2            1

D =
       1              0              0              0
       0              1/12           0              0
       0              0              1/180          0
       0              0              0              1/2800

U =
       1              1/2            1/3            1/4
       0              1              1              9/10
       0              0              1              3/2
       0              0              0              1
```

(d)
```
% From the LDU-factorization of A, find the Cholesky factor.
R1 = (L*sqrt(D))';
disp('The Cholesky factor R1 from the LDU-factorization of A is');
disp(R1);
```

MATLAB results.

```
The Cholesky factor R1 from the LDU-factorization of A is
       1              1/2            1/3            1/4
       0            390/1351       390/1351       351/1351
       0              0            317/4253       323/2889
       0              0              0            153/8096
```

(e)
```
% Find a Cholesky factorization of A by using the MATLAB command.
R2 = chol(A);
disp('The Cholesky factor R2 from the MATLAB command chol is');
disp(R2);
```

MATLAB results.

```
The Cholesky factor R2 from the MATLAB command chol is
       1              1/2            1/3            1/4
       0            390/1351       390/1351       351/1351
       0              0            317/4253       323/2889
       0              0              0            153/8096
```