Mathematics for Data Science: Linear Algebra with Matlab

César Pérez López

Mathematics for Data Science:

Linear Algebra with Matlab

César Pérez López

TABLE OF CONTENTS

ALGEBRAIC EXPRESSIONS, POLYNOMIALS, EQUATIONS AND SYSTEMS

MATRICES, VECTOR SPACES AND LINEAR APPLICATIONS

Chapter 1.

INTRODUCTION. NUMBERS, VARIABLES, OPERATORS AND FUNCTIONS

1.1 MATHEMATICS FOR DATA SCIENCE

M athematics plays a critical role in data science, as it provides the foundational tools and methods used for analyzing and interpreting data. A solid grasp of key mathematical concepts allows data scientists to work effectively with algorithms, make informed decisions, and extract meaningful insights from large datasets.

Here are the main mathematical topics relevant to data science:

- **1. Linear Algebra**

Linear algebra is essential for handling and manipulating data, especially when dealing with large datasets or working with machine learning algorithms.

- **Vectors and matrices** : Data can often be represented in vector and matrix form. For example, a dataset might be represented as a matrix where each row is a data point and each column is a feature.
- **Matrix operations** : Multiplication, inversion, and decomposition (e.g., Singular Value Decomposition, Principal Component Analysis) are key operations used in data processing.
- **Eigenvalues and Eigenvectors** : Used in dimensionality reduction techniques like PCA (Principal Component Analysis).

- **2. Calculus**

Calculus helps in understanding the optimization processes involved in training machine learning models.

- **Differentiation** : Derivatives are used to find the minimum or maximum of functions, which is a core part of optimization algorithms like gradient descent.
- **Optimization** : Gradient descent, a key method for training machine learning models, uses calculus to minimize the loss function.
- **Integration** : Important in probabilistic models and understanding areas under curves (like probability distributions).
- **3. Probability and Statistics**

Data science is all about making inferences from data, and probability and statistics provide the tools for that.

- **Probability theory** : Understanding concepts like random variables, probability distributions (e.g., Gaussian, Poisson), and Bayes' Theorem.
- **Statistical inference** : Estimating parameters, hypothesis testing, p-values, confidence intervals.
- **Distributions** : Common distributions like normal, binomial, and Poisson are essential in modeling real-world data.
- **Sampling** : Techniques such as random sampling and stratified sampling help ensure that datasets are representative of the population.
- **Regression and correlation** : Used to model relationships between variables.

- **4. Optimization**

Optimization is central to machine learning, as it is used to find the best parameters for models.

- **Convex optimization** : Many machine learning algorithms, like support vector machines (SVM), rely on convex optimization to ensure that they find a global minimum.
- **Linear programming** : Used in operations research and optimization problems with linear constraints and objectives.

- **5. Discrete Mathematics**

Discrete math provides important concepts for data structures, algorithms, and graph theory, which are key in data science.

- **Graph theory** : Used for analyzing networks (e.g., social networks, recommendation systems) and search algorithms.
- **Combinatorics** : Important for counting and understanding the complexity of different problems.
- **Set theory** : Used in data cleaning and data processing to work with different data types and subsets.

- **6. Numerical Methods**

Numerical methods are used to solve mathematical problems that are difficult or impossible to solve analytically.

- **Root finding** : Methods like Newton’s method are used for solving equations.
- **Numerical integration and differentiation** : Used to approximate solutions when analytical methods are not feasible.
- **Solving systems of equations** : Important when working with large-scale data and machine learning models.

- **7. Information Theory**

Information theory helps in understanding how to quantify uncertainty and information content.

- **Entropy** : Measures uncertainty or randomness in data, and is used in decision trees and other models.
- **Kullback-Leibler divergence** : Used to measure the difference between two probability distributions.

- **8. Time Series Analysis**

In data science, especially for fields like finance or forecasting, time series analysis is essential for understanding trends, patterns, and seasonality.

- **Autocorrelation** : Measures how a time series correlates with a lagged version of itself.
- **ARIMA models** : Used for forecasting time series data based on autoregressive and moving average processes.

- **9. Bayesian Inference**

Bayesian statistics is a method of statistical inference in which Bayes' theorem is used to update the probability for a hypothesis as more evidence becomes available.

- **Prior and Posterior distributions** : In Bayesian inference, the prior represents what is known before data is observed, and the posterior is the updated belief after observing the data.

- ## 10. Key Concepts and Algorithms

- **Machine Learning Algorithms** : Understanding the math behind algorithms like K-Nearest Neighbors (K-NN), Decision Trees, SVMs, and Neural Networks often requires knowledge of optimization, calculus, and linear algebra.
- **Deep Learning** : Requires more advanced calculus and linear algebra, especially when working with large neural networks and backpropagation.

Mathematics is an essential toolset for data scientists. By mastering these mathematical concepts, data scientists can better understand the underlying mechanisms of machine learning algorithms, improve their ability to clean and manipulate data, and make more accurate predictions and decisions based on data.

Matlab has functions that allow easy work in Linear Algebra. In this book, typical algebra topics are developed, such as work in discrete mathematics through numerical algebra in the real and complex fields. Work with algebraic expressions, polynomials, equations, systems of equations, matrices, vector spaces, linear maps, and quadratic forms is presented. Matrix algebra is specially developed with advanced treatment of eigenvalues, eigenvectors and diagonalization. He also delves into drawing curves and surfaces in explicit, implicit, parametric, and polar coordinates. The concepts are accompanied by examples solved step by step with Matlab

1.2 THE MATLAB LANGUAGE

T he matrix-based MATLAB language is the most natural way to express computational mathematics. Built-in graphics make it easy to visualise data and derive information from it. The desktop environment invites you to experiment, explore, and discover. All of these MATLAB tools and features are rigorously tested and designed to work together.

Mathematical functions provide a wide variety of computational methods for analysing data, developing algorithms and creating models. The core functions use processor-optimised libraries to enable fast vector and matrix calculations. You can use MATLAB as a powerful numerical computer. While most calculators handle numbers only to a preset degree of precision, MATLAB performs exact calculations to any desired degree of precision. In addition, unlike calculators, we can perform operations not only with individual numbers, but also with objects such as matrices. Most topics in algebra and classical numerical analysis are handled by this software. It supports matrix calculus, statistics, interpolation, least squares fitting, numerical integration, function minimisation, linear programming, numerical and algebraic solutions of differential equations and a long list of additional methods that make MATLAB a very suitable software for mathematical work.

1.3 GETTING STARTED WITH MATLAB ON WINDOWS. THE MATLAB FRAMEWORK

T o start MATLAB, simply double-click the program shortcut icon on the Windows desktop. Alternatively, if there is no shortcut on the desktop, the easiest and most common way to run the program is to choose Programs from the Windows Start menu and select MATLAB. Having started MATLAB using either of these methods, the welcome screen appears briefly, followed by the screen shown in Figure 1-1, which provides the general environment in which the program operates.

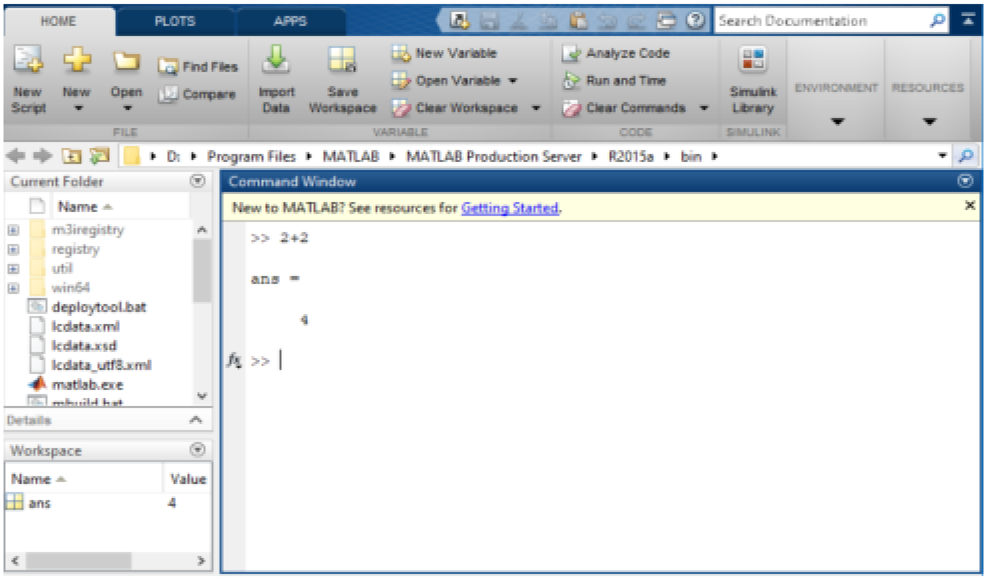
The most important elements of the MATLAB screen are as follows:

- The *Commnd Window* : executes MATLAB functions.

- The *Current Folder* : Displays MATLAB files and execution files (such as open operations and content search).

- The *Workspace* : displays the current contents of the Workspace and allows you to make changes to it.

- The Menu Options: Displays the most important options for working with the programme (Figure 1-2).



```
>> 2+2

ans =

     4

>> |
```

Figure 1-1

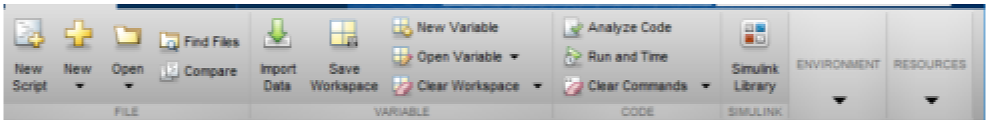
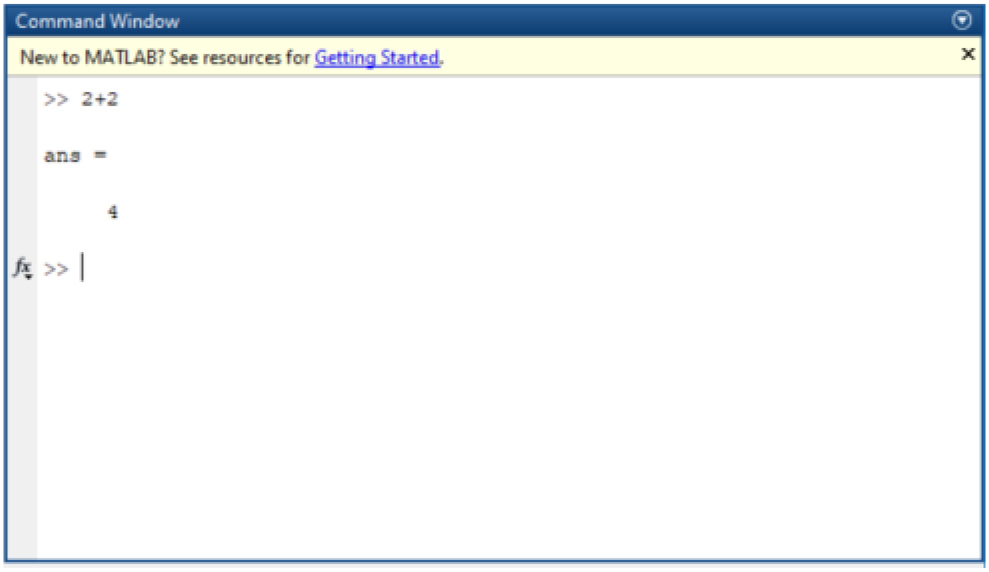


Figure 1-2

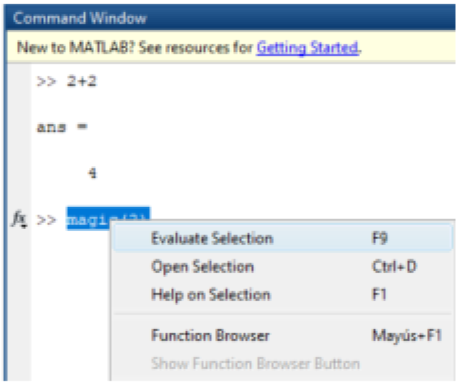
The Command Window (Figure 1-3) is the primary way to communicate with MATLAB. It appears on the desktop when MATLAB is started and is used to execute all operations and functions. Entries are typed below the >> prompt and, once completed, are executed after pressing *Enter.*



```
>> 2+2

ans =

     4

>> 
```

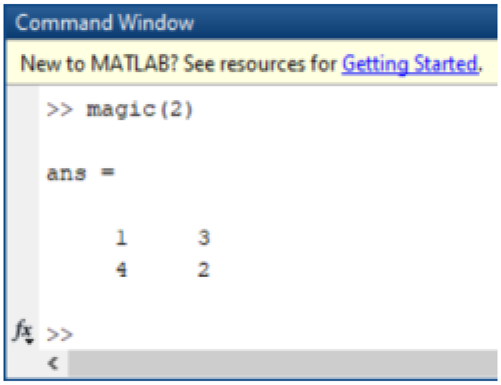
Figure 1-3

In the Command Window you can work with operations, functions, graphics and MATLAB code in general. Simply type an expression with valid MATLAB syntax and it will be executed by pressing *Enter* . It is also possible to evaluate previously executed operations. To do this, simply select the syntax you wish to evaluate, right-click and choose *Evaluate selection* from the resulting pop-up menu (Figures 1-4 and 1-5). Choosing *Open selection* on a command from the same menu opens in the Editor / Debugger an M-file with the code of the command previously selected in the Command Window (Figures 1-6 and 1-7). In this way the MATLAB programming code is visible.



New to MATLAB? See resources for Getting Started.

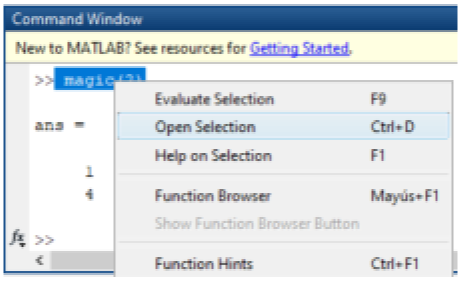
```
>> 2+2

ans =

     4

>> magic(2)
```



New to MATLAB? See resources for Getting Started.

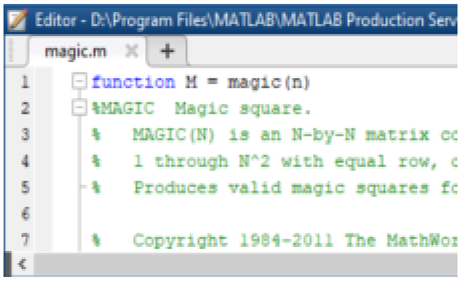
```
>> magic(2)

ans =

     1     3
     4     2

>>
```

Figure 1-4   Figure 1-5



New to MATLAB? See resources for Getting Started.

```
>> magic(2)

ans =

     1
     4

>>
```

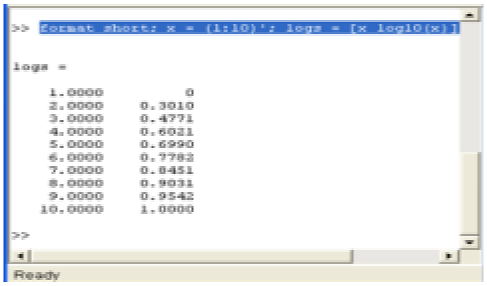


```
function M = magic(n)
%MAGIC  Magic square.
%   MAGIC(N) is an N-by-N matrix co
%   1 through N^2 with equal row, c
%   Produces valid magic squares fo

%   Copyright 1984-2011 The MathWor
```

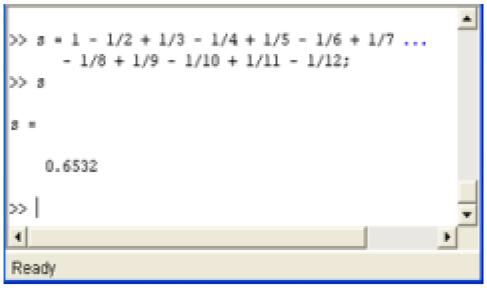
Figure 1-6   Figure 1-7

M ATLAB is case-sensitive , and you can use white space before and after minus signs, colons, and parentheses. MATLAB also allows you to type multiple commands on the same line, as long as they are separated by semicolons (Figure 1-8). The entries are executed sequentially in the order in which they appear on the line. Each command that ends with a semicolon will be executed, but will not display its output.

Long entries that do not fit on one line can be continued on a second line by placing dots at the end of the first line (Figure 1-9).



```
>> format short; x = (1:10)'; logs = [x log10(x)]

logs =

    1.0000         0
    2.0000    0.3010
    3.0000    0.4771
    4.0000    0.6021
    5.0000    0.6990
    6.0000    0.7782
    7.0000    0.8451
    8.0000    0.9031
    9.0000    0.9542
   10.0000    1.0000

>>
```



```
>> s = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 ...
     - 1/8 + 1/9 - 1/10 + 1/11 - 1/12;
>> s

s =

    0.6532

>> |
```

Figure 1-8   Figure 1-9

B elow is a list of keys, arrows and combinations that can be used in the Command Window.

Key	***Control key***	***Operation***
↑	**CTRL+ p**	*Call the last executed entry.*
↓	**CTRL+ n**	*Call up the following line.*
←	**CTRL+ b**	*Move one character backwards.*
→	**CTRL+ f**	*Moves one character forward.*
CTRL+ →	**CTRL+ r**	*Move one word to the right.*
CTRL+ ←	**CTRL+ l**	*Move one word to the left.*
Home	**CTRL+ a**	*Move to the beginning of the line.*
End	**CTRL+ e**	*Move to the end of the line.*
ESC	**CTRL+ u**	*Delete the line.*
Delete	**CTRL+ d**	*Deletes the character where the cursor is.*
BACKSPACE	**CTRL+ h**	*Deletes the character before the cursor.*
	CTRL+ k	*Deletes all text up to the end of the line.*

Shift+ home	*Lightens the text from the beginning of the line.*
Shift+ end	*Lightens the text up to the end of the line.*

1.4 NUMBERS, ARITHMETIC OPERATIONS AND FORMATS

N umerical work in MATLAB is very extensive. Integers, rational, real and complex numbers can be used, which in turn can be arguments of functions giving rise to integer, rational, real and complex functions. Therefore, the complex variable is a tractable field in MATLAB. Arithmetic operations in MATLAB are defined according to standard mathematical conventions. The following table presents the syntax of the basic arithmetic operations:

Sum

x + y

```
>> 4+8
ans =
12
```

Difference

x - y

```
>> 4-8
ans =
-4
```

Product

x * y or x y

```
>> 4*8
ans =
32
```

Division

x/y

```
>> 4/8
ans =
0.5000
```

Power

x ^y

```
>> 4^8
ans =
65536
```

MATLAB performs arithmetic operations as if it were a conventional calculator, but with full computational accuracy. The results of the operations are displayed either in exact form or by specifying the degree of precision the user desires. But this only affects the presentation, since the precision of the calculation is unlimited, which is the feature that differentiates MATLAB from other numerical calculation programs in which the word length with which the computer works determines the precision (hardware-dependent precision). MATLAB can represent the results with the required accuracy, although internally it always works with exact calculations to avoid rounding errors. This is how we have the different formats of approximate representation, which sometimes facilitate the interpretation of the results.

The commands relating to formats for the presentation of results are listed below:

format short

It gives the results to 4 decimal places. This is the default format of MATLAB.

```
>> sqrt(23)
ans =
4.7958
```

format long

Gives results to 16 decimal places

```
>> format long; sqrt (23)
    ans =
    4.795831523312719
```

format long e

Provides results to 16 decimal places plus the required power of 10.

```
>> format long e; sqrt(23)
```

```
ans =
4.795831523312719e+000
```

format short e

Gives results to 4 decimal places plus the required power of 10.

```
>> format short e; sqrt(23)
ans =
4.7958e+000
```

format long g

Delivers results in optimal long format

```
>> format long g; sqrt(23)
ans =
4.79583152331272
```

format short g

Delivers results in optimal short format

```
>> format short g; sqrt(23)
ans =
4.7958
```

format bank

Gives results to 2 decimal places

```
>> format bank; sqrt(23)
ans =
4.80
```

format rat

Gives the results in the form of an approximate rational number.

```
>> format rat; sqrt(23)
ans =
1151/240
```

format +

Gives the sign of the results (+, -) and ignores the imaginary part of complex numbers.

```
>> format +; sqrt(23)
ans =
+
```

hex format

Gives the results in the hexadecimal system.

```
>> format hex; sqrt(23)
```

	ans = 40132eee75770416
vpa 'operations' n	*Gives the result of operations to n exact decimal digits* >> vpa 'sqrt(23)' 20 ans = 4.7958315233127195416

1.5 INTEGERS AND INTEGER VARIABLE FUNCTIONS

M ATLAB works with integers and integer variable functions exactly. Regardless of the format in which the results are displayed, the calculations are exact. However, the *vpa* command allows you to obtain exact outputs with the required precision.

As for the functions with integer variables, the most important ones that MATLAB provides are the following (the expressions in inverted commas are in string format):

rem(n,m)

Remainder of the division of n by m

(function valid for real n and m)

```
>> rem(15,2)
ans =
1
```

sign(n)

Sign of n (1 if n>0 , -1 if n<0, n real)

```
>> sign(-8)
ans =
-1
```

max(n1,n2)

Maximum of the numbers n1 and n2

```
>> max(17,12)
ans =
17
```

min(n1,n2)

Minimum of the numbers n1 and n2

```
>> min(17,12)
ans =
12
```

gcd(n1,n2)

Greatest common divisor of n1 and n2

```
>> gcd(17,12)
```

```
ans =
1
```

lcm(n1,n2)

Least common multiple of n1 and n2

```
>> lcm(17,12)
ans =
204
```

factorial(n)

Factorial of n (n(n-1)(n-2)...1)

```
>> factorial(9)
ans =
362880
```

factor(n)

Decompose n into prime factors

```
> factor(51)
ans =
3 17
```

dec2base(decimal,n_base)

Converts the specified decimal number (base 10) to the new given base n_base

```
>> dec2base(2345,7)
ans =
6560
```

base2dec(number,B)

Convert the given number in base B to decimal

```
>> base2dec('ab12579',12)
ans =
32621997
```

dec2bin(decimal)

Converts specified decimal number to base 2 (binary)

```
>> dec2bin(213)
ans =
11010101
```

dec2hex(decimal)

Converts the specified decimal number to base 16 (Hexadecimal)

```
>> dec2hex(213)
```

	ans = D5
bin2dec(binary)	*Converts the specified binary number to decimal base* >> bin2dec('1110001') ans = 113
hex2dec(hexadecimal)	*Converts the specified base 16 number to decimal base* >> hex2dec('FFAA23') ans = 16755235

1.6 REAL NUMBERS AND FUNCTIONS OF REAL VARIABLES

The real numbers are the disjoint union of the rational numbers and the irrational numbers. A rational number is of the form *p/q* , where *p* is an integer and *q is* another integer. MATLAB's treatment of rationals is different from that of most calculators. Rational numbers are quotients of integers, and MATLAB can also work with them in exact mode, so that the result of expressions involving rational numbers is always another rational number or integer. To do this, it is necessary to activate the rational format with the *format rat* command. MATLAB also returns decimal approximations of the results if the user so desires, by activating any other type of format (for example, *format short* or *format long*). Consider the following example:

```
> format rat;
>> 1/6+1/5-2/10
ans =
1/6
```

MATLAB is dealing with rationals as quotients of integers and keeps them in this form during calculations. In this way, rounding errors are not carried over in calculations with fractions, which can be very serious, as shown in Error Theory. Once the rational format is enabled, operations with rationals will be exact until a different format is enabled. When the rational format is enabled, a floating-point number, that is, a number with a decimal point, is interpreted as exact and MATLAB treats the whole expression as exact rational by representing the result in rational numbers. In turn, if there is an irrational number in a rational expression, MATLAB matches it to the closest fraction to work in rational format. Consider the following examples:

```
>> format rat;
>> 2.64/25+4/100
ans =
91/625
>> 2.64/sqrt(25)+4/100
ans =
71/125
>> sqrt(2.64)/25+4/100
ans =
204/1943
```

MATLAB also works with irrational numbers by representing the results as accurately as it can or as precisely as required by the user, keeping in mind the caveat that irrationals cannot be represented exactly as the ratio of two integers. An example is given below.

```
>> sqrt(235)
ans =
15.3297
```

There are very typical real constants that MATLAB represents as follows:

Number π= 3.1415926

pi

```
>> 2*pi
ans =
6.2832
```

Number e = 2.7182818

exp(1)

```
>> exp(1)
ans =
2.7183
```

inf *Infinity (e.g. 1/0)*

```
>> 1/0
ans =
Inf
```

NaN *Indeterminacy (e.g. 0/0)*

```
>> 0/0
ans =
NaN
```

realmin *Smallest usable positive real number*

```
>> realmin
ans =
2.2251e-308
```

realmax *Largest usable positive real number*

```
>> realmax
ans =
1.7977e+308
```

MATLAB has a range of predefined real variable functions, which will of course be valid for rational, irrational and integer variables. The most important of these are presented in the following paragraphs.

1.6.1 TRIGONOMETRIC FUNCTIONS

Below is a table with the trigonometric functions and their inverses that MATLAB incorporates, illustrated with examples.

Function	***Inverse***
sin(x) *Sine* >> sin(pi/2) ans = 1	**asin(x)** *Sine arc* >> asin(1) ans = 1.5708
cos(x) *Cosine* >> cos(pi) ans = -1	**acos(x)** *Arc cosine* >> acos(-1) ans = 3.1416
tan(x) *Tangent* >> tan(pi/4) ans = 1.0000	**atan(x) and atan2(x)** *Arc tangent* >> atan(1) ans = 0.7854
csc(x) *Cosine* >> csc(pi/2) ans = 1	**acsc(x)** *Cosine arc* >> acsc(1) ans = 1.5708
sec(x) *Secant* >> sec(pi) ans = -1	**asec(x)** *Secant arc* >> asec(-1) ans = 3.1416
cot(x) *Cotangent* >> cot(pi/4) ans = 1.0000	**acot(x)** *Arc cotang.* >> acot(1) ans = 0.7854

1.6.2 HYPERBOLIC FUNCTIONS

B elow is a table of hyperbolic functions and their inverses in MATLAB illustrated with examples.

Function	*Inverse*
sinh(x) *Hyperbolic sine* >> sinh(2) ans = 3.6269	**asinh(x)** *Hyperbolic sine arc* >> asinh(3.6269) ans = 2.0000
cosh(x) *Hyperbolic cosine* >> cosh(3) ans = 10.0677	**acosh(x)** *Hyperbolic cosine arc* >> acosh(10.0677) ans = 3.0000
tanh(x) *Hyperbolic tangent* >> tanh(1) ans = 0.7616	**atanh(x)** *Hyperbolic arc tangent* >> atanh(0.7616) ans = 1.0000
csch(x) *Hyperbolic cosecant* >> csch(3.14159) ans = 0.0866	**acsch(x)** *Hyperbolic cosecant arc* >> acsch(0.0866) ans = 3.1415
sech(x) *Hyperbolic secant* >> sech(2.7182818) ans = 0.1314	**asech(x)** *Hyperbolic secant arc* >> asech(0.1314) ans = 2.7183
coth(x) *Hyperbolic cotangent* >> coth(9) ans = 1.0000	**acoth(x)** *Cotangent hyperbolic arc* >> acoth(0.9999) ans = 4.9517 + 1.5708i

1.6.3 EXPONENTIAL AND LOGARITHMIC FUNCTIONS

B elow is a table of the exponential and logarithmic functions that MATLAB incorporates, illustrated with examples.

Function	***Meaning***
exp(x)	*Exponential function in base e (e^x)* >> exp(log(7)) ans = 7
log(x)	*Logarithm function in base e of x* >> log(exp(7)) ans = 7
log10(x)	*Logarithm function in base 10 of x* >> log10(1000) ans = 3
log2(x)	*Logarithm function in base 2 of x* >> log2(2^8) ans = 8
pow2(x)	*Function power of base 2 of x* >> pow2(log2(8)) ans = 8
sqrt(x)	*Square root function of x* >> sqrt(2^8) ans = 16

1.6.4 NUMERIC VARIABLE SPECIFIC FUNCTIONS

M ATLAB incorporates a group of numeric variable functions whose purpose is to obtain approximate results of operations and rounding, to work with signs, etc. These functions include the following:

Function ***Meaning***

Absolute value of real x

abs(x)

```
>> abs(-8)
ans =
8
```

The largest integer less than or equal to the real x

floor(x)

```
>> floor(-23.557)
ans =
-24
```

The smallest integer greater than or equal to the real x

ceil(x)

```
>> ceil(-23.557)
ans =
-23
```

The nearest integer to the real x

round(x)

```
>> round(-23.557)
ans =
-24
```

Removes the decimal part of the real x

fix(x)

```
>> fix(-23.557)
ans =
-23
```

rem(a,b) *Gives the remainder of the division between the reals a and b.*

```
>> rem(7,3)
```

```
ans =
1
```

sign(x)

Sign of real x (1 if x>0 , -1 if x<0)

```
>> sign(-23.557)
ans =
-1
```

1.7 ONE-DIMENSIONAL, VECTOR AND MATRIX VARIABLES

M ATLAB is a software focused on the matrix language, so it focuses especially on working with matrices.

The initial way to define a variable is very simple. Just use the following syntax:

Variable = object

Where the object can be a scalar, a vector or a matrix.

If the variable is a vector, its syntax in terms of its components can be written in one of the following two ways:

Variable = [v1 v2 v3 ... vn].

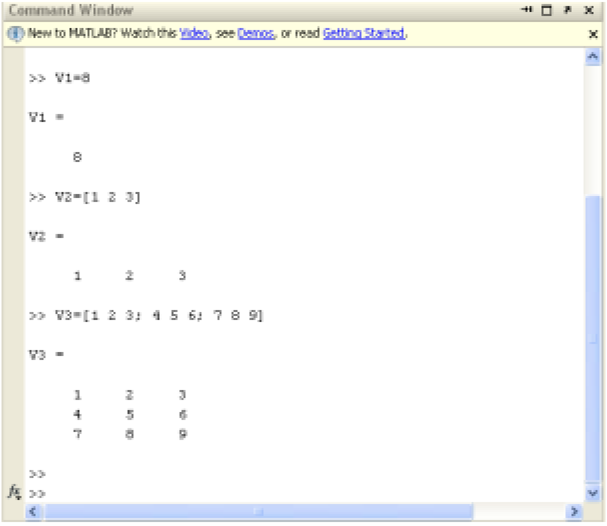
Variable = [v1, v2, v3, ..., vn].

If the variable is an array, its syntax in terms of its components can be written in one of the following two ways:

Variable = [v11 v12 v12 v13 ... v1n; v21 v22 v23 ... v2n; ...]

Variable = [v11, v12, v13, ..., v1n; v21, v22, v23, ..., v2n; ...]

Writing variables is done in the MATLAB *Command Window in* a natural way, as shown in Figure 1-10.

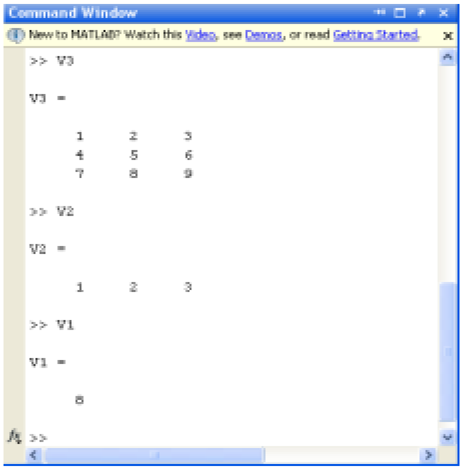

Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
>> V1=8

V1 =

     8

>> V2=[1 2 3]

V2 =

     1     2     3

>> V3=[1 2 3; 4 5 6; 7 8 9]

V3 =

     1     2     3
     4     5     6
     7     8     9

>>
>>
```

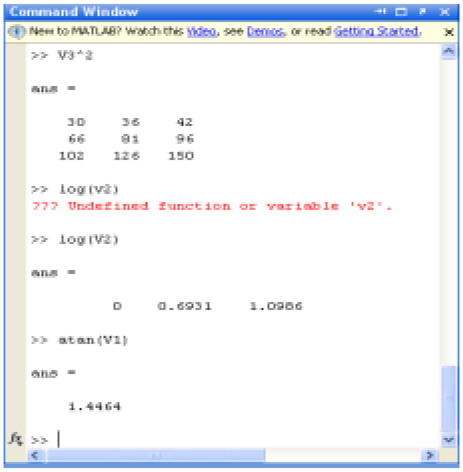
Figure 1-10

O nce a variable is defined, we can operate on it by using it as a regular variable in mathematics, keeping in mind that variable names are case sensitive (Figure 1-11). Figure 1-12 shows some operations with one-dimensional, vector and matrix variables. It is important to note that when calculating the logarithm of V2 the error was made of using the variable in lower case.

In MATLAB, variable names begin with a letter followed by any number of letters, digits, or underscores up to 31 characters.


Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
>> V3

V3 =

     1     2     3
     4     5     6
     7     8     9

>> V2

V2 =

     1     2     3

>> V1

V1 =

     8

>>
```


Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
>> V3^2

ans =

    30    36    42
    66    81    96
   102   126   150

>> log(v2)
??? Undefined function or variable 'v2'.

>> log(V2)

ans =

         0    0.6931    1.0986

>> atan(V1)

ans =

    1.4464

>> 
```

Figure 1-11   Figure 1-12

There are also specific ways of defining vector variables, including the following:

variable=[a:b] *Defines the vector whose first and last elements are a and b, respectively, and the elements in between differ by one unit*

```
>> vector1=[2:6]
vector1 =
2 3 4 5 6
```

variable=[a:s:b] *Defines the vector whose first and last elements are a and b, and the elements in between differ by the quantity s specified by the increment*

```
>> vector2=[2:2:8]
vector2 =
2 4 6 8
```

variable=linespace[a,b,n] *Defines the vector whose first and last elements are a and b, and which has a total of n evenly spaced elements*

```
> vector3=linspace(10,30,6)
vector3 =
10 14 18 22 26 30
```

1.7.1 ELEMENTS OF VECTOR VARIABLES

M ATLAB allows the selection of elements of vector variables using the following commands:

x(n)

Returns the nth element of the vector x

```
>> X=(2:8)
X =
2 3 4 5 6 7 8
>> X(3)
ans =
4
```

x(a:b)

Returns the elements of the vector x located between the a-th and the b-th, both inclusive.

```
>> X(3:5)
ans =
4 5 6
```

x(a:p:b)

Returns the elements of the vector x located between the a-th and the b-th, both inclusive, but separated from p by p units (a>b).

```
>> X(1:2:6)
ans =
2 4 6
```

x(b:-p:a)

Returns the elements of the vector x located between the bth and the ath, both inclusive, but separated from p by p units and starting with the bth (b>a).

```
>> X(6:-2:1)
ans =
7 5 3
```

1.7.2 ELEMENTS OF MATRIX VARIABLES

A s in the case of vectors , MATLAB allows the selection of matrix variable elements using the following commands:

A(m,n) *Defines element (m,n) of matrix A (row m and column n)*

```
>> A=[3 5 7 4; 1 6 8 9; 2 6 8 1]
A =
3 5 7 4
1 6 8 9
2 6 8 1
>> A(2,4)
ans =
9
```

A(a:b,c:d) *Define the submatrix of A consisting of the rows between the a-th and b-th rows and the columns between the c-th and d-th rows.*

```
>> A(1:2,2:3)
ans =
5 7
6 8
```

A(a:p:b,c:q:d) *Define the submatrix of A consisting of the rows between the a-th and b-th rows from p to p, and the columns between the c-th and d-th rows from q to q.*

```
>> A(1:2:3,1:2:4)
ans =
3 7
2 8
```

A([a b],[c d]) *Define the submatrix of A formed by the intersection of the a-th and b-th rows and the c-th and* *d-th columns.*

```
>> A([2 3],[2 4])
ans =
6 9
6 1
```

A([a b c ...], [e f g ...])

Define the submatrix of A formed by the intersection of rows a, b, c, ... and columns e, f, g, ...

```
>> A([2 3],[1 2 4])
ans =
1 6 9
2 6 1
```

A(:,c:d)

Define the submatrix of A consisting of all the rows of A and the columns between the c-th and d-th rows.

```
>> A(:,2:4)
ans =
5 7 4
6 8 9
6 8 1
```

A(:,[c d e ...])

Define the submatrix of A consisting of all rows of A and columns c, d, e, ...

```
>> A(:,[2,3])
ans =
5 7
6 8
6 8
```

A(a:b,:)

Defines the submatrix of A consisting of all the columns of A and the rows between the a-th and b-th rows.

```
>> A(2:3,:)
ans =
1 6 8 9
2 6 8 1
```

A([a b c ...],:)

Defines the submatrix of A consisting of all the *columns of A and the rows a, b, c, ...*

```
>> A([1,3],:)
ans =
3 5 7 4
2 6 8 1
```

A(a,:)

Defines the a-th row of the matrix A

```
>> A(3,:)
ans =
2 6 8 1
```

A(:,b)

Define the b-th column of the matrix A

```
>> A(:,3)
ans =
7
8
8
```

A(:)

Defines a column vector whose elements are the columns of A placed in order one below the other.

```
>> A(:)
ans =
3
1
2
5
6
6
7
8
8
4
9
1
```

A(:,:)

Equivalent to the whole matrix A

```
>> A(:,:)
ans =
3 5 7 4
```

```
1 6 8 9
2 6 8 1
```

[A,B,C,...]

Define the matrix formed by the submatrices A, B, C, ...

```
>> A1=[2 6;4 1], A2=[3 8;6 9], A2=[3 8;6 9], A2=[3 8;6 9].
A1 =
2 6
4 1
A2 =
3 8
6 9
>> [A1,A2] [A1,A2] [A1,A2] [A1,A2] [A1,A2] [A1,A2
ans =
2 6 3 8
4 1 6 9
```

1.7.3 SPECIFIC MATRIX FUNCTIONS

M ATLAB uses a set of predefined matrix functions that facilitate work in the matrix field. The most important of these are as follows:

diag(v)

Creates a diagonal matrix with the vector v on the diagonal

```
>> diag([2 0 9 8 7])
ans =
2 0 0 0 0
0 0 0 0 0
0 0 9 0 0
0 0 0 8 0
0 0 0 0 7
```

diag(A)

Extracts the diagonal of the matrix A as a column vector

```
>> A=[1 3 5 ;2 0 8;-1 -3 2]
A =
1 3 5
2 0 8
-1 -3 2
>> diag(A)
ans =
1
0
2
```

eye(n)

Creates the identity matrix of order n

```
>> eye(4)
ans =
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

eye(m,n)

Creates the matrix of order mxn with ones on the *main diagonal and zeros on the rest.*

```
>> eye(3,5)
ans =
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
```

zeros(m,n)

Creates the null matrix of order mxn

```
>> zeros(2,3)
ans =
0 0 0
0 0 0
```

ones(m,n)

Creates the matrix of order mxn with all its elements 1

```
>> ones(2,3)
ans =
1 1 1
1 1 1
```

rand(m,n)

Creates a uniform random matrix of order mxn

```
>> rand(4,5)
ans =
0.8147 0.6324 0.9575 0.9572 0.4218
0.9058 0.0975 0.9649 0.4854 0.9157
0.1270 0.2785 0.1576 0.8003 0.7922
0.9134 0.5469 0.9706 0.1419 0.9595
```

randn(m,n)

Creates a normal random matrix of order mxn

```
>> randn(4,5)
ans =
0.6715 0.4889 0.2939 -1.0689 0.3252
-1.2075 1.0347 -0.7873 -0.8095 -0.7549
0.7172 0.7269 0.8884 -2.9443 1.3703
1.6302 -0.3034 -1.1471 1.4384 -1.7115
```

flipud(A)

Returns the matrix whose rows are arranged in reverse order (from top to bottom) of the rows of A

```
>> flipud(A)
```

```
ans =
-1 -3 2
2 0 8
1 3 5
```

fliplr(A)

Returns the matrix whose columns are placed in reverse order (from left to right) of the columns of A.

```
>> fliplr(A)
ans =
5 3 1
8 0 2
2 -3 -1
```

rot90(A)

Rotate the matrix A by 90 degrees

```
>> rot90(A)
ans =
5 8 2
3 0 -3
1 2 -1
```

reshape(A,m,n)

Returns the matrix of order mxn extracted from the matrix A by taking consecutive elements of A by columns.

```
>> reshape(A,3,3)
ans =
1 3 5
2 0 8
-1 -3 2
```

size(A)

Returns the order (size) of the array A

```
>> size(A)
ans =
3 3
```

length(v)

Returns the length of the vector v

```
>> length([1 3 4 5 -1])
ans =
5
```

tril(A)

Returns the lower triangular part of the matrix A

```
>> tril(A)
ans =
1 0 0
2 0 0
-1 -3 2
```

triu(A)

Returns the upper triangular part of the matrix A

```
>> triu(A)
ans =
1 3 5
0 0 8
0 0 2
```

A'

Returns the transpose matrix of A

```
>> A'
ans =
1 2 -1
3 0 -3
5 8 2
```

inv(A)

Returns the inverse matrix of A

```
>> inv(A)
ans =
-0.5714 0.5000 -0.5714
0.2857 -0.1667 -0.0476
0.1429 0 0.1429
```

1.8 RANDOM NUMBERS

M ATLAB incorporates functions that make it possible to work with random numbers in a simple way. The *rand* and *randn* functions are the basic functions that generate uniformly and normally distributed random numbers respectively. The following are the most common functions for working with random numbers.

rand

Returns a random decimal number uniformly distributed over the interval [0,1].

```
>> rand
ans =
0.8147
```

rand(n)

Returns a matrix of dimension nxn whose elements are random decimal numbers uniformly distributed in the interval [0,1].

```
>> rand(3)
ans =
0.9058 0.6324 0.5469
0.1270 0.0975 0.9575
0.9134 0.2785 0.9649
```

rand(m,n)

Returns a matrix of dimension mxn whose elements are random decimal numbers uniformly distributed in the interval [0,1].

```
>> rand(2,3)
ans =
0.1576 0.9572 0.8003
0.9706 0.4854 0.1419
```

rand(size(A))

Returns a matrix of the same size as the matrix A and whose elements are random decimal numbers uniformly distributed in the interval [0,1].

```
>> rand(size(eye(3)))
ans =
0.4218 0.9595 0.8491
0.9157 0.6557 0.9340
0.7922 0.0357 0.6787
```

rand('seed')

Returns the current value of the uniform random number generator seed.

```
>> rand('seed')
ans =
931316785
```

rand('seed',n)

Place in quantity n the current value of the uniform random number generator seed.

```
>> rand('seed')
ans =
931316785
>> rand('seed',1000)
>> rand('seed')
ans =
1000
```

randn

Returns a random decimal number distributed according to a normal distribution of mean 0 and variance 1.

```
>> randn
ans =
-0.4326
```

randn(n)

Returns a matrix of dimension nxn whose elements are random decimal numbers distributed according to a normal of mean 0 and variance 1.

```
>> randn(3)
ans =
-1.6656 -1.1465 -0.0376
0.1253 1.1909 0.3273
0.2877 1.1892 0.1746
```

randn(m,n)

Returns a matrix of dimension mxn whose elements are random decimal numbers distributed according to a normal of mean 0 and variance 1.

```
>> randn(2,3)
ans =
-0.1867 -0.5883 -0.1364
0.7258 2.1832 0.1139
```

randn(size(A))

Returns a matrix of the same size as the matrix A and whose elements are random decimal numbers distributed according to a normal of mean 0 and variance 1.

```
>> randn(size(eye(3)))
ans =
1.0668 -0.8323 0.7143
0.0593 0.2944 1.6236
-0.0956 -1.3362 -0.6918
```

randn('seed')

Returns the current value of the normal random number generator seed.

```
>> randn('seed')
ans =
931316785
```

randn('seed',n)

Place in quantity n the current value of the uniform random number generator seed.

```
>> randn('seed',1000)
>> randn('seed')
ans =
1000
```

1.9 OPERATORS

M ATLAB is a language that incorporates arithmetic, logical and relational operators in the same way as any other language. The following is a list of the types of operators cited within the scope of the MATLAB language.

1.9.1 ARITHMETIC OPERATORS

M ATLAB incorporates the usual arithmetic operators (addition, subtraction, multiplication and division) for working with numbers. But at the same time it extends the meaning of these operators to work with scalars, vectors and matrices, as shown in the table below:

Operator	***Role***
+	*Sum of scalars, vectors or matrices* >> A=[1 3 -2 6];B=[4 -5 8 2]; c=3; >> V1=A+c V1 = 4 6 1 9 >> V2=A+B V2 = 5 -2 6 8
-	*Subtraction of scalars, vectors or matrices* >> V3=A-c V3 = -2 0 -5 3 >> V4=A-B V4 = -3 8 -10 4
*	*Product of scalars or of matrices or of scalars by vectors or matrices* >> V3=A*c V3 = 3 9 -6 18
.*	*Product of scalars or element-by-element vectors* >> V4=A.*B V4 = 4 -15 -16 12

.^ *Power of vectors (A.^B = [$A(i,j)^{B(i,j)}$], A and B vectors)*

```
>> V6=A.^B
V6 =
1.0000 0.0041 256.0000 36.0000
>> A.^c
ans =
1 27 -8 216
>> c.^A
ans =
3.0000 27.0000 0.1111 729.0000
```

./ *A./B = [A(i,j)/B(i,j)], where A and B are vectors [dim(A)=dim(B)].*

```
>> V7=A./B
V7 =
0.2500 -0.6000 -0.2500 3.0000
```

. *A.B = [B(i,j)/A(i,j)], where A and B are vectors [dim(A)=dim(B)].*

```
>> V8=A.B
V8 =
4.0000 -1.6667 -4.0000 0.3333
```

\ *A = inv(A)*B, where A and B are matrices A and B.*

```
    >> A=rand(4)
    A =



0 .6868 0.5269 0.7012 0.0475
    0.5890 0.0920 0.9103 0.7361
    0.9304 0.6539 0.7622 0.3282
    0.8462 0.4160 0.2625 0.6326
    >> B=randn(4)



B =
    -0.1356 -0.0449 -0.0562 0.4005
    -1.3493 -0.7989 0.5135 -1.3414
    -1.2704 -0.7652 0.3967 0.3750
    0.9846 0.8617 0.7562 1.1252
    >> A
    ans =
    25.0843 17.5201 -1.8497 7.1332
    -25.8285 -17.9297 2.4290 -4.6114
    -4.4616 -3.1424 -0.2409 -2.7058
    -13.1598 -8.9778 2.1720 -3.6075
```

*Scalar quotient or B/A = B*inv(A), where A and B are matrices*

/

```
>> B/A
ans =
-4.8909 0.0743 4.8972 -1.6273
4.6226 0.9230 -4.2151 -1.3541
-10.2745 1.2107 10.4001 -5.4409
-9.6925 -0.1247 11.1342 -3.1260
```

Power of scalars or matrix scalar power (M^p)

^

```
>> A^3
ans =
3.5152 2.1281 3.1710 1.6497
4.0584 2.3911 3.7435 1.9881
4.6929 2.8299 4.2067 2.2186
3.5936 2.1520 3.2008 1.7187
```

1.9.2 LOGICAL OPERATORS

M ATLAB also incorporates the usual logical operators using the most common notation for them. Typically, the results of the logical operators are 1 if true and 0 if false. The following table shows these operators.

<table>
<tr><td><b>~A</b></td><td><i>Logical negation (NOT) or complementary of A</i>
>> not(2>3)
ans =
1</td></tr>
<tr><td><b>A & B</b></td><td><i>Logical conjunction (AND) or intersection of A and B</i>
>> (2>3) & (5>1)
ans =
0</td></tr>
<tr><td><b>A | B</b></td><td><i>Logical disjunction (OR) or union of A and B</i>
>> (2>3)||(5>1)
ans =
1</td></tr>
</table>

xor(A,B) *Exclusive OR (XOR) or symmetrical difference of A and B (worth 1 if either A or B, but not both, are 1)*

```
>> xor((2>3),(5>1))

ans =

1
```

1.9.3 RELATIONAL OPERATORS

M ATLAB also deals with relational operations that perform element-to-element comparisons between two arrays and return an array of the same size whose elements are zeros if the corresponding relation is true, or ones if the corresponding relation is false. Relational operators also allow scalars to be compared with vectors or matrices, in which case the scalar is compared with all the elements of the matrix. The following table shows the MATLAB relational operators.

<

Minor (for complexes only affecting real parts)

```
>> 3<5
ans =
1
```

<=

Less than or equal to (only affects real parts)

```
>> 4>=6
ans =
0
```

>

Major (only affects real parts)

```
>> X=3*ones(3,3)
X =
3 3 3
3 3 3
3 3 3
>> X>[1 2 3; 4 5 6; 1 2 3]
ans =
1 1 0
0 0 0
1 1 0
```

>=

Greater than or equal to (only affects real parts)

```
>> X>=[1 2 3; 4 5 6; 1 2 3]
ans =
1 1 1
```

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

Equality (concerns complex numbers)

x == y

```
>> X==ones(3,3)
ans =
0 0 0
0 0 0
0 0 0
```

Inequality (concerns complex numbers)

x ~= y

```
>> X~=ones(3,3)
ans =
1 1 1
1 1 1
1 1 1
```

1.10 SYMBOLIC VARIABLES

Until now, we have always handled numerical variables. However, MATLAB allows us to handle symbolic mathematical calculations perfectly, to manipulate formulas and algebraic expressions easily and quickly and to perform most operations with them.

However, to perform these tasks it is necessary to have the MATLAB *Symbolic Math Toolbox* module. For an algebraic variable or expression to be symbolic, it must first be declared as such with the *syms* command , with the *sym* command or by inserting it in inverted commas. The following table shows the syntax of the commands for conversion to variables and symbolic expressions.

syms x y z...t	*Converts the variables x, y, z, ..., t into symbolic variables.* >> syms x y >> x+x+y-6*y ans = 2*x - 5*y
syms x and z... treal	*Converts the variables x, y, z, ..., t to symbolic with real values* >> syms a b c real; >> A = [a b c; c a b; b c a]. A = [a, b, c] [c, a, b] [b, c, a]
syms x and z...t	*Converts the variables x, y, z, ..., t to symbolic*

unreal *variables with non-real values*

syms *List symbolic variables in the workspace*

```
>> syms
'A' 'a' 'ans' 'b' 'c' 'x' 'y' 'a' 'ans' 'b' 'c' 'x' 'y'
```

y=sym('x') *Converts variable or number x to symbolic (equivalent to syms x)*

```
>> rho = sym('(1 + sqrt(5))/2')
rho =
5^(1/2)/2 + 1/2
```

y=sym('x',real) *Converts x to a real symbolic variable*

y=sym('x',unreal) *Converts x to a non-real symbolic variable*

S=sym(A) *Creates a symbolic object from A, where A can be a string, a scalar, a matrix, a numeric expression, etc.*

```
>> S=sym([0.5 0.75 1; 0 0.5 0.1; 0.2 0.3 0.4])
S =
[ 1/2, 3/4, 1]
[ 0, 1/2, 1/10]
[ 1/5, 3/10, 2/5]
```

S = sym(A,'option') *Converts the matrix, scalar or numeric expression A to symbolic according to the specified option. The option can be 'f' for floating point, 'r' for rational, 'e' for error format and 'd' for decimal.*

numeric(x) or double(x) *Converts the variable or expression x to double precision numeric*

pretty(expr) *Converts symbolic expression to mathematical writing*

```
>> pretty(rho)
```

```
1/2
5 1
——+ -
2 2
```

digits

Gives the current accuracy for symbolic variables

```
>> digits
Digits = 32
```

digits(d)

Sets the precision of symbolic variables to d exact decimal digits

```
>> digits(25)
>> digits
Digits = 25
```

vpa('expr')

Numerical result of the expression with the decimal digits of precision placed in digits

```
>> phi = vpa('(1+sqrt(5))/2')
phi =
1.618033988749894848204587
```

vpa('expr', n)

Numerical result of the expression with n decimal digits

```
>> phi = vpa('(1+sqrt(5))/2',10)
phi =
1.618033989
```

vpa(expr, n)

Numerical result of the expression with n decimal digits

```
>> phi = vpa((1+sqrt(5))/2,20)
phi =
1.6180339887498949025
```

1.11 SYMBOLIC FUNCTIONS AND FUNCTIONAL OPERATIONS. COMPOSITE AND INVERSE FUNCTIONS

I n MATLAB it is possible to define custom functions of one and several variables using the syntax *f='function'* .

The syntax *f=function* can also be used if all variables have been previously defined as symbolic with *syms* . It is then possible to make substitutions in their arguments according to the notation presented in the following table.

Results are often simplified with the commands *simplify* and *simple* .

f='function'

Defines the function f as symbolic

```
>> f1='x^3'.
f1 =
x^3
>> f2='z^2+2*2*t'.
f2 =
z^2+2*t
>> syms x z t
>> g1=x^2
g1 =
x^2
>> g2=sqrt(x+2*z+exp(t))
g2 =
(x + 2*z + exp(t))^(1/2)
```

subs(f,a)

Apply the function f at point a

```
>> subs(f1,2)
ans =
8
```

subs(f,

Substitute into the equation of f the variable by the

variable, value)	*value* >> subs(f1,x,3) ans = 27
subs(f, {x,y,...}, {a,b,...})	*Substitute in the equation of f the variables {x,y,...} by the values {a,b,...}.* >> subs(f2, {z,t}, {1,2}) ans = 5

In addition MATLAB implements several functional operations which are summarised in the following table:

f+g	*Sum of functions f and g (f+g)* >> syms x >> f=x^2+x+1;g=2*x^2-x^3+cos(x); h=-x+log(x); >> f+g ans = x + cos(x) + 3*x^2 - x^3 + 1
f+g+h+...	*Perform the sum f+g+h+....* >> f+g+h ans = cos(x) + log(x) + 3*x^2 - x^3 + 1
f-g	*Realise the difference of f and g (f-g)* >> f-g ans = x - cos(x) - x^2 + x^3 + 1
f-g-h-...	*Realise the difference f-g-h-...* >> f-g-h ans = 2*x - cos(x) - log(x) - x^2 + x^3 + 1
f*g	*Perform the product of f and g (f*g).*

```
>> f*g
ans =
(cos(x) + 2*x^2 - x^3)*(x^2 + x + 1)
```

f*g*h*...

*Realise the product f*g*h*...*

```
>> f*g*h
ans =
-(x - log(x))*(cos(x) + 2*x^2 - x^3)*(x^2 + x + 1)
```

f/g

Make the quotient between f and g (f/g).

```
>> f/g
ans =
(x^2 + x + 1)/(cos(x) + 2*x^2 - x^3)
```

f/g/h/...

Make the quotient f/g/h/....

```
>> f/g/h
ans =
-(x^2 + x + 1)/((x - log(x))*(cos(x) + 2*x^2 - x^3))
```

f^k

Raises f to the power k (k is a scalar)

```
>> f^2
ans =
(x^2 + x + 1)^2
```

f^g

Raises a function to another function (f^{g})

```
>> f^g
ans =
(x^2 + x + 1)^(cos(x) + 2*x^2 - x^3)
```

compose(f,g)

Composite function of f and g (f°g(x) = f(g(x)))

```
>> compose(f,g)
ans =
cos(x) + (cos(x) + 2*x^2 - x^3)^2 + 2*x^2 - x^3 + 1
```

compose(f,g,u)

Composite function of f and g, taking the expression u as the domain of f and g

```
>> compose(f,g,h)
ans =
cos(log(x) - x) + 2*(x - log(x))^2 + (x - log(x))^3 + (cos(log(x)
```

g = finverse(f) *Inverse function of f*

```
>> finverse(f)
Warning: finverse(x^2 + x + 1) is not unique.
ans =
(4*x - 3)^(1/2)/2 - 1/2
```

1.12 COMMANDS THAT HANDLE VARIABLES IN THE WORKSPACE AND STORE THEM IN FILES

M ATLAB has a set of commands that allow you to define and manage variables, and to store them in files in MATLAB's own format (extension . *mat*) or in ASCII format, in a very simple way. When performing large calculations, it is convenient to give names to intermediate results. These intermediate results are assigned to variables to make them easier to use. Remember that the value assigned to a variable is permanent, until it is explicitly changed or until you leave the current work session. MATLAB incorporates a set of commands for handling variables, including the following:

clear	*Deletes all variables from the workspace*
clear(v1,v2, ..., vn)	*Deletes the specified numeric variables*
clear('v1', 'v2', ..., 'vn')	*Deletes the specified string variables*
disp (X)	*Displays an array without including its name*
length (X)	*Displays the length of the vector X and, if X is a matrix or array, displays its highest dimension*
load	*Reads all variables from MATLAB.mat file*
load file	*Reads all variables from the specified .mat file*
load file X Y Z	
load file -ascii	

load file -mat	*Read X, Y, Z variables from the given .mat file*
S = load(...)	*Reads the file as ASCII whatever its extension.*
	Reads the file as .mat whatever its extension is
	Returns the contents of a .mat file in the variable S
saveas(h,'f.ext')	*Save the figure or model h as the file f.ext*
saveas(h,'f','format')	*Save the figure or model h as file f with the specified format.*
d = size(X)	*Returns the sizes of each dimension in a vector.*
[m,n] = size(X)	*Returns the dimensions of the matrix X as two variables of names m and n*
[d1,d2,d3,...,dn] = size(X)	*Returns the dimensions of the array X as variables of names d1, d2, ..., dn*
who	*List the workspace variables*
whos	*Lists the workspace variables with their sizes and types*
who('-file','file')	*List the variables in the given .mat file*
whos('-file','file')	*List the variables in the given .mat file and their sizes and types*
who('var1','var2',...)	*Lists the given workspace string variables*

who('-file','filename', 'var1','var2',...)	*Lists the string variables specified in the given .mat file.*
s = who(...)	*Stores in s the listed variables*
s = whos(...)	*Stores in s the variables with their sizes and types*
who -file filename var1 var2 ...	*List the numeric variables specified in the given .mat file*
whos -file filename var1 var2 ...	*Lists the numeric variables specified in the given .mat file with their sizes and types*
workspace	*Displays a browser to manage the workspace*

The *save* command is the essential tool for saving data in MATLAB .*mat* files (readable only by the MATLAB program) and in ASCII files (readable by any application). By default, variables are usually saved in MATLAB .*mat* files. To store variables in ASCII format files, it is necessary to use the *save* command with the following options:

Option	***Data storage mode***
-append	*Variables are added at the end of the file*
-ascii	*The variables are stored in a file in 8-digit ASCII format.*
-ascii -double	*Variables are stored in a 16-digit ASCII format file.*
-ascii -tabs	*Variables are stored in a tab-delimited, 8-digit ASCII file format*

-ascii -double -tabs	*Variables are stored in a 16-digit tab-delimited ASCII format file.*
-mat	*The variables are stored in a MATLAB-formatted binary .mat file (MAT-file).*

E xercise 1. Calculate the value of 7 to the power 400 to the nearest 500 decimal places.

```
>> vpa '7^400' 500

ans =

1.0945006043361130854242544564866621752997548733597061863354194075154390631634920900214785684696871528073999537352825386155249571017070263772889172085286838471044006674397286276116996066357907929105887893308827487569817802497708822339639826555559691647353679243713463273971938996969063052331711311172768319581983900349200609799472931224000 1*10^338
```

In this case, 500 digits are not necessary to express the exact value of the result. You can see that 338 digits are sufficient. If you want the result with fewer exact digits than the actual number, MATLAB completes the result with powers of 10:

```
>> vpa '7^400' 45

ans =

1.09450060433611308542425445648666217529975487*10^338
```

Calculate the greatest common divisor and least common multiple of the numbers 1000, 500 and 625.

```
>> gcd(gcd(1000,500),625)

ans =

125
```

As the function gcd only takes two arguments in MATLAB, we have applied the property: *gcd(a,b,c) = gcd(gcd(a,b),c) = gcd(a,gcd(b,c)).* As the property is analogous for the least common multiple: *lcm(a,b,c) = lcm(lcm(lcm(a,b),c) = lcm(a,lcm(b,c)).* we will calculate it as follows:

```
>> lcm(lcm(1000,500),625)
ans =
5000
```

Is the number 99,991 prime? Find the prime factors of the number 135,678,742.

We decompose both numbers into prime factors:

```
>> factor(99991)
ans =
99991
>> factor(135678742)
ans =
2 1699 39929
```

It is observed that 99991 is a prime number because when decomposed into prime factors it turns out to be the only prime factor of itself. It is also observed that the number 135678742 has three prime factors.

Find the remainder of the division of 2^{134} *by 3. Is the number* $2^{32} - 1$ *divisible by 17? Find also the number N which, when divided by 16, 24, 30 and 32, gives a remainder of 5.*

The first part of the problem is solved as follows:

```
>> rem(2^134,3)
ans =
0
```

It is noted that 2134 is a multiple of 3.

To see if 2^{32} -1 is divisible by 17, we break it down into prime factors:

```
>> factor(2^32-1)
ans =
3 5 17 257 65537
```

It is noted that 17 is one of its factors, so it is divisible by 17.

To solve the last part of the problem, we calculate the least common multiple of all the numbers and add 5 to it.

```
>> N=5+lcm(16,lcm(24,lcm(30,32)))
N =
485
```

Calculate the default value offered by MATLAB for the factorial of 100. Calculate it also with 70 and with 200 significant figures.

```
>> factorial(100)
    ans =
    9.3326e+157
    >> vpa(factorial(100))
    ans =
    9.332621544394410218832560610857 5*10^157
    >> vpa(factorial(100),70)
    ans =
    9.332621544394410218832560610857526724094425485496057150916691040040 8*10^157
    >> vpa(factorial(100),200)
    ans =
    9.33262154439441021883256061085752672409442548549605715091669104004079950642 42
93714863269403045051289804298929694447489825873720431123664147756187701650181 3248
*10^157
```

Exercise 6. Obtain in base 5 the result of the following operation:

a25aaff6 + 6789aba + 35671 + 1100221 - 1250

16 12 8 3

As this is an operation between numbers in numbering systems with different bases, it will be necessary to convert them all to base 10 and carry out the operation to calculate the result in base 10.

```
>> R10=base2dec('a25aaf6',16)+base2dec('6789aba',12)+base2dec('35671',8)
+base2dec('1100221',3)-1250
R10 =
190096544
>> R5=dec2base(R10,5)
R5 =
342131042134
```

Obtain, in base 13, the result of the following operation:

(666551) (aa199800a)+(fffaaa125) / (33331 +6)

7 11 16 4

We use the strategy of the previous exercise. First, we perform the operation with all the numbers in base 10 and finally we convert the result to base 13.

```
>>
R10=vpa(base2dec('666551',7)*base2dec('aa199800a',11)+79*base2dec('fffaaa125',16)/(base2dec('3
3331',4)+6))
R10 =
275373340490851.53125
>> R13=dec2base(275373340490852,13)
R13 =
BA867963C1496
```

Exercise 8. Perform the following operations with rational numbers:

a) 3/5+2/5+7/5

b) 1/2+1/3+1/4+1/5+1/6

c) 1/2-1/3+1/4-1/5+1/6

d) (2/3-1/6)-(4/5+2+1/3)+(4-5/7)

a)

```
" format rat
" 3/5+2/5+7/5
ans =
12/5
```

b)

```
" 1/2+1/3+1/4+1/5+1/6
ans =
29/20
```

c)

```
" 1/2-1/3+1/4-1/5+1/6
ans =
23/60
```

d)

```
» (2/3-1/6)-(4/5+2+1/3)+(4-5/7)
ans =
137/210
```

A lternatively, operations can also be performed as follows:

```
" simplify(sym(3/5+2/5+7/5))
ans =
12/5
» simplify(sym(1/2+1/3+1/4+1/5+1/6))
ans =
29/20
» simplify(sym(1/2-1/3+1/4-1/5+1/6))
ans =
23/60
» simplify(sym((2/3-1/6)-(4/5+2+1/3)+(4-5/7)))
ans =
137/210
```

E xercise 9. Perform the following rational operations:

a) 3/a+2/a+2/a+7/a

b) 1/(2a)+1/(3a)+1/(4a)+1/(5a)+1/(6a)

c) 1/(2a)+1/(3b+1/(4a)+1/(5b)+1/(6c)

When dealing with operations on expressions containing the symbolic variable *a* , it is necessary to prepend the command ***syms a to*** declare the variable *a* as symbolic, and then use *simplify or simple.*

a)

```
"syms a
" simplify(3/a+2/a+2/a+7/a)
ans =
12/a
"3/a+2/a+2/a+7/a
ans =
12/a
>> Ra=simple(3/a+2/a+7/a)
Ra =
12/a
```

b)

```
>> simplify(1/(2*a)+1/(3*a)+1/(4*a)+1/(5*a)+1/(6*a))
ans =
29/(20*a)
>> Rb=simple(1/(2*a)+1/(3*a)+1/(4*a)+1/(5*a)+1/(6*a))
Rb =
29/(20*a)
```

c)

```
>> syms a b c
>> pretty(simplify(1/(2*a)+1/(3*b)+1/(4*a)+1/(5*b)+1/(6*c)))
3 8 1
-—+——+ -—
4 a 15 b 6 c
>> pretty(simple(1/(2*a)+1/(3*b)+1/(4*a)+1/(5*b)+1/(6*c)))
3 8 1
-—+——+ -—
4 a 15 b 6 c
```

Exercise 10. Simplify the following rational operations as much as possible:

a) $(1-a^{9})/(1-a^{3})$

b) $(3a+2a+7a)/(a^{3}+a)$

(c) $1/(1+a)+1/(1+a)^{2}+1/(1+a)^{3}$

(d) $1+a/(a+b)+a^{2}/(a+b)^{2}$

a)

```
>> syms a b
>> pretty(simple((1-a^9)/(1-a^3))))
```

```
6 3
a + a + 1
```

b)

```
>> pretty(simple((3*a+2*a+7*a)/(a^3+a))))
12
```

```
——————

2
a + 1
```

c)

```
>> pretty(simple((1/(1+a)+1/(1+a)^2+1/(1+a)^3))))
2
a + 3 a + 3
```

```
————————————

3
(a + 1)
```

d)

```
>> pretty(simple((1+a/(a+b)+ a^2/(a+b)^2))))
2
2 a + b a
——————————+ 1
2
(a + b)
```

Exercise 11. Perform the following operations with irrational numbers:

a) $3\ \sqrt{a} + 2\ \sqrt{a} - 5\ \sqrt{a} + 7\ \sqrt{a}$

b) $\sqrt{a} + 3\ \sqrt{a} - \sqrt{a}\ /2$

c) $4\ a^{1/3} - 3\ b^{1/3} - 5\ a^{1/3} - 2\ b^{1/3} + ma^{1/3}$

d) $\sqrt{3a}\ \ \sqrt{27a}$

e) $\sqrt{a}\ \ \sqrt{a}$

f) $\sqrt{a\sqrt[5]{a}}$

a)

We use the *simplify* command or the *simple* command.

```
>> syms a b m
>> pretty(simplify(3*sqrt(a) + 2*sqrt(a) - 5*sqrt(a) + 7*sqrt(a))))
1/2
7 a
```

b)

```
>> pretty(simplify(sym(sqrt(2)+3*sqrt(2)-sqrt(2)/2)))))
1/2
7 2
```

```
______

2
```

c)

```
>> syms a b m
>> pretty(simplify(4*a^(1/3)- 3*b^(1/3)-5*a^(1/3)- 2*b^(1/3)+m*a^(1/3)))))
1 1

-            -

3 1/3 3
a m - a - 5 b
```

d)

```
  >> pretty(simplify(sqrt(3*a)*sqrt(27*a))))
  9 a
```

e)

```
  >> pretty(simplify(a^(1/2)*a^(1/3))))
  5

-

6
  a
```

f)

```
  >> pretty(simplify(sqrt(a*(a^(1/5)))))
  / 6 \1/2
  | - |
  | 5 |
  \ a /
```

Carry out the rationalisation of denominators in the following irrational expressions:

$$\frac{2}{\sqrt{2}} \qquad \frac{3}{\sqrt{3}} \; \frac{a}{\sqrt{a}}$$

a) b) c)

I n these cases of rationalisation , the simple use of the *simplify* command solves the problems. The *radsimp* command can also be used.

a)

```
  >> simplify(sym( 2/sqrt(2)))
  ans =
  2^(1/2)
```

b)

```
    >> simplify(sym(3/sqrt(3)))
    ans =
    3^(1/2)
```

c)

```
    >> syms a
    >> simplify(sym(a/sqrt(a)))
    ans =
    a^(1/2)
```

*G iven the vector variables a=[π ,2π ,3 ,π4 ,π5 π] and b=[e,2e,3e,4e,5e] calculate c= Sen(a)+b, d=Cosh(a), e=Ln(b), f=c*d, g=c/d,* $h=d^{2}$ *.*

```
    >> a=[pi,2*pi,3*pi,4*pi,5*pi],b=[exp(1),2*exp(1),3*exp(1),4*exp(1),5*exp(1)]
    a =
    3.1416 6.2832 9.4248 12.5664 15.7080



b =
    2.7183 5.4366 8.1548 10.8731 13.5914
    >> c=sin(a)+b,d=cosh(a),e=log(b),f=c.*d,g=c./d,h=d.^2
    c =
    2.7183 5.4366 8.1548 10.8731 13.5914



d =
    1.0e+006 *
    0.0000 0.0003 0.0062 0.1434 3.3178
```

```
e =

     1.0000 1.6931 2.0986 2.3863 2.6094
```

```
f =

     1.0e+007 *

     0.0000 0.0001 0.0051 0.1559 4.5094
```

```
g =

     0.2345 0.0203 0.0013 0.0001 0.0000
```

```
h =

     1.0e+013 *

     0.0000 0.0000 0.0000 0.0021 1.1008
```

E xercise 14. Given the vector of the first 10 natural numbers, find:

Its sixth element

Its elements from the fourth to the seventh inclusive

Its elements from the second to the ninth inclusive in threes

The elements of the previous section, but in descending order of importance.

" x=(1:10)

```
x =

1 2 3 4 5 6 7 8 9 10
```

```
 " x(6)
ans =
6
```

We have obtained the sixth element of the vector x.

```
" x(4:7)
ans =
4 5 6 7
```

We have obtained the elements of the vector x located between the fourth and seventh, inclusive.

```
" x(2:3:9)
ans =
2 5 8
```

We have obtained the elements of the vector x located between the second and the ninth, both inclusive, but separated by three units.

```
" x(9:-3:2)
ans =
9 6 3
```

We have obtained the elements of the vector *x* located between the ninth and the second, both inclusive, but separated by three units and starting with the ninth.

Given a 2x3 matrix whose rows are the first six consecutive odd numbers, ask:

Cancel its element (2,3), transpose it and attach the identity matrix of order 3 to its right-hand side.

If we call C the matrix obtained above, construct a matrix D by extracting the odd columns of matrix C, a matrix E formed by the intersection of the first two rows of C and its third and fifth columns, and a *matrix F formed by the intersection of the first two rows and the last three columns of matrix C.*

Construct the diagonal matrix G such that the elements of its main diagonal are the same as those of the main diagonal of D.

We construct the matrix H, formed by the intersection of the first and third rows of C and its second, third and fifth columns.

We first consider the *2x3* matrix whose rows are the first 6 consecutive odd numbers:

```
" A=[1 3 5;7 9 11]

A =

1 3 5

7 9 11
```

Now we are going to cancel the element *(2,3)* , that is, its last element:

```
" A(2,3)=0

A =

1 3 5

7 9 0
```

Next, we consider the matrix *B* transposed from *A* :

```
"B=A' B=A' B=A' B=A' B=A' B=A' B=A'

B =

1 7

3 9

5 0
```

We now construct a matrix *C* , formed by the matrix *B* and the identity matrix of order 3 attached to its right:

```
"C=[B eye(3)] C=[B eye(3)]

C =

1 7 1 0 0
```

```
   3 9 0 1 0
   5 0 0 0 1
```

We are going to construct a matrix *D* by extracting the odd columns of the matrix *C* , a matrix *E* formed by the intersection of the first two rows of *C* and its third and fifth columns, and a matrix *F* formed by the intersection of the first two rows and the last three columns of the matrix *C* :

```
   " D=C(::,1:2:5)
   D =
   1 1 0
   3 0 0
   5 0 1
   " E=C([1 2],[3 5])
   E =
   1 0
   0 0


" F=C([1 2],3:5)
   F =
   1 0 0
   0 1 0
```

We now construct the diagonal matrix *G* such that the elements of its main diagonal are the same as those of the main diagonal of *D* :

```
   " G=diag(diag(D))
   G =
   1 0 0
   0 0 0
   0 0 1
```

Next , we construct the matrix *H* , formed by the intersection of the first and third rows of *C* and its second, third and fifth columns:

```
" H=C([1 3],[2 3 5])
H =
7 1 0
0 0 1
```

C onstruct a matrix I formed by the identity matrix of order 5x4 and the null and unit matrices of the same order attached to its right. Then, extract the first row of I and, finally, form the matrix J with the odd rows and even columns of Y and calculate its order (size).

Additionally , construct a random matrix K of order 3x4 and invert first the order of its rows, then the order of its columns and then the order of its rows and columns at the same time. Finally, find the matrix L of order 4x3 whose columns result from taking the elements of the columns of K consecutively.

```
" D=C(::,1:2:5)
D =
1 1 0
3 0 0
5 0 1
" E=C([1 2],[3 5])
E =
1 0
0 0
```

```
" F=C([1 2],3:5)
    F =
    1 0 0
    0 1 0
```

We now construct the diagonal matrix *G* such that the elements of its main diagonal are the same as those of the main diagonal of *D* :

```
    " G=diag(diag(D))
    G =
    1 0 0
    0 0 0
    0 0 1
```

Next , we construct the matrix *H* , formed by the intersection of the first and third rows of *C* and its second, third and fifth columns:

```
    " H=C([1 3],[2 3 5])
    H =
    7 1 0
    0 0 1
```

We now construct a matrix *I* consisting of the identity matrix of order *5x4* and the null and unit matrices of the same order attached to its right. Next, we extract the first row of *I* and, finally, we form the matrix *J* with the odd rows and even columns of *Y* and calculate its order (size).

```
    "I = [eye(5,4) zeros(5,4) ones(5,4)].
    ans =
   1 0 0 0 0 0 0 0 1 1 1 1
   0 1 0 0 0 0 0 0 1 1 1 1
   0 0 1 0 0 0 0 0 1 1 1 1
   0 0 0 1 0 0 0 0 1 1 1 1
   0 0 0 0 0 0 0 0 1 1 1 1
    " I(1,:)
    ans =
   1 0 0 0 0 0 0 0 1 1 1 1
```

```
" J=I(1:2:5,2:2:12)
J =
0 0 0 0 1 1
0 0 0 0 1 1
0 0 0 0 1 1
" size(J)
ans =
3 6
```

Next, we construct a random matrix *K* of order *3x4* and reverse first the order of its rows, then the order of its columns and then the order of its rows and columns at the same time. Finally, we find the matrix *L* of order *4x3* whose columns result from taking the elements of the columns of *K* consecutively.

```
"K=rand(3,4)
K =
0.5269 0.4160 0.7622 0.7361
0.0920 0.7012 0.2625 0.3282
0.6539 0.9103 0.0475 0.6326
" K(3:-1:1,:)
ans =
0.6539 0.9103 0.0475 0.6326
0.0920 0.7012 0.2625 0.3282
0.5269 0.4160 0.7622 0.7361
" K(:,4:-1:1)
ans =
0.7361 0.7622 0.4160 0.5269
0.3282 0.2625 0.7012 0.0920
0.6326 0.0475 0.9103 0.6539
" K(3:-1:1,4:-1:1)
ans =
0.6326 0.0475 0.9103 0.6539
0.3282 0.2625 0.7012 0.0920
0.7361 0.7622 0.4160 0.5269
"L=reshape(K,4,3)
L =
0.5269 0.7012 0.0475
0.0920 0.9103 0.7361
0.6539 0.7622 0.3282
0.4160 0.2625 0.6326
```

Given the square matrix of order 3, whose rows are the first 9 natural numbers, obtain its inverse, its transpose and its diagonal. Transform it into a lower triangular matrix and an upper triangular matrix and rotate it 90 degrees. Obtain the sum of the elements of the first row and the sum of the elements of the diagonal. Extract the submatrix whose diagonal are the elements a_{11} and a and $_{22}$ also extract the submatrix whose diagonal elements are a_{11} and a_{33}.

```
" M=[1,2,3;4,5,6;7,8,9]
M =
1 2 3
4 5 6
7 8 9
"A=inv(M)
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 2.937385e-018
A =
1.0e+016 *
0.3152 -0.6304 0.3152
-0.6304 1.2609 -0.6304
0.3152 -0.6304 0.3152
" B=M'
```

```
B =

1 4 7

2 5 8

3 6 9

"V=diag(M)

V =

1

5

9

"TI=tril(M)

TI =

1 0 0

4 5 0

7 8 9

"TS=triu(M)

TS =

1 2 3

0 5 6

0 0 9

"TR=rot90(M)

TR =

3 6 9

2 5 8

1 4 7

"s=M(1,1)+M(1,2)+M(1,3)

s =

6

"sd=M(1,1)+M(2,2)+M(3,3)

sd =

15

" SM=M(1:2,1:2)
```

```
SM =
1 2
4 5
"SM1=M([1 3],[1 3])
SM1 =
1 3
7 9
```

Given the square matrix of order 3, whose rows are the first 9 natural numbers (without zero), identify its values less than 5.

If we now consider the vector whose elements are the first 9 natural numbers (without zero) and the vector with the same elements from largest to smallest, identify the values resulting from subtracting from the elements of the second vector the quantity 1 if the corresponding element of the first vector is greater than 2, or the quantity 0 if it is less than or equal to 2.

```
"X=5*ones(3,3); X>= [1 2 3; 4 5 6; 7 8 9].
ans =
1 1 1
1 1 0
0 0 0
```

The elements of the matrix *X* that are greater than or equal to those of the matrix [1 2 3; 4 5 6; 7 8 9] correspond to a 1 in the answer matrix. The rest of the elements correspond to a 0 (the result of the operation would be the same if we compare the scalar 5 with the matrix [1 2 3; 4 5 6; 7 8 9], by means of the expression " *X=5* ; *X*>= [1 2 3; 4 5 6; 7 8 9]).

```
>> X=5; X>= [1 2 3; 4 5 6; 7 8 9]
ans =
1 1 1
1 1 0
0 0 0
```

Next, we solve the second part of the exercise:

```
>> A=1:9,B=10-A,Y=A>4,Z=B-(A>2)
A =
1 2 3 4 5 6 7 8 9
B =
9 8 7 6 5 4 3 2 1
Y =
0 0 0 0 1 1 1 1 1
Z =
9 8 6 5 4 3 2 1 0
```

T he values of *Y* equal to 1 correspond to the elements of A greater than 4. The values of *Z* result from subtracting from the corresponding elements of *B* the quantity 1 if the corresponding element of *A* is greater than 2, or the quantity 0 if the corresponding element of *A* is less than or equal to 2.

If we now consider the vector A whose elements are the first 9 natural numbers (without zero), identify its values greater than 2 and less than 6.

If we call P the vector of the same dimension as A that associates a 1 with the outcomes identified in the previous section and a zero with the rest, identify when A or P, but not both, are worth 1.

```
" A=1:9;P=(A>2)&(A<6 )
P =
0 0 1 1 1 0 0 0 0
```

R eturns 1 when *A* is greater than 2 and less than 6, and returns 0 otherwise.

Now we solve the second part of the problem.

```
    >> P=(A>=1)&(A<6),xor(A,P)
    P =
    1 1 1 1 1 0 0 0 0



a ns =
    0 0 0 0 0 1 1 1 1
```

The ones in the last result are the requested solution.

Find the difference matrix between a square random matrix of order 4 and a normal random matrix of order 4. Calculate the transpose and the inverse of the above difference. Check that the inverse is correctly calculated.

```
    >> A=rand(4)-randn(4)
    A =
    0.9389 -0.0391 0.4686 0.6633
    -0.5839 1.3050 -0.0698 1.2727
    -1.2820 -0.4387 -0.5693 -0.0881
    -0.5038 -1.0834 1.2740 1.2890
```

We calculate the inverse and multiply it by the initial matrix to check that the matrix identity results.

```
    >> B=inv(A)
    B =
    0.9630 -0.1824 0.1288 -0.3067
    -0.8999 0.4345 -0.8475 -0.0239
    -1.6722 0.0359 -1.5242 0.7209
    1.2729 0.2585 0.8445 -0.0767
```

```
>> A*B
ans =
1.0000 0.0000 0 -0.0000
0 1.0000 0.0000 -0.0000
-0.0000 -0.0000 1.0000 0.0000
0.0000 -0.0000 0.0000 1.0000
>> B*A
ans =
1.0000 0.0000 0.0000 0.0000
-0.0000 1.0000 -0.0000 0.0000
0.0000 -0.0000 1.0000 0.0000
-0.0000 -0.0000 0 1.0000
>> A'
ans =
0.9389 -0.5839 -1.2820 -0.5038
-0.0391 1.3050 -0.4387 -1.0834
0.4686 -0.0698 -0.5693 1.2740
0.6633 1.2727 -0.0881 1.2890
```

Exercise 21. Given the function f(x) = x^{3} *calculate f(2) and f(b+2). We now consider the function of two variables f (a,b) = a+b* and we want to calculate f(*4,b*), *f* (*a,5*) and (*f* (3,5).

```
>> f='x^3'.
f =
x^3
>> A=subs(f,2)
A =
8
>> syms b
>> B=subs(f,b+2)
B =
(b+2)^3
>> syms a b
>> subs(a+b,a,4)
ans =
4+b
>> subs(a+b,{a,b},{3,5})
ans =
8
```

In the following example , given the functions f (x) =x^{2} +x, g (x) =x^{3} +1 and h (x) = Sen (x) +Cos (x), calculate (f+g)(x) , (f-g+h)(x) , (f/g) (x) , f (g (x-1)) , f (h (π/3)) and f (g(h (Sen (x)))))).

```
>> syms x
>> f=x^2;g=x^3+1;h=sin(x)+cos(x);
>> sum=f+g
sum =
x^2+x^3+1
>> combined= f-g+h
combined =
x^2-x^3-1+sin(x)+cos(x)
>> quotient=f/g
quotient =
x^2/(x^3+1)
>> composite=subs(compose(g,f),x-1)
compound =
(x-1)^6+1
>> compose1=subs(compose(f,h),pi/3)
composite1 =
1.8660
```

```
>> compose1=subs(compose(f,h),'pi/3')
composite1 =
(sin((pi/3))+cos((pi/3)))^2
>> composite2=subs(compose(f,compose(g,h)),'sin(x)')
compound2 =
((sin(sin(x))+cos(sin(x)))^3+1)^2
```

Find the inverse function of $f (x) = Sen (x^{2})$ *and check that the result is correct .*

```
>> syms x
>> f=sin(x^2)
f =
sin(x^2)
>> g=finverse(f)
g =
asin(x)^(1/2)
>> compose(f,g)
ans =
x
```

Given the functions $f(x)=sin(cos(x^{1/2})$ *and* $g(x)=sqrt(tan(x^{2}))$ *calculate the composite of f and g and the composite of g and f. Calculate also the inverse functions of f and g.*

```
>> syms x, f= sin(cos(x^(1/2)));
>> g=sqrt(tan(x^2));
>> simple(compose(f,g))
ans =
sin(cos(cos(tan(x^2)^(1/4))))
>> simple(compose(g,f))
ans =
tan(sin(sin(cos(cos(x^(1/2)))))^2)^(1/2)
>> F=finverse(f)
```

```
F =
acos(asin(x))^2
>> G=finverse(g)
G =
atan(x^2)^(1/2)
```

Given the function h defined by: h(x,y)= (cos(x^2 -y^2), sin(x^2 -y^2)); calculate h(1,2), h(-Pi,Pi) and h(cos(a^2), cos(1-a^2)).

```
>> syms x y a
>> h=[cos(x^2-y^2), sin(x^2-y^2)].
h =
[ cos(x^2 - y^2), sin(x^2 - y^2)].
>> subs(h,{x,y},{1,2})
ans =
-0.9900 -0.1411
>> subs(h,{x,y},{-pi,pi})
ans =
1 0
>> pretty(simple(subs(h,{x,y},{cos(a^2),cos(1-a^2)}))))))
+- 2 2 2 2 2 2 2 2 -+
| cos(- cos(1 - a ) + cos(a ) ), sin(- cos(1 - a ) + cos(a ) ) | |
+- -+
```

Generate a vector that is a row matrix of 10 uniform random numbers and generate another vector that is the matrix of ones of order 10. Save both vectors as variables in a MATLAB format file and in another ASCII format file. Finally, read the two previously saved files into MATLAB.

```
>> p = rand(1, 10)
p =
Columns 1 through 9
0.6797 0.6551 0.1626 0.1190 0.4984 0.9597 0.3404 0.5853 0.2238
Column 10
```

```
   0.7513
   >> q = ones(10)
   q =
   1 1 1 1 1 1 1 1 1 1
   1 1 1 1 1 1 1 1 1 1
   1 1 1 1 1 1 1 1 1 1
   1 1 1 1 1 1 1 1 1 1
   1 1 1 1 1 1 1 1 1 1
   1 1 1 1 1 1 1 1 1 1
   1 1 1 1 1 1 1 1 1 1
   1 1 1 1 1 1 1 1 1 1
   1 1 1 1 1 1 1 1 1 1
   1 1 1 1 1 1 1 1 1 1
```

We now save the above results as variables in a MATLAB format file (extension . *mat*) named *file1* . We also save the above results as variables in a file in ASCII format (extension . *txt*) named *file2.txt* .

```
> save('file1', 'p' , 'q')
   >> save file1 p q q
   >> save('file2.txt', 'p', 'q', '-ASCII')
   >> save file2.txt p q -ASCII
```

These files could then be read and their variables incorporated into the workspace as follows:

```
   >> load file2.txt -ASCII
   >> load file1
```

Chapter 2.

COMPLEX NUMBERS AND COMPLEX VARIABLE FUNCTIONS

2.1 COMPLEX NUMBERS

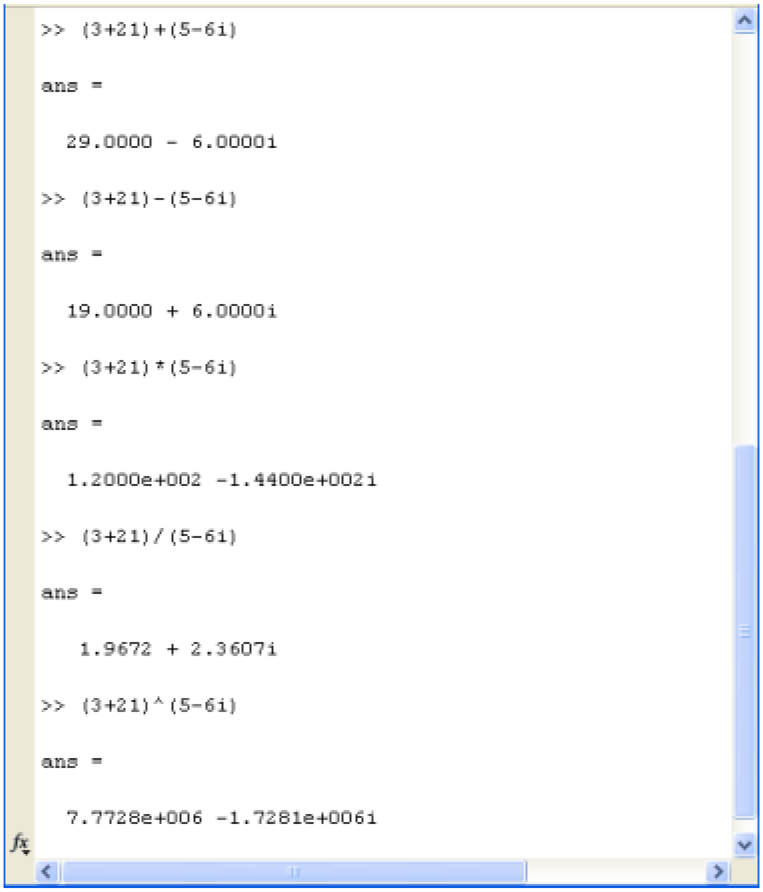
M ATLAB implements in a simple way the work with complex numbers in binomial form by representing the imaginary unit by the symbols *i* or *j* .

We already know that a complex number in binomial form is represented in the form *a+bi* or *a+bj* . It is not necessary to include the product symbol (the asterisk) before the imaginary unit, although if it is included, everything works correctly. What you do have to bear in mind is that you cannot insert spaces between the imaginary part number and the imaginary unit *i* .

Complex numbers can have symbolic real part or symbolic imaginary part or both, with operations being performed in either exact or decimal mode with the precision set with the *format* command. Therefore, it will also be possible to work with complex numbers in exact rational format through the *format rat* command.

As for the usual operations (addition, difference, product, division and power) with complex numbers, they are performed in the usual way. Figure 2-1 shows examples.

Obviously, since the real numbers are a subset of the complex numbers, the functions of complex variables will also be valid for real variables.



```
>> (3+21)+(5-6i)

ans =

  29.0000 - 6.0000i

>> (3+21)-(5-6i)

ans =

  19.0000 + 6.0000i

>> (3+21)*(5-6i)

ans =

  1.2000e+002 -1.4400e+002i

>> (3+21)/(5-6i)

ans =

   1.9672 + 2.3607i

>> (3+21)^(5-6i)

ans =

  7.7728e+006 -1.7281e+006i
```

Figure 2-4

2.2 GENERAL COMPLEX VARIABLE FUNCTIONS

M ATLAB has a range of general predefined complex variable functions, which will of course be valid for rational, irrational and integer variables. The most important of these are presented in the following paragraphs.

2.2.1 TRIGONOMETRIC FUNCTIONS OF COMPLEX VARIABLE

B elow is a table with the trigonometric functions of complex variable and their inverses that MATLAB incorporates, illustrated with examples.

Function	***Inverse***
sin(z) *Sine*	**asin(z)** *Sine arc*
>> sin(5-6i) ans = -1.9343e+002 -5.7218e+001i	>> asin(1-i) ans = 0.6662 - 1.0613i
cos(z) *Cosine*	**acos(z)** *Arc cosine*
>> cos(3+4i) ans = -27.0349 - 3.8512i	>> acos(-i) ans = 1.5708 + 0.8814i
tan(z) *Tangent*	**atan(z) and atan2(z)** *Tangent arc*
>> tan(pi/4i) ans = 0 - 0.6558i	>> atan(-pi*i) ans = 1.5708 - 0.3298i
csc(z) *Cosecant*	**acsc(z)** *Cosecant arc*
>> csc(1-i) ans = 0.6215 + 0.3039i	>> acsc(2i) ans = 0 - 0.4812i
sec(z) *Secant*	**asec(z)** *Secant arc*
>> sec(-i) ans = 0.6481	>> asec(0.6481+0i) ans = 0 + 0.9999i
cot(z) *Cotangent*	**acot(z)** *Cotangent arc*
>> cot(-j) ans = 0 + 1.3130i	>> acot(1-6j) ans = 0.0277 + 0.1635i

2.2.2 HYPERBOLIC FUNCTIONS OF COMPLEX VARIABLE

B elow is a table with the complex variable hyperbolic functions and their inverses that MATLAB incorporates, illustrated with examples.

Function	***Inverse***
sinh(z) *Hyperbolic sine*	**asinh(z)** *Hyperbolic sine arc*
>> sinh(1+i) ans = 0.6350 + 1.2985i	>> asinh(0.6350 + 1.2985i) ans = 1.0000 + 1.0000i
cosh(z) *Hyperbolic cosine*	**acosh(z)** *Hyperbolic cosine arc*
>> cosh(1-i) ans = 0.8337 - 0.9889i	>> acosh(0.8337 - 0.9889i) ans = 1.0000 - 1.0000i
tanh(z) *Hyperbolic tangent*	**atanh(z)** *Hyperbolic tangent arc*
>> tanh(3-5i) ans = 1.0042 + 0.0027i	>> atanh(3-41) ans = -0.0263 - 1.5708i
csch(z) *Hyperbolic cosecant*	**acsch(z)** *Hyplebolic cosecant arc*
>> csch(i) ans = 0 - 1.1884i	>> acsch(- 1.1884i) ans = 0 + 1.0000i
sech(z) *Hyperbolic secant*	**asech(z)** *Hyperbolic secant arc*
>> sech(i^i) ans = 0.9788	>> asech(5-0i) ans = 0 + 1.3694i
coth(z) *Hyperbolic cotangent*	**acoth(z)** *Cotangent hyperbolic arc*
>> coth(9+i) ans = 1.0000 - 0.0000i	>> acoth(1-i) ans = 0.4024 + 0.5536i

2.2.3 EXPONENTIAL AND LOGARITHMIC FUNCTIONS OF COMPLEX VARIABLE

Below is a table of the exponential and logarithmic functions that MATLAB incorporates, illustrated with examples.

Function	***Meaning***
exp(z)	*Exponential function in base e (e^x)* >> exp(1-i) ans = 1.4687 - 2.2874i
log(x)	*Logarithm function in base e of x* >> log(1.4687 - 2.2874i) ans = 1.0000 - 1.0000i
log10(x)	*Logarithm function in base 10 of x* >> log10(100+100i) ans = 2.1505 + 0.3411i
log2(x)	*Logarithm function in base 2 of x* >> log2(4-6i) ans = 2.8502 - 1.4179i
pow2(x)	*Function power of base 2 of x* >> pow2(2.8502 - 1.4179i) ans = 3.9998 - 6.0000i
sqrt(x)	*Square root function of x* >> sqrt(1+i) ans =

```
1.0987 + 0.4551i
```

image image
image

2.2.4 Specific complex variable functions

image

M ATLAB incorporates a group of complex variable functions specifically for working with modules, arguments, and real and imaginary parts. These functions include the following:

Function	Meaning
abs(Z)	Z-complex module >> abs(12.425-8.263i) ans = 14.9217
angle(Z)	Z-complex argument >> angle(12.425-8.263i) ans = -0.5869
conj(Z)	Z-complex conjugate

	>> conj(12.425-8.263i) ans = 12.4250 + 8.2630i
real(Z)	Real part of complex Z >> real(12.425-8.263i) ans = 12.4250
imag(Z)	Imaginary part of complex Z >> imag(12.425-8.263i) ans = -8.2630
floor(Z)	Applies the floor function to real(Z) and imag(Z) >> floor(12.425-8.263i) ans = 12.0000 - 9.0000i
ceil(Z)	Applies the ceil function to real(Z) and imag(Z) >> ceil(12.425-8.263i) ans =

	13.0000 - 8.0000i
round(Z)	Applies round function to real(Z) and imag(Z) >> round(12.425-8.263i) ans = 12.0000 - 8.0000i
fix(Z)	Applies fix function to real(Z) and imag(Z) >> fix(12.425-8.263i) ans = 12.0000 - 8.0000i

image image
image

2.3 BASIC FUNCTIONS WITH ARGUMENT A COMPLEX VECTOR

image

M ATLAB allows you to work with vector and even matrix complex variable functions. Let us not forget that these functions are also valid for real variables, since real numbers are a particular case of complex numbers with zero imaginary part. Below is a table with the specific vector complex variable functions that MATLAB incorporates. Later, when the matrix complex variable functions are presented, we will see that all of them are also valid for vector variables, since a vector is a particular case of a matrix.

max(V)	Largest component (max(abs(V)) is calculated for complexes) >> max([1-i 1+i 3-5i 6i]) ans = 0 + 6.0000i >> max([1, 0, -23, 12, 16]) ans = 16

min(V) Smallest component (for complexes calculate min(abs(V)))

```
>> min([1-i 1+i 3-5i 6i])

ans =

   1. - 1.0000i

>> min([1, 0, -23, 12, 16])

ans =

-23
```

Mean of the components of V

```
          >> mean([1-i 1+i 3-5i 6i])

          ans =

mean(V)   1.2500 + 0.2500i

          >> mean([1, 0, -23, 12, 16])

          ans =

          1.2000
```

median(V) Median of the components of V

```
>> median([1-i 1+i 3-5i 6i])

ans =

2.0000 - 2.0000i

>> median([1, 0, -23, 12, 16])
```

```
ans =

1
```

Standard deviation of the components of V

```
>> std([1-i 1+i 3-5i 6i])

ans =
```

std(V)

```
4.7434

>> std([1, 0, -23, 12, 16])

ans =

15.1888
```

sort(V) Sort in ascending order the components of V. For complexes, sort according to absolute values.

```
>> sort([1-i 1+i 3-5i 6i])

ans =

Columns 1 through 2

1.0000 - 1.0000i 1.0000 + 1.0000i

Columns 3 through 4

3.0000 - 5.0000i 0 + 6.0000i

>> sort([1, 0, -23, 12, 16])

ans =
```

	-23 0 1 12 16
sum(V)	Sum of the components of V >> sum([1-i 1+i 3-5i 6i]) ans = 5.0000 + 1.0000i >> sum([1, 0, -23, 12, 16]) ans = 6
prod(V)	Multiplies the elements of V, so that n ! = prod(1:n) >> prod([1-i 1+i 3-5i 6i]) ans = 60.0000 +36.0000i >> prod([1, 0, -23, 12, 16]) ans = 0
cumsum(V)	Gives the vector of cumulative sums of V >> cumsum([1-i 1+i 3-5i 6i]) ans =

```
Columns 1 through 2

1.0000 - 1.0000i 2.0000

Columns 3 through 4

5.0000 - 5.0000i 5.0000 + 1.0000i

>> cumsum([1, 0, -23, 12, 16])

ans =

1 1 -22 -10 6
```

cumprod(V) Gives the vector of cumulative products of V

```
>> cumprod([1-i 1+i 3-5i 6i])

ans =

Columns 1 through 2

1.0000 - 1.0000i 2.0000

Columns 3 through 4

6.0000 -10.0000i 60.0000 +36.0000i

>> cumprod([1, 0, -23, 12, 16])

ans =

1 0 0 0 0
```

diff(V) Gives the vector of first differences of the components of V ($V_t - V_{t-1}$)

	>> diff([1-i 1+i 3-5i 6i]) ans = 0 + 2.0000i 2.0000 - 6.0000i -3.0000 +11.0000i >> diff([1, 0, -23, 12, 16]) ans = -1 -23 35 4
gradient(V)	Approximates the gradient of V > gradient([1-i 1+i 3-5i 6i]) ans = Columns 1 through 3 0 + 2.0000i 1.0000 - 2.0000i -0.5000 + 2.5000i Column 4 -3.0000 +11.0000i >> gradient([1, 0, -23, 12, 16]) ans = -1.0000 -12.0000 6.0000 19.5000 4.0000
del2(V)	Laplacian of V (discrete 5-point) >> del2([1-i 1+i 3-5i 6i]) ans =

```
Columns 1 through 3

2.2500 - 8.2500i 0.5000 - 2.0000i -1.2500 + 4.2500i

Column 4

-3.0000 +10.5000i

>> del2([1, 0, -23, 12, 16])

ans =

-25.5000 -5.5000 14.5000 -7.7500 -30.0000
```

fft(V) Discrete Fourier transform of V

```
>> fft([1-i 1+i 3-5i 6i])

ans =

Columns 1 through 3

5.0000 + 1.0000i -7.0000 + 3.0000i 3.0000 -13.0000i

Column 4

3.0000 + 5.0000i

>> fft([1, 0, -23, 12, 16])

ans =

Columns 1 through 3

6.0000 14.8435 +35.7894i -15.3435 -23.8824i

Columns 4 through 5

-15.3435 +23.8824i 14.8435 -35.7894i
```

fft2(V)	Two-dimensional discrete Fourier transform of V >> fft2([1-i 1+i 3-5i 6i]) ans = Columns 1 through 3 5.0000 + 1.0000i -7.0000 + 3.0000i 3.0000 -13.0000i Column 4 3.0000 + 5.0000i >> fft2([1, 0, -23, 12, 16]) ans = Columns 1 through 3 6.0000 14.8435 +35.7894i -15.3435 -23.8824i Columns 4 through 5 -15.3435 +23.8824i 14.8435 -35.7894i
ifft(V)	Inverse of the discrete Fourier transform of V >> ifft([1-i 1+i 3-5i 6i]) ans = Columns 1 through 3 1.2500 + 0.2500i 0.7500 + 1.2500i 0.7500 - 3.2500i

```
Column 4

-1.7500 + 0.7500i

>> ifft([1, 0, -23, 12, 16])

ans =

Columns 1 through 3

1.2000 2.9687 - 7.1579i -3.0687 + 4.7765i

Columns 4 through 5

-3.0687 - 4.7765i 2.9687 + 7.1579i
```

ifft2(V) Inverse of the discrete 2-D Fourier transform of V

```
>> ifft2([1-i 1+i 3-5i 6i])

ans =

Columns 1 through 3

1.2500 + 0.2500i 0.7500 + 1.2500i 0.7500 - 3.2500i

Column 4

-1.7500 + 0.7500i

>> ifft2([1, 0, -23, 12, 16])

ans =

Columns 1 through 3

1.2000 2.9687 - 7.1579i -3.0687 + 4.7765i

Columns 4 through 5
```

-3.0687 - 4.7765i 2.9687 + 7.1579i

image image
image

2.4 BASIC FUNCTIONS WITH ARGUMENT A COMPLEX MATRIX

image

T he functions in the previous table also admit a complex matrix Z as an argument, in which case the result is a row vector whose components are the results of applying the function to each column of the matrix. Let us not forget that these functions are also valid for real variables, since real numbers are a particular case of complex numbers with zero imaginary part.

max(Z) Vector with the largest components (for complexes max(abs(Z))) of each column of the matrix Z is calculated.

```
>> Z=[1-i 3i 5;-1+i 0 2i;6-5i 8i -7]

Z =

1.0000 - 1.0000i 0 + 3.0000i 5.0000

-1.0000 + 1.0000i 0 0 0 + 2.0000i

6.0000 - 5.0000i 0 + 8.0000i -7.0000i

>> Z=[1-i 3i 5-12i;-1+i 0 2i;6-5i 8i -7+6i]

Z =
1.0000 - 1.0000i 0 + 3.0000i 5.0000 -12.0000i

-1.0000 + 1.0000i 0 0 0 + 2.0000i

6.0000 - 5.0000i 0 + 8.0000i -7.0000 + 6.0000i

>> max(Z)

ans =

6.0000 - 5.0000i 0 + 8.0000i 5.0000 -12.0000i

>> Z1=[1 3 5;-1 0 2;6 8 -7]

Z1 =

1 3 5

-1 0 2

6 8 -7

>> max(Z1)

ans =

6 8 5
```

min(Z) Vector with the smallest components (for complexes min(abs(Z))) of each column of the matrix Z is calculated.

```
>> min(Z)

ans =

1.0000 - 1.0000i 0 0 0 + 2.0000i

>> min(Z1)
```

```
ans =

-1 0 -7
```

Vector of averages of the column components of Z

mean(Z)

```
>> mean(Z)

ans =

2.0000 - 1.6667i 0 + 3.6667i -0.6667 - 1.3333i

>> mean(Z1)

ans =

2.0000 3.6667 0
```

Vector of medians of the components of the columns of Z

median(Z)

```
>> median(Z)

ans =

-1.0000 + 1.0000i 0 + 3.0000i -7.0000 + 6.0000i

>> median(Z1)

ans =

1 3 2
```

std(Z) Vector of standard deviations of the column components of Z

```
>> std(Z)
```

```
ans =

4.7258 4.0415 11.2101

>> std(Z1)

ans =

3.6056 4.0415 6.2450
```

sort(Z)

Sorts the components of the columnsd of Z in ascending order. For complexes it sorts according to absolute values.

```
>> sort(Z)

ans =

1.0000 - 1.0000i 0 0 0 + 2.0000i

-1.0000 + 1.0000i 0 + 3.0000i -7.0000 + 6.0000i

6.0000 - 5.0000i 0 + 8.0000i 5.0000 -12.0000i

>> sort(Z1)

ans =

-1 0 -7

1 3 2

6 8 5
```

sum(Z)

Vector with the sum of the components of the columns of the matrix Z

```
>> sum(Z)
```

```
ans =

6.0000 - 5.0000i 0 +11.0000i -2.0000 - 4.0000i

>> sum(Z1)

ans =

6 11 0
```

prod(Z)

Vector with the products of the column elements of Z

```
> prod(Z)

ans =

1.0e+002 *

0.1000 + 0.1200i 0 -2.2800 + 0.7400i

>> prod(Z1)

ans =

-6 0 -70
```

cumsum(Z) Gives the matrix of cumulative sums of the columns of Z

```
>> cumsum(Z)

ans =

1.0000 - 1.0000i 0 + 3.0000i 5.0000 -12.0000i

0 0 0 + 3.0000i 5.0000 -10.0000i

6.0000 - 5.0000i 0 +11.0000i -2.0000 - 4.0000i
```

```
>> cumsum(Z1)

ans =

1 3 5

0 3 7

6 11 0
```

cumprod(V)

Gives the cumulative product matrix of the columns of Z

```
>> cumprod(Z)

ans =

1.0e+002 *

0.0100 - 0.0100i 0 + 0.0300i 0.0500 - 0.1200i

0 + 0.0200i 0 0.2400 + 0.1000i

0.1000 + 0.1200i 0 -2.2800 + 0.7400i

>> cumprod(Z1)

ans =

1 3 5

-1 0 10

-6 0 -70
```

diff(Z)

Gives the matrix of first differences of the column components of Z

```
>> diff(Z)

ans =

-2.0000 + 2.0000i 0 - 3.0000i -5.0000 +14.0000i

7.0000 - 6.0000i 0 + 8.0000i -7.0000 + 4.0000i

>> diff(Z1)

ans =

-2 -3 -3

7 8 -9
```

Approximates the gradient matrix of the columns of Z

```
>> gradient(Z)

ans =

-1.0000 + 4.0000i 2.0000 - 5.5000i 5.0000 -15.0000i

1.0000 - 1.0000i 0.5000 + 0.5000i 0 + 2.0000i

gradient(Z) -6.0000 +13.0000i -6.5000 + 5.5000i -7.0000 - 2.0000i

>> gradient(Z1)

ans =

2.0000 2.0000 2.0000

1.0000 1.5000 2.0000

2.0000 -6.5000 -15.0000
```

del2(V) Approximates the Laplacian matrix of the columns of Z

```
>> del2(Z)

ans =

3.7500 - 6.7500i 1.5000 - 2.0000i 1.0000 - 7.2500i

2.0000 - 1.2500i -0.2500 + 3.5000i -0.7500 - 1.7500i

2.0000 - 5.7500i -0.2500 - 1.0000i -0.7500 - 6.2500i

>> del2(Z1)

ans =

2.2500 2.7500 -1.5000

2.5000 3.0000 -1.2500

-2.0000 -1.5000 -5.7500
```

fft(Z) Matrix with the discrete Fourier transforms of the columns of Z

```
>> fft(Z)

ans =

6.0000 - 5.0000i 0 +11.0000i -2.0000 - 4.0000i

3.6962 + 7.0622i -6.9282 - 1.0000i 5.0359 -22.0622i

-6.6962 - 5.0622i 6.9282 - 1.0000i 11.9641 - 9.9378i

>> fft(Z1)

ans =
6.0000 11.0000 0

-1.5000 + 6.0622i -1.0000 + 6.9282i 7.5000 - 7.7942i

-1.5000 - 6.0622i -1.0000 - 6.9282i 7.5000 + 7.7942i
```

fft2(Z)

Matrix with the two-dimensional discrete Fourier transforms of the columns of the matrix Z

```
>> fft2(Z)

ans =

4.0000 + 2.0000i 19.9904 -10.2321i -5.9904 - 6.7679i

1.8038 -16.0000i 22.8827 +28.9545i -13.5981 + 8.2321i

12.1962 -16.0000i -8.4019 + 4.7679i -23.8827 - 3.9545i

>> fft2(Z1)

ans =

17.0000 0.5000 - 9.5263i 0.5000 + 9.5263i

5.0000 + 5.1962i 8.0000 +13.8564i -17.5000 - 0.8660i

5.0000 - 5.1962i -17.5000 + 0.8660i 8.0000 -13.8564i
```

ifft(Z)

Matrix with the inverses of the discrete Fourier transforms of the columns of Z

```
>> ifft(Z)

ans =

2.0000 - 1.6667i 0 + 3.6667i -0.6667 - 1.3333i
```

```
-2.2321 - 1.6874i 2.3094 - 0.333333i 3.9880 - 3.3126i

1.2321 + 2.3541i -2.3094 - 0.333333i 1.6786 - 7.3541i

>> ifft(Z1)

ans =

2.0000 3.6667 0

-0.5000 - 2.0207i -0.3333 - 2.3094i 2.5000 + 2.5981i

-0.5000 + 2.0207i -0.3333 + 2.3094i 2.5000 - 2.5981i
```

ifft2(Z)

Matrix with the inverses of the 2-D discrete Fourier transforms of the columns of Z

```
>> ifft2(Z)

ans =

0.4444 + 0.2222i -0.6656 - 0.7520i 2.2212 - 1.1369i

1.3551 - 1.7778i -2.6536 - 0.4394i -0.9335 + 0.5298i

0.2004 - 1.7778i -1.5109 + 0.9147i 2.5425 + 3.2172i

>> ifft2(Z1)

ans =

1.8889 0.0556 + 1.0585i 0.0556 - 1.0585i

0.5556 - 0.5774i 0.8889 - 1.5396i -1.9444 + 0.0962i

0.5556 + 0.5774i -1.9444 - 0.0962i 0.8889 + 1.5396i
```

image image
image

2.5 GENERAL FUNCTIONS WITH ARGUMENT A COMPLEX MATRIX

image

M ATLAB incorporates a large group of trigonometric, hyperbolic, exponential and logarithmic functions that take a complex matrix as an argument. Obviously, all of these functions will also take a complex vector as an argument, since the vector is a particular case of a matrix. All functions are applied element by element in the matrix.

image image
image

2.5.1 Trigonometric functions of complex matrix variable

image

B elow is a table with the complex variable trigonometric functions and their inverses that MATLAB incorporates, illustrated with examples. For all the functions, the Z and Z1 matrices defined in the sine function are used as arguments.

Direct trigonometric function

sin(z) Sine function

```
>> Z=[1-i, 1+i, 2i;3-6i, 2+4i, -i;i,2i,3i]

Z =

1.0000 - 1.0000i 1.0000 + 1.0000i 0 + 2.0000i

3.0000 - 6.0000i 2.0000 + 4.0000i 0 - 1.0000i

0 + 1.0000i 0 + 2.0000i 0 + 3.0000i

>> Z1=[1,1,2;3,2,-1;1,2,3]

Z1 =

1 1 2

3 2 -1
```

```
1 2 3
>> sin(Z)
ans =
1.0e+002 *
0.0130 - 0.0063i 0.0130 + 0.0063i 0 + 0.0363i
0.2847 + 1.9969i 0.2483 - 0.1136i 0 - 0.0118i
0 + 0.0118i 0 + 0.0363i 0 + 0.1002i
>> sin(Z1)
ans =
0.8415 0.8415 0.9093
0.1411 0.9093 -0.8415
0.8415 0.9093 0.1411
```

cos(z) Cosine function

```
>> cos(Z)
ans =
1.0e+002 *
0.0083 + 0.0099i 0.0083 - 0.0099i 0.0376
-1.9970 + 0.2847i -0.1136 - 0.2481i 0.0154
0.0154 0.0376 0.1007
>> cos(Z1)
```

```
ans =

0.5403 0.5403 -0.4161

-0.9900 -0.4161 0.5403

0.5403 -0.4161 -0.9900
```

tan(z) Tangent function

```
>> tan(Z)

ans =

0.2718 - 1.0839i 0.2718 + 1.0839i 0 + 0.9640i

-0.0000 - 1.0000i -0.0005 + 1.0004i 0 - 0.7616i

0 + 0.7616i 0 + 0.9640i 0 + 0.9951i

>> tan(Z1)

ans =

1.5574 1.5574 -2.1850

-0.1425 -2.1850 -1.5574

1.5574 -2.1850 -0.1425
```

csc(z) Cosecant function

```
>> csc(Z)

ans =

0.6215 + 0.3039i 0.6215 - 0.3039i 0 - 0.2757i
```

```
0.0007 - 0.0049i 0.0333 + 0.0152i 0 + 0.8509i

0 - 0.8509i 0 - 0.2757i 0 - 0.0998i

>> csc(Z1)

ans =

1.1884 1.1884 1.0998

7.0862 1.0998 -1.1884

1.1884 1.0998 7.0862
```

sec(z) Secant function

```
>> sec(Z)

ans =

0.4983 - 0.5911i 0.4983 + 0.5911i 0.2658

-0.0049 - 0.0007i -0.0153 + 0.0333i 0.6481

0.6481 0.2658 0.0993

>> sec(Z1)

ans =

1.8508 1.8508 -2.4030

-1.0101 -2.4030 1.8508

1.8508 -2.4030 -1.0101
```

cot(z) Cotangent function

```
>> cot(Z)

ans =

0.2176 + 0.8680i 0.2176 - 0.8680i 0 - 1.0373i

-0.0000 + 1.0000i -0.0005 - 0.9996i 0 + 1.3130i

0 - 1.3130i 0 - 1.0373i 0 - 1.0050i

>> cot(Z1)

ans =

0.6421 0.6421 -0.4577

-7.0153 -0.4577 -0.6421

0.6421 -0.4577 -7.0153
```

image

Inverse trigonometric functions

asin(z) Sine arc function

```
>> asin(Z)

ans =

0.6662 - 1.0613i 0.6662 + 1.0613i 0 + 1.4436i

0.4592 - 2.5998i 0.4539 + 2.1986i 0 - 0.8814i

0 + 0.8814i 0 + 1.4436i 0 + 1.8184i
```

```
>> asin(Z1)

ans =

1.5708 1.5708 1.5708 - 1.3170i

1.5708 - 1.7627i 1.5708 - 1.3170i -1.5708

1.5708 1.5708 - 1.3170i 1.5708 - 1.7627i
```

acos(z) Arc cosine function

```
>> acos(Z)

ans =

0.9046 + 1.0613i 0.9046 - 1.0613i 1.5708 - 1.4436i

1.1115 + 2.5998i 1.1169 - 2.1986i 1.5708 + 0.8814i

1.5708 - 0.8814i 1.5708 - 1.4436i 1.5708 - 1.8184i

>> acos(Z1)

ans =

0 0 0 + 1.3170i

0 + 1.7627i 0 + 1.3170i 3.1416

0 0 + 1.3170i 0 + 1.7627i
```

atan(z) and atan2(z) Tangent arc function

```
>> atan(Z)

Warning: Singularity in ATAN. This warning will be removed in
```

```
a future release.
Consider using DBSTOP IF NANINF when debugging.
Warning: Singularity in ATAN. This warning will be removed in
a future release.
Consider using DBSTOP IF NANINF when debugging.
ans =
1.0172 - 0.4024i 1.0172 + 0.4024i -1.5708 + 0.5493i
1.5030 - 0.1335i 1.4670 + 0.2006i 0 - Infi
0 + Infi -1.5708 + 0.5493i -1.5708 + 0.3466i
>> atan(Z1)
ans =
0.7854 0.7854 1.1071
1.2490 1.1071 -0.7854
0.7854 1.1071 1.2490
```

acsc(z) Cosecant arc function

```
>> acsc(Z)
ans =
0.4523 + 0.5306i 0.4523 - 0.5306i 0 - 0.4812i
0.0661 + 0.1332i 0.0982 - 0.1996i 0 + 0.8814i
0 - 0.8814i 0 - 0.4812i 0 - 0.3275i
```

```
>> acsc(Z1)

ans =

1.5708 1.5708 0.5236

0.3398 0.5236 -1.5708

1.5708 0.5236 0.3398
```

asec(z) Secant arc function

```
>> asec(Z)

ans =

1.1185 - 0.5306i 1.1185 + 0.5306i 1.5708 + 0.4812i

1.5047 - 0.1332i 1.4726 + 0.1996i 1.5708 - 0.8814i

1.5708 + 0.8814i 1.5708 + 0.4812i 1.5708 + 0.3275i

>> asec(Z1)

ans =

0 0 1.0472

1.2310 1.0472 3.1416

0 1.0472 1.2310
```

acot(z) Cotangent arc function

```
>> acot(Z)

Warning: Singularity in ATAN. This warning will be removed in
```

```
a future release.
Consider using DBSTOP IF NANINF when debugging.
> In acot at 13
ans =
0.5536 + 0.4024i 0.5536 - 0.4024i 0 - 0.5493i
0.0678 + 0.1335i 0.1037 - 0.2006i 0 + Infi
0 - Infi 0 - 0.5493i 0 - 0.3466i
>> acot(Z1)
ans =
0.7854 0.7854 0.4636
0.3218 0.4636 -0.7854
0.7854 0.4636 0.3218
```

image image
image

2.5.2 Hyperbolic functions of complex matrix variable

image

B elow is a table with the complex variable hyperbolic functions and their inverses that MATLAB incorporates, illustrated with examples.

Direct hyperbolic functions

sinh(z) Hyperbolic sine function

```
>> sinh(Z)

ans =

0.6350 - 1.2985i 0.6350 + 1.2985i 0 + 0.9093i

9.6189 + 2.8131i -2.3707 - 2.8472i 0 - 0.8415i

0 + 0.8415i 0 + 0.9093i 0 + 0.1411i

>> sinh(Z1)

ans =

1.1752 1.1752 3.6269

10.0179 3.6269 -1.1752

1.1752 3.6269 10.0179
```

cosh(z) Hyperbolic cosine function

```
>> cosh(Z)

ans =

0.8337 - 0.9889i 0.8337 + 0.9889i -0.4161

9.6667 + 2.7991i -2.4591 - 2.7448i 0.5403

0.5403 -0.4161 -0.9900

>> cosh(Z1)

ans =

1.5431 1.5431 3.7622

10.0677 3.7622 1.5431

1.5431 3.7622 10.0677
```

tanh(z) Hyperbolic tangent function

```
>> tanh(Z)

ans =

1.0839 - 0.2718i 1.0839 + 0.2718i 0 - 2.1850i

0.9958 + 0.0026i 1.0047 + 0.0364i 0 - 1.5574i

0 + 1.5574i 0 - 2.1850i 0 - 0.1425i

>> tanh(Z1)

ans =
0.7616 0.7616 0.9640

0.9951 0.9640 -0.7616

0.7616 0.9640 0.9951
```

csch(z) Hyperbolic cosecant function

```
>> csch(Z)

ans =

0.3039 + 0.6215i 0.3039 - 0.6215i 0 - 1.0998i

0.0958 - 0.0280i -0.1727 + 0.2074i 0 + 1.1884i

0 - 1.1884i 0 - 1.0998i 0 - 7.0862i

>> csch(Z1)

ans =

0.8509 0.8509 0.2757

0.0998 0.2757 -0.8509

0.8509 0.2757 0.0998
```

sech(z) Hyperbolic secant function

```
>> sech(Z)

ans =

0.4983 + 0.5911i 0.4983 - 0.5911i -2.4030

0.0954 - 0.0276i -0.1811 + 0.2021i 1.8508
```

```
1.8508 -2.4030 -1.0101

>> sech(Z1)

ans =

0.6481 0.6481 0.2658

0.0993 0.2658 0.6481

0.6481 0.2658 0.0993
```

coth(z) Hyperbolic cotangent function

```
>> coth(Z)

ans =

0.8680 + 0.2176i 0.8680 - 0.2176i 0 + 0.4577i

1.0042 - 0.0027i 0.9940 - 0.0360i 0 + 0.6421i

0 - 0.6421i 0 + 0.4577i 0 + 7.0153i

>> coth(Z1)

ans =

1.3130 1.3130 1.0373

1.0050 1.0373 -1.3130

1.3130 1.0373 1.0050
```

image

Inverse hyperbolic functions

asinh(z) Hyperbolic sine arc function

```
>> asinh(Z)
ans =
1.0613 - 0.6662i 1.0613 + 0.6662i 1.3170 + 1.5708i
2.5932 - 1.1027i 2.1836 + 1.0969i 0 - 1.5708i
0 + 1.5708i 1.3170 + 1.5708i 1.7627 + 1.5708i
>> asinh(Z1)
ans =
0.8814 0.8814 1.4436
1.8184 1.4436 -0.8814
0.8814 1.4436 1.8184
```

acosh(z) Hyperbolic arc cosine function

```
>> acosh(Z)
ans =
1.0613 - 0.9046i 1.0613 + 0.9046i 1.4436 + 1.5708i
2.5998 - 1.1115i 2.1986 + 1.1169i 0.8814 - 1.5708i
0.8814 + 1.5708i 1.4436 + 1.5708i 1.8184 + 1.5708i
>> acosh(Z1)
```

```
ans =
0 0 1.3170
1.7627 1.3170 0 + 3.1416i
0 1.3170 1.7627
```

atanh(z) Hyperbolic arc tangent function

```
>> atanh(Z)
ans =
0.4024 - 1.0172i 0.4024 + 1.0172i 0 + 1.1071i
0.0656 - 1.4377i 0.0964 + 1.3715i 0 - 0.7854i
0 + 0.7854i 0 + 1.1071i 0 + 1.2490i
>> atanh(Z1)
ans =
Inf Inf 0.5493 + 1.5708i
0.3466 + 1.5708i 0.5493 + 1.5708i -Inf
Inf 0.5493 + 1.5708i 0.3466 + 1.5708i
```

acsch(z) Hyperbolic arc cosecant function

```
>> acsch(Z)
ans =
0.5306 + 0.4523i 0.5306 - 0.4523i 0 - 0.5236i
```

```
0.0672 + 0.1334i 0.1019 - 0.2003i 0 + 1.5708i

0 - 1.5708i 0 - 0.5236i 0 - 0.3398i

>> acsch(Z1)

ans =

0.8814 0.8814 0.4812

0.3275 0.4812 -0.8814

0.8814 0.4812 0.3275
```

asech(z) Hyperbolic arc secant function

```
>> asech(Z)

ans =

0.5306 + 1.1185i 0.5306 - 1.1185i 0.4812 - 1.5708i

0.1332 + 1.5047i 0.1996 - 1.4726i 0.8814 + 1.5708i

0.8814 - 1.5708i 0.4812 - 1.5708i 0.3275 - 1.5708i

>> asech(Z1)

ans =

0 0 0 + 1.0472i

0 + 1.2310i 0 + 1.0472i 0 + 3.1416i

0 0 0 + 1.0472i 0 + 1.2310i
```

acoth(z) Hyperbolic cotangent arc function

```
>> acoth(Z)

ans =

0.4024 + 0.5536i 0.4024 - 0.5536i 0 - 0.4636i

0.0656 + 0.1331i 0.0964 - 0.1993i 0 + 0.7854i

0 - 0.7854i 0 - 0.4636i 0 - 0.3218i

>> acoth(Z1)

ans =

Inf Inf 0.5493

0.3466 0.5493 -Inf

Inf 0.5493 0.3466
```

image image
image

2.5.3 Exponential and logarithmic functions of complex matrix variable

image

B elow is a table with the exponential and logarithmic functions that MATLAB incorporates, illustrated with examples. The Z and Z1 matrices are the same as in the previous examples.

————

image

Function	Meaning
exp(z)	Exponential function in base e (e^x)
————	>> exp(Z)
image	ans = 1.4687 - 2.2874i 1.4687 + 2.2874i -0.4161 + 0.9093i 19.2855 + 5.6122i -4.8298 - 5.5921i 0.5403 - 0.8415i 0.5403 + 0.8415i -0.4161 + 0.9093i -0.9900 + 0.1411i >> exp(Z1)

```
         ans =

         2.7183 2.7183 7.3891

         20.0855 7.3891 0.3679

         2.7183 7.3891 20.0855

         Logarithm function in base e of x

         >> log(Z)

         ans =

         0.3466 - 0.7854i 0.3466 + 0.7854i 0.6931 + 1.5708i

         1.9033 - 1.1071i 1.4979 + 1.1071i 0 - 1.5708i

log(x)   0 + 1.5708i 0.6931 + 1.5708i 1.0986 + 1.5708i

         >> log(Z1)

         ans =

         0 0 0.6931

         1.0986 0.6931 0 + 3.1416i

         0 0.6931 1.0986

log10(x) Logarithm function in base 10 of x

         >> log10(Z)

         ans =

         0.1505 - 0.3411i 0.1505 + 0.3411i 0.3010 + 0.6822i
```

```
        0.8266 - 0.4808i 0.6505 + 0.4808i 0 - 0.6822i

        0 + 0.6822i 0.3010 + 0.6822i 0.4771 + 0.6822i

        >> log10(Z1)

        ans =

        0 0 0.3010

        0.4771 0.3010 0 + 1.3644i

        0 0.3010 0.4771
```

Logarithm function in base 2 of x

```
        >> log2(Z)

        ans =

        0.5000 - 1.1331i 0.5000 + 1.1331i 1.0000 + 2.2662i

        2.7459 - 1.5973i 2.1610 + 1.5973i 0 - 2.2662i

log2(x) 0 + 2.2662i 1.0000 + 2.2662i 1.5850 + 2.2662i

        >> log2(Z1)

        ans =

        0 0 1.0000

        1.5850 1.0000 0 + 4.5324i

        0 1.0000 1.5850
```

pow2(x) Function power of base 2 of x

```
>> pow2(Z)

ans =

1.5385 - 1.2779i 1.5385 + 1.2779i 0.1835 + 0.9830i

-4.2054 + 6.8055i -3.7307 + 1.4427i 0.7692 - 0.6390i

0.7692 + 0.6390i 0.1835 + 0.9830i -0.4870 + 0.8734i

>> pow2(Z1)

ans =

2.0000 2.0000 4.0000

8.0000 4.0000 0.5000

2.0000 4.0000 8.0000
```

sqrt(x)   Square root function of x

```
>> sqrt(Z)

ans =

1.0987 - 0.4551i 1.0987 + 0.4551i 1.0000 + 1.0000i

2.2032 - 1.3617i 1.7989 + 1.1118i 0.7071 - 0.7071i

0.7071 + 0.7071i 1.0000 + 1.0000i 1.2247 + 1.2247i

>> sqrt(Z1)

ans =

1.0000 1.0000 1.4142

1.7321 1.4142 0 + 1.0000i
```

```
1.0000 1.4142 1.7321
```

image image
image

2.5.4 Specific complex matrix variable functions

image

M ATLAB incorporates a group of complex variable functions specifically for working with modules, arguments, and real and imaginary parts. These functions include the following:

image

Function	Meaning
abs(Z)	Z-complex module >> abs(Z) ans = 1.4142 1.4142 2.0000 6.7082 4.4721 1.0000 1.0000 2.0000 3.0000 >> abs(Z1) ans = 1 1 2

3 2 1

1 2 3

Z-complex argument

>> angle(Z)

ans =

-0.7854 0.7854 1.5708

-1.1071 1.1071 -1.5708

angle(Z) 1.5708 1.5708 1.5708

>> angle(Z1)

ans =

0 0 0

0 0 3.1416

0 0 0

conj(Z) Z-complex conjugate

>> conj(Z)

ans =

1.0000 + 1.0000i 1.0000 - 1.0000i 0 - 2.0000i

3.0000 + 6.0000i 2.0000 - 4.0000i 0 + 1.0000i

0 - 1.0000i 0 - 2.0000i 0 - 3.0000i

```
>> conj(Z1)

ans =

1 1 2

3 2 -1

1 2 3
```

Real part of complex Z

```
        >> real(Z)

        ans =

        1 1 0

        3 2 0

real(Z) 0 0 0

        >> real(Z1)

        ans =

        1 1 2

        3 2 -1

        1 2 3
```

imag(Z) Imaginary part of complex Z

```
>> imag(Z)

ans =
-1 1 2

-6 4 -1

1 2 3

>> imag(Z1)

ans =

0 0 0

0 0 0

0 0 0
```

floor(Z)

Applies the floor function to real(Z) and imag(Z)

```
>> floor(12.357*Z)

ans =

12.0000 -13.0000i 12.0000 +12.0000i 0 +24.0000i

37.0000 -75.0000i 24.0000 +49.0000i 0 -13.0000i

0 +12.0000i 0 +24.0000i 0 +37.0000i

>> floor(12.357*Z1)

ans =

12 12 24

37 24 -13

12 24 37
```

ceil(Z) Applies the ceil function to real(Z) and imag(Z)

```
>> ceil(12.357*Z)
ans =
13.0000 -12.0000i 13.0000 +13.0000i 0 +25.0000i
38.0000 -74.0000i 25.0000 +50.0000i 0 -12.0000i
0 +13.0000i 0 +25.0000i 0 +38.0000i
>> ceil(12.357*Z1)
ans =
13 13 25
38 25 -12
13 25 38
```

round(Z) Applies round function to real(Z) and imag(Z)

```
>> round(12.357*Z)
ans =
12.0000 -12.0000i 12.0000 +12.0000i 0 +25.0000i
37.0000 -74.0000i 25.0000 +49.0000i 0 -12.0000i
0 +12.0000i 0 +25.0000i 0 +37.0000i
>> round(12.357*Z1)
ans =
12 12 25
```

```
37 25 -12

12 25 37
```

Applies fix function to real(Z) and imag(Z)

```
>> fix(12.357*Z)

ans =

12.0000 -12.0000i 12.0000 +12.0000i 0 +24.0000i

37.0000 -74.0000i 24.0000 +49.0000i 0 -12.0000i
```

fix(Z)

```
0 +12.0000i 0 +24.0000i 0 +37.0000i

>> fix(12.357*Z1)

ans =

12 12 24

37 24 -12

12 24 37
```

2.6 OPERATIONS WITH REAL AND COMPLEX MATRIX VARIABLES

M ATLAB covers the usual operations of addition, difference, product, potentiation, exponentiation and inversion with complex matrix variables. Obviously, all these operations are also valid for real matrix variables as they are a particular case of complex variables. The following table summarises these operations, which are valid both for real and complex numerical matrix variables and for algebraic matrix variables.

A+B *Sum of matrix variables*

```
>> A=[1+i, 1-i, 2i; -i,-3i,6-5i;2+3i, 2-3i, i]
A =
1.0000 + 1.0000i 1.0000 - 1.0000i 0 + 2.0000i
0 - 1.0000i 0 - 3.0000i 6.0000 - 5.0000i
2.0000 + 3.0000i 2.0000 - 3.0000i 0 + 1.0000i
>> B=[i, -i, 2i; 1-i,7-3i,2-5i;8-6i, 5-i, 1+i]
B =
0 + 1.0000i 0 - 1.0000i 0 + 2.0000i
1.0000 - 1.0000i 7.0000 - 3.0000i 2.0000 - 5.0000i
8.0000 - 6.0000i 5.0000 - 1.0000i 1.0000 + 1.0000i
>> A1=[1 6 2;3 5 0; 2 4 -1]
A1 =
1 6 2
3 5 0
2 4 -1
>> B1=[-3 -6 1;-3 -5 2; 12 14 -10]
B1 =
-3 -6 1
-3 -5 2
12 14 -10
>> A+B
```

```
ans =
1.0000 + 2.0000i 1.0000 - 2.0000i 0 + 4.0000i
1.0000 - 2.0000i 7.0000 - 6.0000i 8.0000 -10.0000i
10.0000 - 3.0000i 7.0000 - 4.0000i 1.0000 + 2.0000i
>> A1+B1
ans =
-2 0 3
0 0 2
14 18 -11
```

Difference of matrix variables

A-B

```
>> A-B
ans =
1.0000 1.0000 0
-1.0000 -7.0000 4.0000
-6.0000 + 9.0000i -3.0000 - 2.0000i -1.0000i
>> A1-B1
ans =
4 12 1
6 10 -2
-10 -10 9
```

Product of matrix variables

A*B

```
>> A*B
ans =
11.0000 +15.0000i 7.0000 - 1.0000i -7.0000 - 3.0000i
16.0000 -79.0000i 15.0000 -52.0000i -2.0000 - 5.0000i
2.0000 + 5.0000i 9.0000 -24.0000i -18.0000 -11.0000i
>> A1*B1
ans =
3 -8 -7
-24 -43 13
-30 -46 20
```

A^ $^{\mathbf{n}}$

nth power of the matrix variable A

```
>> A^3
ans =
1.0e+002 *
0.1000 - 0.3400i -0.3200 - 0.1200i 0.3400 - 0.3600i
0.0900 - 0.0300i -1.0700 + 0.2100i -2.2500 - 0.6700i
```

```
0.3700 - 0.7900i -1.0300 - 0.0300i -0.0700 - 0.3700i
>> A1^3
ans =
155 358 46
159 347 30
106 232 19
ans =
Columns 1 through 2
1.0000 - 1.0000i 2.0000
Columns 3 through 4
6.0000 -10.0000i 60.0000 +36.0000i
>> cumprod([1, 0, -23, 12, 16])
ans =
1 0 0 0 0
```

Scalar p raised to the matrix variable A

P^ A

```
>> 3^A
ans =
0.0159 - 1.2801i -0.5297 + 2.8779i -1.9855 + 3.0796i
-10.3372 + 0.4829i 17.0229 +12.9445i 14.7327 +20.1633i
-5.0438 + 0.2388i 7.0696 + 6.9611i 5.7189 + 9.5696i
>> 3^A1
ans =
1.0e+003 *
2.2230 4.9342 0.4889
2.1519 4.7769 0.4728
1.4346 3.1844 0.3156
```

Matrix variable transposed from variable A

A'

```
>> A'
ans =
1.0000 - 1.0000i 0 + 1.0000i 2.0000 - 3.0000i
1.0000 + 1.0000i 0 + 3.0000i 2.0000 + 3.0000i
0 - 2.0000i 6.0000 + 5.0000i 0 - 1.0000i
>> A1'
ans =
1 3 2
6 5 4
2 0 -1
```

A^ -1 *Inverse matrix variable of matrix variable A*

```
>> A^-1
ans =
-2.5000 + 2.0000i -0.0500 + 0.6500i 0.8500 - 1.0500i
0.5000 + 3.0000i 0.5500 + 0.3500i -0.3500 - 0.9500i
-1.0000 - 1.0000i -0.2000 + 0.1000i 0.4000 + 0.3000i
>> A1^-1
ans =
-0.2941 0.8235 -0.5882
0.1765 -0.2941 0.3529
0.1176 0.4706 -0.7647
>> A*A^-1
ans =
1.0000 0.0000 - 0.0000i -0.0000 + 0.0000i
-0.0000 - 0.0000i 1.0000 + 0.0000i 0.0000
0.0000 + 0.0000i 0.0000 1.0000 + 0.0000i
>> A1*A1^-1
ans =
1.0000 -0.0000 0
-0.0000 1.0000 0
-0.0000 -0.0000 1.0000
```

A

*If A is square, A = B=(A^{-1})*B and if A is not square, A = B is the least squares solution of the system AX=B.*

```
>> A
ans =
-0.9000 -15.3000i 6.8000 + 1.1000i 1.0500 - 3.6500i
-10.6000 - 5.2000i 5.2000 - 4.1000i -2.5500 - 2.3500i
5.9000 - 0.7000i 0.2000 + 3.4000i 2.2000 - 0.1000i
>> A1B1
ans =
-8.6471 -10.5882 7.2353
4.5882 5.3529 -3.9412
-10.9412 -13.7647 8.7059
```

B/A *Coincides with (A'B')*

```
>> B/A
ans =
3.0000 - 5.0000i -0.5000 - 1.0000i -0.5000 + 2.0000i
```

```
5.0000 +27.0000i 5.6000 + 2.7000i -3.2000 - 8.9000i
-2.5000 +43.5000i 6.3000 + 6.6000i -2.1000 -17.2000i
>> A'B'
ans =
3.0000 + 5.0000i 5.0000 -27.0000i -2.5000 -43.5000i
-0.5000 + 1.0000i 5.6000 - 2.7000i 6.3000 - 6.6000i
-0.5000 - 2.0000i -3.2000 + 8.9000i -2.1000 +17.2000i
>> B1/A1
ans =
-0.0588 -0.2353 -1.1176
0.2353 -0.0588 -1.5294
-2.2353 1.0588 5.5294
>> A1'\B1'
ans =
-0.0588 0.2353 -2.2353
-0.2353 -0.0588 1.0588
-1.1176 -1.5294 5.5294
```

G iven the complex numbers $z_1 = 1 - i$ *and* $z_2 = 5i$*, calculate:* z_1^3 , z_1^2 / z_2^4 , z , $z_1^{1/2}$, $z_2^{3/2}$, *ln(*$z_1 + z_2$*) ,sen(*$z_1 - z_2$*) and tanh(*z_1 / z_2*).*

```
>> Z1=1-i
Z1 =
1.0000 - 1.0000i
>> Z2=5i
Z2 =
0 + 5.0000i
>> Z1^3
ans =
-2.0000 - 2.0000i
>> Z1^2/Z2^4
ans =
0 - 0.0032i
>> sqrt(Z1)
ans =
1.0987 - 0.4551i
```

```
>> sqrt(Z2^3)

ans =

7.9057 - 7.9057i

>> log(Z1+Z2)

ans =

1.4166 + 1.3258i

>> sin(Z1-Z2)

ans =

1.6974e+002 -1.0899e+002i

>> tanh(Z1/Z2)

ans =

-0.2052 - 0.1945i
```

Exercise 2. Perform the following operations with complex numbers:

```
>> (i^8-i^(-8))/(3-4 *i) + 1

   ans =

   1

   >> i^(sin(1+i))

   ans =

   -0.16665202215166 + 0.32904139450307i

   >> (2+log(i))^(1/i)

   ans =

   1.15809185259777 - 1.56388053989023i

   >> (1+i)^i

   ans =

   0.42882900629437 + 0.15487175246425i
```

```
>> i^(log(1+i))

ans =

0.24911518828716 + 0.15081974484717i

>> (1+sqrt(3)*i)^(1-i)

ans =

5.34581479196611 + 1.97594883452873i
```

F ind the real part , imaginary part, modulus and argument of the following complexes:

image

```
>> Z1=i^3*i; Z2=(1+sqrt (3)*i)^(1-i); Z3=(i^i)^i;Z4=i^i;

>> format short

>> real([Z1 Z2 Z3 Z4])

ans =

1.0000 5.3458 0.0000 0.2079

>> imag([Z1 Z2 Z3 Z4])

ans =

0 1.9759 -1.0000 0

>> abs([Z1 Z2 Z3 Z4])

ans =

1.0000 5.6993 1.0000 0.2079

>> angle([Z1 Z2 Z3 Z4])

ans =

0 0.3541 -1.5708 0
```

Consider the imaginary unit product matrix M by the square matrix of order 3 whose row elements are the first nine positive integers.

Obtain its square, its square root and its exponentials of bases 2 and -2.

Obtain its element-by-element neperian logarithm and its element-by-element e-base exponential.

Obtain e^{M} *and Ln(M).*

```
>> M=i*[1 2 3;4 5 6;7 8 9]

M =

0 + 1.0000i 0 + 2.0000i 0 + 3.0000i

0 + 4.0000i 0 + 5.0000i 0 + 6.0000i

0 + 7.0000i 0 + 8.0000i 0 + 9.0000i

>> C=M^2

C =

-30 -36 -42

-66 -81 -96

-102 -126 -150

>> D=M^(1/2)

D =

0.8570 - 0.2210i 0.5370 + 0.2445i 0.2169 + 0.7101i

0.7797 + 0.6607i 0.9011 + 0.8688i 1.0224 + 1.0769i

0.7024 + 1.5424i 1.2651 + 1.4930i 1.8279 + 1.4437i

>> 2^M

ans =

0.7020 - 0.6146i -0.1693 - 0.2723i -0.0407 + 0.0699i

-0.2320 - 0.3055i 0.7366 - 0.3220i -0.2947 - 0.3386i

-0.1661 + 0.0036i -0.3574 - 0.3717i 0.4513 - 0.7471i

>> (-2)^M

ans =

17.3946 -16.8443i 4.3404 - 4.5696i -7.7139 + 7.7050i

1.5685 - 1.8595i 1.1826 - 0.5045i -1.2033 + 0.8506i

-13.2575 +13.1252i -3.9751 + 3.5607i 6.3073 - 6.0038i

>> log(M)
```

```
    ans =
    0 + 1.5708i 0.6931 + 1.5708i 1.0986 + 1.5708i
    1.3863 + 1.5708i 1.6094 + 1.5708i 1.7918 + 1.5708i
    1.9459 + 1.5708i 2.0794 + 1.5708i 2.1972 + 1.5708i
    >> exp(M)
    ans =
    0.5403 + 0.8415i -0.4161 + 0.9093i -0.9900 + 0.1411i
    -0.6536 - 0.7568i 0.2837 - 0.9589i 0.9602 - 0.2794i
    0.7539 + 0.6570i -0.1455 + 0.9894i -0.9111 + 0.4121i
```

```
>> logm(M)
    ans =
    -5.4033 - 0.8472i 11.9931 - 0.3109i -5.3770 + 0.8846i
    12.3029 + 0.0537i -22.3087 + 0.8953i 12.6127 + 0.4183i
    -4.7574 + 1.6138i 12.9225 + 0.7828i -4.1641 + 0.6112i
    >> expm(M)
    ans =
    0.3802 - 0.6928i -0.3738 - 0.2306i -0.1278 + 0.2316i
    -0.5312 - 0.1724i 0.3901 - 0.1434i -0.6886 - 0.1143i
    -0.4426 + 0.3479i -0.8460 - 0.0561i -0.2493 - 0.4602i
```

Consider the vector V as the sum of the complex vector Z= (i,-i,i) and the real vector R=(0,1,1). Find the mean, median, standard deviation, variance, sum, product, maximum and minimum of the elements of V, as well as its gradient, the discrete Fourier transform and its inverse.

```
    >> Z=[i,-i,i]
    Z =
    0 + 1.0000i 0 - 1.0000i 0 + 1.0000i
    >> R=[0,1,1]
    R =
0 1 1
>> V=Z+R
V =
0 + 1.0000i 1.0000 - 1.0000i 1.0000 + 1.0000i
>> [mean(V),median(V),std(V),var(V),sum(V),prod(V),max(V),min(V)]'].
ans =
0.6667 - 0.3333i
1.0000 + 1.0000i
1.2910
1.6667
2.0000 - 1.0000i
0 - 2.0000i
1.0000 + 1.0000i
0 - 1.0000i
>> gradient(V)
ans =
1.0000 - 2.0000i 0.5000 0 + 2.0000i
>> fft(V)
ans =
2.0000 + 1.0000i -2.7321 + 1.0000i 0.7321 + 1.0000i
>> ifft(V)
ans =
0.6667 + 0.3333i 0.2440 + 0.3333i -0.9107 + 0.3333i
```

Exercise 6. Given the matrices:

Imagen en blanco y negro de un reloj Descripción generada automáticamente con confianza media

Imagen de la pantalla de un celular con texto e imagen Descripción generada automáticamente con confianza media

I nitially calculate A1+A2, B1-B2 and C1+C2.

Then calculate AB - BA , $A^{2} + B^{2} + C^{2}$ *, ABC, sqrt(A)+ sqrt(B) - sqrt(C),* e^{A} *(*e^{B} *+* e^{C} *), their inverses and their transposes.*

Finally, check that multiplying any matrix by its inverse gives the matrix identity.

```
>> A1=eye(3)
A1 =
1 0 0
0 1 0
0 0 1
>> A2=[0 1 0; 0 0 1;0 0 0]
A2 =
0 1 0
0 0 1
0 0 0
>> A= A1+A2
A =
1 1 0
0 1 1
0 0 1
>> B1=[0 1 2;0 -1 3;0 0 0]
B1 =
0 1 2
0 -1 3
0 0 0
>> B2=[-i i -i;0 0 0 i;0 0 0 i]
```

```
B2 =
0 - 1.0000i 0 + 1.0000i 0 - 1.0000i
0 0 0 + 1.0000i
0 0 0 + 1.0000i
>> B=B1-B2
B =
0 + 1.0000i 1.0000 - 1.0000i 2.0000 + 1.0000i
0 -1.0000 3.0000 - 1.0000i
0 0 0 - 1.0000i
>> C1=[1, -1, 0;-1,sqrt(2)*i,-sqrt(2)*i;0,0,-1]
C1 =
1.0000 -1.0000 0
-1.0000 0 + 1.4142i 0 - 1.4142i
0 0 -1.0000
>> C2=[0 2 1;1 0 0;1 -1 0]
C2 =
0 2 1
1 0 0
1 -1 0
>> C=C1+C2
C =
1.0000 1.0000 1.0000
0 0 0 + 1.4142i 0 - 1.4142i
1.0000 -1.0000 -1.0000
>> M1=A*B-B*A
M1 =
0 -1.0000 - 1.0000i 2.0000
0 0 1.0000 - 1.0000i
0 0 0
>> M2=A^2+B^2+C^2
M2 =
   2.0000 2.0000 2.0000 + 3.4142i 3.0000 - 5.4142i

   0 - 1.4142i -0.0000 + 1.4142i 0.0000 - 0.5858i

   0 2.0000 - 1.4142i 2.0000 + 1.4142i

   >> M3=A*B*C

   M3 =

   5.0000 + 1.0000i -3.5858 + 1.0000i -6.4142 + 1.0000i

   3.0000 - 2.0000i -3.0000 + 0.5858i -3.0000 + 3.4142i

   0 - 1.0000i 0 + 1.0000i 0 + 1.0000i

   >> M4=sqrtm(A)+sqrtm(B)-sqrtm(C)

   M4 =

   0.6356 + 0.8361i -0.3250 - 0.8204i 3.0734 + 1.2896i

   0.1582 - 0.1521i 0.0896 + 0.5702i 3.3029 - 1.8025i

   -0.3740 - 0.2654i 0.7472 + 0.3370i 1.2255 + 0.1048i



>> M5=expm(A)*(expm(B )+expm(C))

   M5 =

   14.1906 - 0.0822i 5.4400 + 4.2724i 17.9169 - 9.5842i

   4.5854 - 1.4972i 0.6830 + 2.1575i 8.5597 - 7.6573i

   3.5528 + 0.3560i 0.1008 - 0.7488i 3.2433 - 1.8406i

   >> inv(A)

   ans =

   1 -1 1

   0 1 -1

   0 0 1

   >> inv(B)

   ans =

   0 - 1.0000i -1.0000 - 1.0000i -4.0000 + 3.0000i

   0 -1.0000 1.0000 + 3.0000i

   0 0 0 + 1.0000i

   >> inv(C)
```

```
ans =
0.5000 0 0.5000
0.2500 0 - 0.3536i -0.2500
0.2500 0 + 0.3536i -0.2500
>> [A*inv(A) B*inv(B) C*inv(C)].
ans =
1 0 0 1 0 0 1 0 0
0 1 0 0 1 0 0 1 0
0 0 1 0 0 1 0 0 1
>> A'
ans =
1 0 0
1 1 0
0 1 1
>> B'
ans =
0 - 1.0000i 0 0
1.0000 + 1.0000i -1.0000 0
2.0000 - 1.0000i 3.0000 + 1.0000i 0 + 1.0000i
>> C'
ans =
1.0000 0 1.0000
1.0000 0 - 1.4142i -1.0000
1.0000 0 + 1.4142i -1.0000
```

Exercise 7. Given the matrices:

Imagen en blanco y negro Descripción generada automáticamente con confianza baja

A pply the sine, e-based exponential, logarithm, square root, modulus, argument and rounding functions to their elements.

Calculate e^{B} and Ln(A).

```
" A=[1 2 3; 4 5 6; 7 8 9]
A =
1 2 3
4 5 6
7 8 9
"sin(A)
ans =
0.8415 0.9093 0.1411
-0.7568 -0.9589 -0.2794
0.6570 0.9894 0.4121
"B=[1+i 2+i;3+i,4+i] B=[1+i 2+i;3+i,4+i] B=[1+i,4+i]
B =
1.0000 + 1.0000i 2.0000 + 1.0000i
3.0000 + 1.0000i 4.0000 + 1.0000i
"sin(B)
ans =
1.2985 + 0.6350i 1.4031 - 0.4891i
0.2178 - 1.1634i -1.1678 - 0.7682i
"exp(A)
ans =
1.0e+003 *
0.0027 0.0074 0.0201
0.0546 0.1484 0.4034
1.0966 2.9810 8.1031
"exp(B)
ans =
1.4687 + 2.2874i 3.9923 + 6.2177i
```

```
10.8523 +16.9014i 29.4995 +45.9428i

"log(B)

ans =

0.3466 + 0.7854i 0.8047 + 0.4636i

1.1513 + 0.3218i 1.4166 + 0.2450i

"sqrt(B)

ans =

1.0987 + 0.4551i 1.4553 + 0.3436i

1.7553 + 0.2848i 2.0153 + 0.2481i

"abs(B)

ans =

1.4142 2.2361

3.1623 4.1231

"imag(B)

ans =

1 1

1 1

"fix(sin(B))

ans =

1.0000 1.0000

0 - 1.0000i - 1.0000

" ceil(log(A))

ans =

0 1 2

2 2 2

2 3 3

"sign(B)

ans =

0.7071 + 0.7071i 0.8944 + 0.4472i

0.9487 + 0.3162i 0.9701 + 0.2425i
```

The exponential, square root and logarithm functions used above are applied element by element to the matrix, and have nothing to do with the exponential and logarithmic matrix functions used below.

```
"expm(B)
ans =
1.0e+002 *
-0.3071 + 0.4625i -0.3583 + 0.6939i
-0.3629 + 1.0431i -0.3207 + 1.5102i
"logm(A)
ans =
-5.6588 + 2.7896i 12.5041 - 0.4325i -5.6325 - 0.5129i
12.8139 - 0.7970i -23.3307 + 2.1623i 13.1237 - 1.1616i
-5.0129 - 1.2421i 13.4334 - 1.5262i -4.4196 + 1.3313i
```

Exercise 8. Solve the equation in the complex field:

$$Sen(z) = 2$$

```
>> vpa(solve('sin(z)=2'))
ans =
1.3169578969248167086250463473 08*i + 1.5707963267948966192313216916398
1.5707963267948966192313216916398 - 1.316957896924816708625046347308*i
```

Exercise 9. Solve the following equations:

a) $1+x+x^2+x+x^3+x+x^4+x^5 = 0$

b) $x^2+(6-i)x+8-4i = 0$

(c) $\tan(Z) = 3i/5$

```
>> solve('1+x+x^2+x^3+x^4+x^5 = 0')
```

```
ans =
-1
- 1/2 - (3^(1/2)*i)/2
1/2 - (3^(1/2)*i)/2
- 1/2 + (3^(1/2)*i)/2
1/2 + (3^(1/2)*i)/2
>> solve('x^2+(6-i)*x+8-4*i = 0')
ans =
-4
i - 2
>> vpa(solve('tan(Z) = 3*i/5'))
ans =
0.69314718055994530941723212145818*i
```

Exercise 10. Carry out the following operations:

(a) fourth roots of -1 and of 1

(b) fifth roots of 2+2i and -1+i √3

(c) real part of Tan(iLn((a+ib)/(a-ib))))

(d) imaginary part of Z= $(2+i)^{Cos(4-i)}$

```
>> solve('x^4+1=0')
ans =
2^(1/2)*(- i/2 - 1/2)
2^(1/2)*(i/2 - 1/2)
2^(1/2)*(1/2 - i/2)
2^(1/2)*(i/2 + 1/2)
>> pretty(solve('x^4+1=0'))
+- -+
| 1/2 / i 1 \ |
| 2 | - - - - | |
| \ 2 2 / |
| |
| 1/2 / i 1 \ |
| 2 | - - - | |
| \ 2 2 / |
| |
| 1/2 / 1 i \ |
| 2 | - - - | |
| \ 2 2 / |
| |
| 1/2 / i 1 \ |
| 2 | - + - | |
| \ 2 2 / |
+- -+
>> solve('x^4-1=0')
ans =
-1
1
-i
i
>> vpa(solve('x^5-2-2*i=0'))
ans =
0.19259341768888084906125263406469*i + 1.2159869826496146992458377919696
- 0.87055056329612413913627001747975*i - 0.87055056329612413913627001747975
0.55892786746600970394985946846702*i - 1.0969577045083811131206798770216
0.55892786746600970394985946846702 - 1.0969577045083811131206798770216*i
1.2159869826496146992458377919696*i + 0.19259341768888084906125263406469
>> vpa(solve('x^5+1-sqrt(3)*i=0')))
ans =
0.46721771281818786757419290603946*i + 1.0493881644090691705137652947201
1.1424056652180689506550734259384*i - 0.12007167380592154112409047542785
0.76862922680258900220179378744147 - 0.85364923855044142809268986292246*i
- 0.99480195671282768870147766609475*i - 0.57434917749851750339931347338896
0.23882781722701229856490119703938*i - 1.1235965399072191281921551333441
>> simplify(vpa(real(tan(i*log((a+i*b)/(a-i*b))))))
```

```
ans =

- 0.5*tanh(conj(log((a^2 + 2.0*a*b*i - 1.0*b^2)/(a^2 + b^2))))*i + (0.5*((a^2 + 2.0*a*b*i - 1.0*b^2)^2/(a^2 + b^2)^2 - 1)*i)/((a^2 + 2.0*a*b*i - 1.0*b^2)^2/(a^2 + b^2)^2 + 1)

>> simplify(vpa(imag((2+i)^cos(4-i))))

ans =

-0.62107490808037524310236676683417
```

Chapter 3.

GRAPHICS

M ATLAB is a scientific software that implements high performance graphics. It allows simple exploratory data analysis plots in two and three dimensions, curve plots in explicit, implicit and polar coordinates, surface plots in explicit, implicit and parametric coordinates, mesh and contour plots, volume plots and specialised plots.

It is also possible to choose windows and positions for graphs, to choose line and marker characteristics, to place axis limits, marks and meshes, to place annotations, labels and legends, to export graphs to different formats and other possibilities that will be shown throughout the chapter.

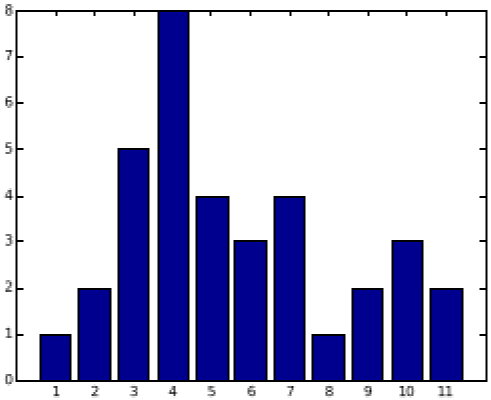
3.1 EXPLORATORY DATA CHARTS

M ATLAB incorporates commands that allow you to perform basic exploratory graphics, such as histograms, bar charts, pie charts, arrow plots, and so on. The following table summarises these commands. For all of them it is necessary to previously define the field of variation of the variable.

bar(Y)

Bar chart relative to the vector of frequencies Y. If Y is a matrix, the multiple bar chart is obtained for each row of Y

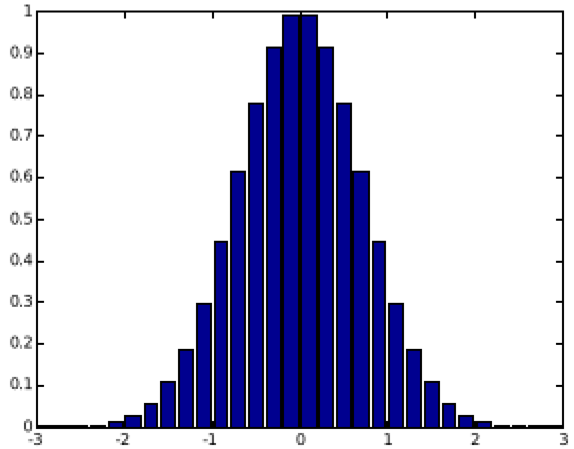
```
>> x=[1 2 5 8 4 3 4 1 2 3 2];
>> bar(x)
```



bar(x,Y)

Bar chart relative to the vector of frequencies Y where x is a vector defining the spaces on the X-axis to place the bars.

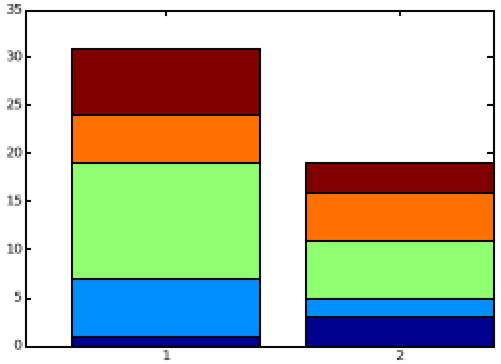
```
>> x = -2.9:0.2:2.9;
>> bar(x,exp(-x.*x))
```

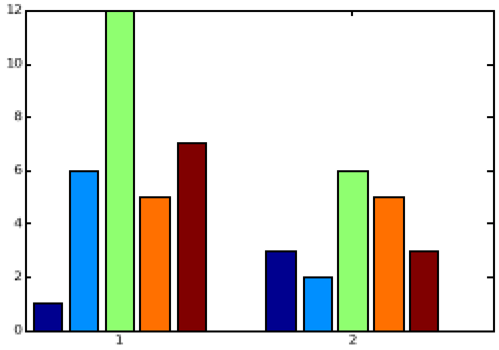


bar(...,width) *Chart with given width of the bars. By default, the width is 0.8 and width 1 causes the bars to touch.*

bar(...,' *style* ') *Chart with style for the given bars. The styles are 'group' (default style with grouped vertical bars) and 'stack' (stacked bars). If the matrix Y is (m,n), the grouped chart has m groups of n vertical bars.*

```
>> A=[1 6 12 5 7; 3 2 6 5 3];
>> bar(A,'stack')
>> bar(A,'group')
```



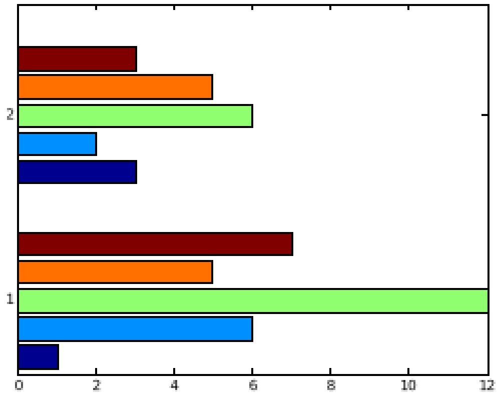


bar(...,colour) *The bars are all of the specified colour (r=red, g=green, b=blue, c=cyan, m=magenta y=yellow, k=black and w=white).*

Horizontal bar charts

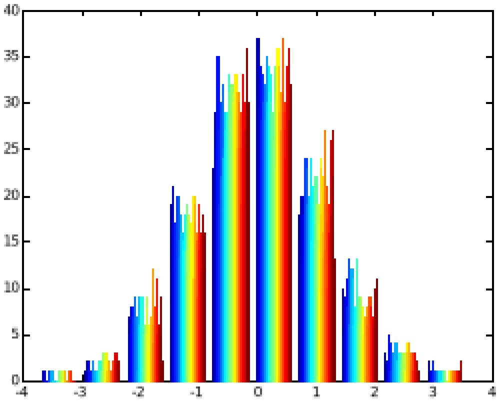
```
>> barh(A,'group')
```

barh(...)



hist(Y) *Draw the histogram relative to the frequency vector Y using 10 equal rectangles of equal base. If Y is a matrix, a histogram is constructed for each of its columns.*

```
>> Y=randn(100);
>> hist(Y)
```



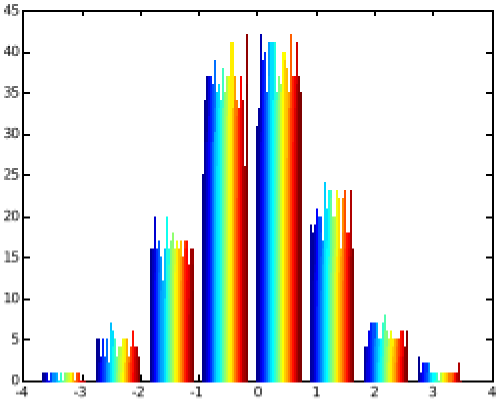
hist(Y,x)

Draw the histogram relative to the frequency vector Y using as many boxes as there are elements in the vector x and centring each box on the successive x values.

hist(Y,k)

Draw the histogram relative to the frequency vector Y using as many boxes as indicated by the scalar k.

```
>> hist(Y, 8)
```

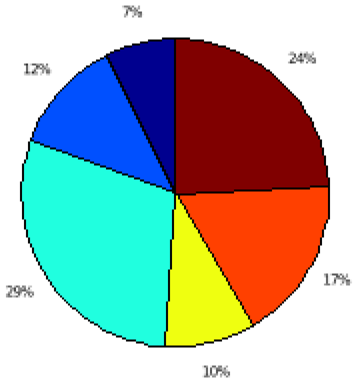


[n,x] = hist(...) *Returns the vectors n and x with the frequencies assigned to each box of the histogram and the values at which the boxes are centred.*

Make the pie chart relative to the vector of frequencies X

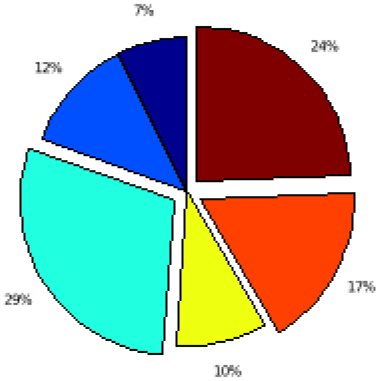
```
>> X=[3 5 12 4 7 10];
>> foot(X)
```

foot(X)



foot(X,Y) *Make the pie chart relative to the vector of frequencies X by shifting out the sectors where Yi 0≠*

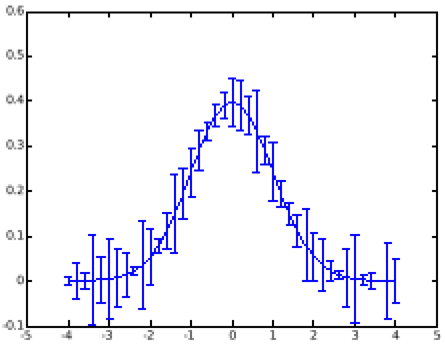
```
>> pie(X,[0 0 1 0 1 1 1])
```



Graph the vector x against the vector y with the errors specified in the vector e. Passing through each point (xi,yi) draw a vertical line of length 2ei whose centre is the point (xi,yi).

errorbar(x,y,e)

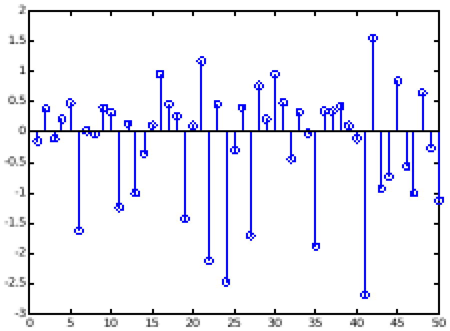
```
>> x = -4:.2:4;
y=(1/sqrt(2*pi))*exp(-(x.^2)/2);
e=rand(size(x))/10;
errorbar(x,y,e)
```



stem(Y)

Draw the cluster graph relative to the vector Y. Each point of Y is joined to the x-axis by a vertical line.

```
>> y=randn(50,1); stem(y)
```



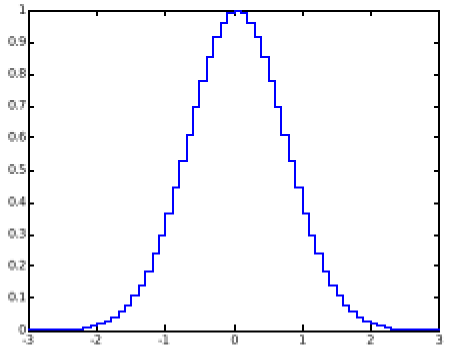
stem(X,Y) *Draw the cluster graph relative to the vector Y whose elements are given through the vector X.*

stairs(Y) *Draw the step graph relative to vector Y*

Step graph of vector Y with elements through vector X

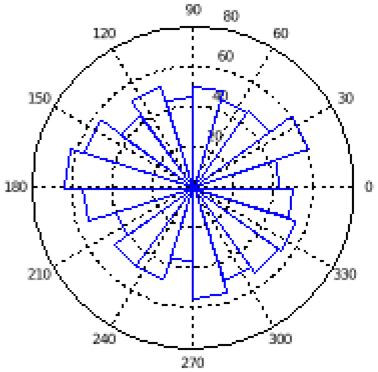
```
>> x=-3:0.1:3; stairs(x,exp(-x.^2))
```

stairs(X,Y)



rose(Y) *Draw the angular histogram relative to the vector Y, of angles in radians using 20 equal radii.*

```
>> y=randn(1000,1)*pi; rose(y)
```

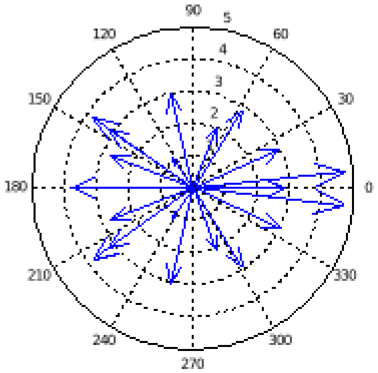


rose(Y,n) *Draw the angular histogram of the vector Y using n equal radii.*

rose(X,Y) *Draw the angular histogram relative to vector Y using radii measuring as specified in the elements of vector X.*

compas(Z) *Draw a diagram of arrows leaving the origin and whose magnitude and direction are determined by the modulus and argument of the complex component numbers of the vector Z. The arrow relative to the complex Zi joins the origin with the affix of Zi*

```
>> z=eig(randn(20,20)); compass(z)
```



compas(X,Y) *Equivalent to compas(X+i*Y)*

compas(Z,S)
or
compas(X,Y,S)

Specifies in S the type of line to be used in the arrows.

feather(Z) or
feather(X,Y)
or
feather(Z,S)
or
feather(X,Y,S)

It is the same as compas, with the only difference that the origin of the arrows is not at the origin of coordinates, but they come from equally spaced points on a horizontal line.

```
>> z=eig(randn(20,20)); feather(z)
```

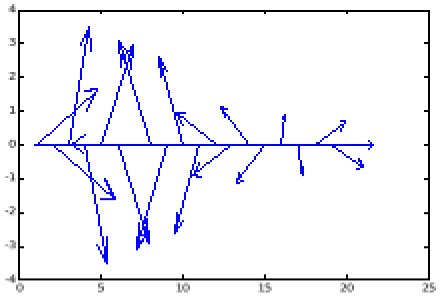


image image
image

3.2 CURVES IN EXPLICIT, IMPLICIT, PARAMETRIC AND POLAR CO-ORDINATES

image

T he most important MATLAB commands for representing two-dimensional curves in explicit, implicit and polar coordinates are presented in the following table.

plot(X,Y)	Draw the set of points (X,Y), where X and Y are row vectors. To plot a function y=f(x) it is necessary to know a set of points (X,f(X)), for which a vector variation interval X for the variable x must be defined initially. X and Y can be matrices of the same dimension, in which case a graph is made for each pair of rows and on the same axes. For complex values of X and Y the imaginary parts are ignored. For x = x (t) and y = y (t) with the given variation of t, graph the specified parametric plane curve >> x=0:0.1:6*pi; y=x.*sin(x); plot(x,y) image
plot(Y)	Plots the elements of the vector Y against their indices, i.e.

i.e. it gives the graph of the set of points (t,y t) t=1,2,...n (n=length(Y)). It is useful for plotting time series. If Y is a matrix, plot(Y) plots a graph for each column of Y, presenting them all on the same axes. If the components of the vector Y are complex, plot(Y) is equivalent to plot(real(Y),imag(Y)).

```
>> Y=[1,3,9,27,81,243,729]; plot(Y)
```

image

plot(X,Y,S)

Plot(X,Y) plot with the options defined in S. Usually, S consists of two digits in single quotes, the first of which sets the colour of the plot line and the second the character to be used in the plot. The possible values of colours and characters are, respectively, the following: y (yellow), m (magenta), c (cyan), r (red), g (green), b (blue), w (white), k (black), . (dots), o (circles), x (x-marks), + (plus sign), - (solid), * (stars), : (colon), -. (dashes and dot) and - (semi-solid).

```
>> plot([1,2,3,4,5,6,7,8,9],[1, 1/2,
1/3,1/4,1/5,1/6, 1/7,1/8,1/9],'r *')
```

image

plot(X1,Y1,S1,X2,Y2,S2,....

It combines, on the same axes, the graphs defined for the triplets (Xi,Yi,Si). This is a way of representing several functions on the same graph.

fplot('f',[xmin, xmax])	Graph the explicit function y=f(x) on the given interval of variation of x. >> fplot('x*sin(1/x)', [0,pi/16]) image
fplot('f',[xmin, xmax, ymin, ymax], S)	Plots the explicit function y=f(x)on the given x and y variation intervals, with the colour and character choices given by S >> fplot('x^2/(x+1)', [-12,12,-8, 8]) image
fplot('f',[xmin, xmax],...,t)	Graph f with tolerance t
fplot('f',[xmin, xmax],...,n)	Plot f with tolerance t as at least n+1 points
fplot('[f1,f2,...,fn]',[xmin, xmax , ymin, ymax], S)	Graph the functions f1, f2, ..., fn on the same axes in the specified ranges of variation of x and y and with the specified colour and character options defined in S >> fplot('[sin(x), sin(2*x), sin(3*x)]', [0,2*pi]) image >> fplot('[sin(x), sin(2*x), sin(3*x)]', [0,2*pi],'k *')

	image
ezplot('f', [xmin xmax])	Plots the explicit function y=f(x) or implicit function f(x,y)=k on the given interval of variation of x. The interval of variation of the variable may be omitted. >> ezplot('y*x^2+x*y^2=10',[-10,10]) image >> ezplot('x^2/(x^2-1)') image
ezplot('f',[xmin, xmax, ymin, ymax])	Graph the explicit function y=f(x) or implicit function f(x,y)=k on the given (or not given) intervals of variation of x and y. >> ezplot('x^2+y^3=1/2',[-10,10,-8,8]) image >> ezplot('x^2-y^4=1') image
ezplot(x,y)	Graph the plane parametric curve x = x(t) and y = y(t) over the domain 0 < t < 2 image >> ezplot('4*cos(t)-cos(4*t)', '4*sin(t)-sin(4*t)') image

ezplot('f', [xmin xmax])	Graph the parametric plane curve x = x(t) and y = y(t) over the domain xmin < t < xmax >> ezplot('t*sin(t)', 't*cos(t)',[-4*pi,4*pi image
ezplot('f')	Plot the curve f in implicit coordinates in [-2 image ,2 image]. >> ezplot('y^4-x^4-24*y^2+25*x^2=0') image
loglog(X,Y)	Performs graphs similar to plot(X,Y), but with logarithmic scale on both axes. >> x=0:0.1:pi; y=x.*sin(x); loglog(x,y) image
semilogx(X,Y)	It makes graphs similar to plot(X,Y), but with logarithmic scale on the X-axis and normal scale on the Y-axis. >> x=0:0.1:pi; y=x.*sin(x); semilogx(x,y) image
semilogy(X,Y)	It makes graphs similar to plot(X,Y), but with logarithmic scale on the Y-axis and normal scale on the X-axis. >> x=0:0.1:pi; y=x.*sin(x); semilogy(x,y)

	image
polar(α,r)	Draw the curve in polar coordinates r=r(α) >> t=0:0.1:2*pi;r=sin(t).*cos(t); polar(t,r) image
polar(α,r,S)	Draw the curve in polar coordinates r=r(α) with the line style given by S >> t=0:0.05:2*pi;r=sin(t).*cos(t); polar(t,r,'*r') image

ezpolar(r)
ezpolar(r,[a,b])
	Draw the curve in polar coordinates r=r(α) with the field of variation of at α[0 2π] if nothing is specified or between a and b if not specified
	>> ezpolar('1+cos(t)')
	image

fill(X,Y,C)	Draw the compact polygon whose vertices are the component pairs (Xi,Yi) of the column vectors X and Y. C is a vector of the same dimension as X and Y, containing the colours Ci of each point (Xi,Yi). The values of Ci can be: 'r', 'g', 'b', 'c', 'm', 'y', 'w', 'k', whose meanings we already know. If C is a single character, all the points of the polygon shall be painted in the colour corresponding to the

character. If X and Y are matrices of the same dimension, several polygons corresponding to each pair of column vectors (X.j,Y.j) will be represented simultaneously. In this case, C can be a row vector whose elements Cj determine the unique colour of each polygon corresponding to the pair of column vectors (X.j,Y.j). C can also be a matrix of the same dimension as X and Y, in which case its elements determine the colours of each point (Xij,Yij) of the set of polygons.

```
>> t = (1/16:1/8:1)'*2*pi;x = sin(t);y = cos(t);
fill(x,y,'r')
```

image

fill(X1,Y1,C1,...)

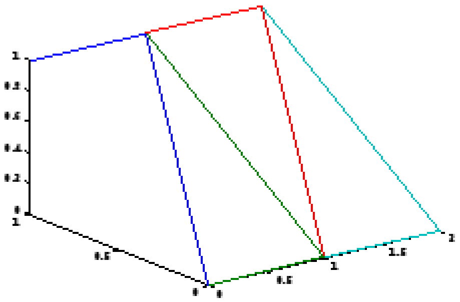
Draw the compact polygon whose vertices are given by the points (Xi, Yi, Ci)

3.3 WARPED CURVES

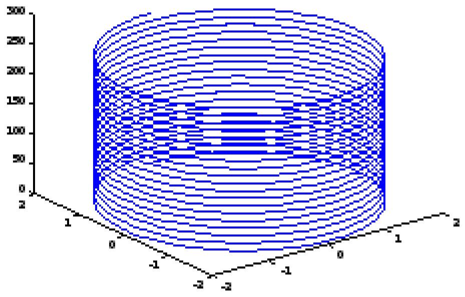
M ATLAB incorporates commands that allow you to represent warped curves in three dimensions. The most important of these are listed in the table below.

plot3(X,Y,Z) *Plots the set of points (X,Y,Z), where X, Y and Z are row vectors. X, Y and Z can be parametric coordinates or matrices of the same dimension, in which case one plot is made for each row triplet and on the same axes. For complex values of X, Y and Z, the imaginary parts are ignored.*

```
>> X = [0 1 1 2;1 1 2 2;0 0 1 1];
Y = [1 1 1 1;1 0 1 0;0 0 0 0];
Z = [1 1 1 1;1 0 1 0;0 0 0 0];
>> plot3(X,Y,Z)
```



```
>> t=0:pi/100:20*pi;plot3 (2*sin(2*t),2*cos(2*t),4*t)
```



plot3(X,Y,Z,S) *Plot(X,Y,Z) plot with the options defined in S. Usually S consists of two digits in single quotes, the first of which sets the colour of the plot line and the second the character to be used in the plot. The possible values of colours and characters have already been described when explaining the plot command*

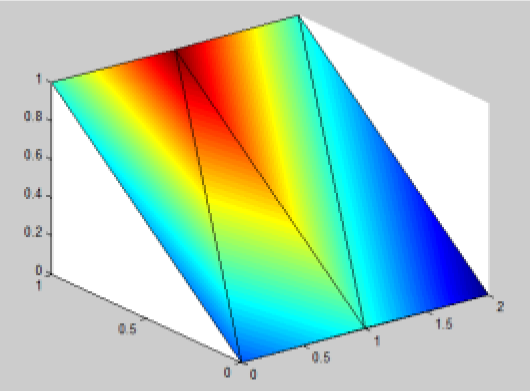
plot3(X1,Y1,Z1,S1, X2,Y2,Z2,S2, X3,Y3,Z3,S3,...)

It combines, on the same axes, the graphs defined for the triplets (Xi,Yi,Zi,Si). This is a way of representing several functions on the same graph.

fill3(X,Y,Z,C)

Draw the compact polygon whose vertices are the triplets of components (Xi,Yi,Zi) of the column vectors X, Y and Z. C is a vector of the same dimension as X, Y and Z, containing the colours Ci of each point (Xi,Yi,Zi). The values of Ci can be 'r','g','b','c','m','y','w','k', whose meanings we already know. If C is a single character, all the points of the polygon will be painted in the colour corresponding to the character. If X, Y and Z are of the same dimension, several polygons corresponding to each triplet of column vectors (X.j,Y.j,Z.j) will be represented simultaneously. In this case, C can be a row vector whose elements Cj determine the unique colour of each polygon corresponding to the triplet of column vectors (X.j,Y.j,Z.j). C can also be a matrix of the same dimension as X, Y and Z, in which case its elements determine the colours of each point (Xijk,Yijk,Zijk) of the set of polygons.

```
>> X = [0 1 1 2;1 1 2 2;0 0 1 1];
Y = [1 1 1 1;1 0 1 0;0 0 0 0];
Z = [1 1 1 1;1 0 1 0;0 0 0 0];
C = [0.5000 1.0000 1.0000 0.5000;
1.0000 0.5000 0.5000 0.1667;
0.3330 0.3330 0.5000 0.5000];
fill3(X,Y,Z,C)
```



fill3(X1,Y1,Z1,C1, X2,Y2, *Z2* , C2,...) *Draw the compact polygon whose vertices are given by the points (Xi, Yi, Zi, Ci).*

ezplot3(x(t),y(t),z(t)) *Draw the warped curve defined by its three parametric components*

ezplot3(x(t),y(t),z(t), [tmin,tmax]) *Draw the warped curve defined by its three parametric components for a variation field of the given parameter*

```
>> ezplot3('sin(t)','cos(t)','t',[0,6*pi])
```

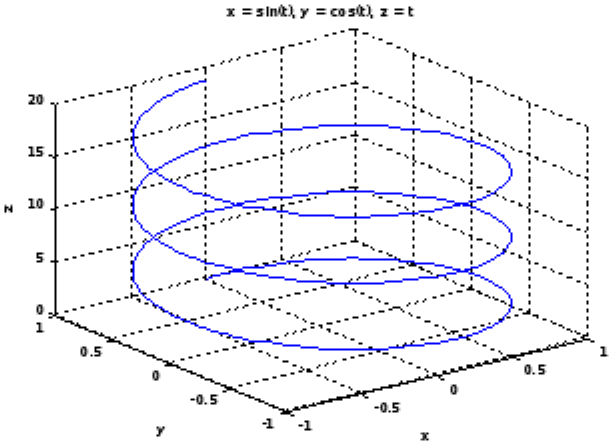

x = sin(t), y = cos(t), z = t
z
20
15
10
5
0
y
1
0.5
0
-0.5
-1
x
-1
-0.5
0
0.5
1

image image
image

3.4 EXPLICIT AND PARAMETRIC SURFACES CONTOUR LINES

image

M ATLAB incorporates commands that allow the representation of surfaces of equation z=f(x,y) . The first step is to use the meshgrid command, which defines the matrix of points (X,Y) on which the function is evaluated to represent it.

The surf command is then used to perform the surface representation.

The mesh command is also used to represent a mesh graph, which is defined by a function z=f(x,y) , such that the surface points are represented on a grid, the result of lifting the values of z given by f(x,y) over the corresponding points in the (x,y) plane. The appearance of a mesh graph is like a fishing net, with the surface points on the knots of the net. Actually, it is a surface graph whose graph is in the form of a net.

It is also possible to represent the contour lines of a surface using the contour command. These curves are characterised by the fact that they are points (x,y) over which the value of f(x,y) is constant.

The following table shows the MATLAB commands for representing meshes, contour lines and surfaces in both explicit and parametric coordinates.

[X,Y] = meshgrid(x,y)	Transforms the given definition field of the x and y variables of the function to be represented z=f(x,y)

————

image

into matrix arguments usable by the surf and mesh commands to obtain surface and mesh plots, respectively.

surf(X,Y,Z,C)

Represents the explicit surface z=f(x,y) or the parametric x=x(t,u), y=y(t,u), z=z(t,u), drawing with the colours specified in C. The argument C can be ignored.

————

image

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);

Z = X .* exp(-X.^2 - Y.^2);surf(X,Y,Z)
```

image

surfc(X,Y,Z,C)

Represents the explicit surface z=f(x,y) or the parametric x=x(t,u), y=y(t,u), z=z(t,u), together with the corresponding contour plot (contour lines projected on the XY plane).

————

image

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);

Z = X .* exp(-X.^2 - Y.^2);surfc(X,Y,Z)
```

image

surfl(X,Y,Z)

Represents the explicit surface z=f(x,y) or the parametric surface x=x(t,u), y=y(t,u), z=z(t,u), with the shaded drawing

```
>> r=(0:0.1:2*pi)';

t=(-pi:0.1:2*pi);

X=cos(r)*sin(t);
```

```
Y=sin(r)*sin(t);

Z=ones(1,size(r))'*t;

surfl(X,Y,Z)
```

image

mesh(X,Y,Z,C)

Represent the explicit surface z=f(x,y) or the parametric surface x=x(t,u),y=y(t,u), z=z(t,u), drawing the grid lines composing the mesh with the colours specified in C (optional).

image

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);

Z = X .* exp(-X.^2 - Y.^2);mesh(X,Y,Z)
```

image

meshz(X,Y,Z,C)

It represents the explicit surface z=f(x,y) or the parametric x=x(t,u),y=y(t,u), z=z(t,u) with a kind of curtain at the bottom.

meshc(X,Y,Z,C)

Represents the explicit surface z=f(x,y) or the parametric x=x(t,u),y=y(t,u), z=z(t,u) together with the corresponding contour plot (contour lines projected on the XY plane).

image

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);

Z = X .* exp(-X.^2 - Y.^2);meshc(X,Y,Z)
```

image

contour(Z)

Draw the contour plot (contour lines) for the Z

image

matrix. The number of contour lines to be used is chosen automatically.

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);

Z = X .* exp(-X.^2 - Y.^2);

>> contour(Z)
```

image

contour(Z,n)

Draw the contour plot (contour lines) for the Z matrix using n contour lines.

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);

Z = X .* exp(-X.^2 - Y.^2);

>> contour(Z)
```

image

contour(x,y,Z,n)

image

Draw the contour plot (contour lines) for the Z matrix using in the X and Y axes the scaling defined by the x and y vectors (n contour lines).

```
>> r=(0:0.1:2*pi)';t=(-pi:0.1:2*pi);

X=cos(r)*cos(t);Y=sin(r)*sin(t);Z=ones(1,size(r))'*t;

>> contour(X,Y,Z)
```

image

contour3(Z),

Draw the contour plots in 3 dimensions.

contour3(Z,n) and
contour3(x,y,Z,n)
	>> r=(0:0.1:2*pi)';t=(-pi:0.1:2*pi);
	X=cos(r)*cos(t);Y=sin(r)*sin(t);Z=ones(1,size(r))'*t;
	>> contour3(X,Y,Z)
	image

contourf(...)	Draw a contour plot and fill in the areas between the isolines. >> r=(0:0.1:2*pi)';t=(-pi:0.1:2*pi); X=cos(r)*cos(t);Y=sin(r)*sin(t);Z=ones(1,size(r))'*t; >> contourf(X,Y,Z) image
pcolor(X,Y,Z) ________ image	Draws a contour plot (contour lines) for the (X,Y,Z) matrix using a representation based on colour densities. Often referred to as a density plot >> [X,Y] = meshgrid(-2:.2:2, -2:.2:2); Z = X .* exp(-X.^2 - Y.^2);meshc(X,Y,Z) >> pcolor(X,Y,Z) image
trimesh(Tri,X,Y,Z,C)	Displays triangles defined in the Tri matrix as a mesh. Each row of the Tri matrix defines a single triangular face and C defines the colours as in surf. The C argument is optional.

trisurf(Tri,X,Y,Z,C)	Displays triangles defined in the Tri matrix as a surface. Each row of the Tri matrix defines a single triangular face and C defines the colours as in surf. The C argument is optional.

image image
image

3.5 THREE-DIMENSIONAL GEOMETRIC SHAPES

image

W ith MATLAB it is possible to plot cylinders, spheres, bar charts, sections, stems, waterfall and other three-dimensional geometric shapes. The following table summarises the commands for this purpose.

bar3(Y)	Bar chart relative to the vector of frequencies Y. If Y is a matrix, the multiple bar chart is obtained for each row of Y >> bar3(rand(4,4)) image
bar3(x,Y)	Bar chart relative to the vector of frequencies Y where x is a vector defining the spaces on the X-axis to place the bars.
bar3(...,width)	Chart with given width of the bars. By default, the width is 0.8 and width 1 causes the bars to touch.
bar3(...,'style')	Chart with style for the given bars. The styles are 'detached' (default style) 'grouped' (style with grouped vertical bars) and 'stacked' (stacked bars).

bar3(...,colour)	The bars are all of the specified colour (r=red, g=green, b=blue, c=cyan, m=magenta y=yellow, k=black and w=white).
comet3(z)	Comet graph relative to the z vector
comet3(x,y,z)	Parametric kite graph (x(t), y(t), z(t)) >> t = -pi:pi/500:pi;comet3(sin(5*t),cos(3*t),t) image
[X,Y,Z] = cylinder	Gives the coordinates of the unit cylinder
[X,Y,Z] = cylinder(r(t))	Gives the coordinates of the cylinder generated by the curve r
[X,Y,Z] = cylinder(r(t),n)	Gives the coordinates of the cylinder generated by the curve r with n points on the circumference horizontal section aligned with the Z axis (n = 20 by default).

cylinder(...)	Graph the above cylinders
	>> t = 0:pi/10:2*pi;
	[X,Y,Z] = cylinder(2+cos(t));
	surf(X,Y,Z)
	image

sphere	Plots the unit sphere using 20x20 faces
sphere (n)	Generate a sphere using nxn faces

	>> sphere(100) image
[X,Y,Z] = sphere(n)	Gives the coordinates of the sphere in three matrices (n+1)x(n+1)
	Draw slices along the x, y , z directions in the volume V (array mxnxp) defined by the vectors (sx,sy,sz)
slice(V,sx,sy,sz)	Draw slices defined by the vectors (sx,sy,sz) in the volume V defined by the three-dimensional arrays (X,Y,Z).
slice(X,Y,Z,V,sx,sy,sz)	Draw slices in the volume V defined by the matrices (XI,YI,ZI) that generate a surface
slice(V,XI,YI,ZI)	Draw slices defined by the arrays (XI,YI,ZI) that generate a surface in the volume V defined by the three-dimensional arrays (X,Y,Z).
slice(X,Y,Z,V,XI,YI,ZI)	Draws cuts according to the specified interpolation method (linear for linear, cubic for cubic and nearest for neighbouring)

slice(...,'method')	>> [x,y,z] = meshgrid(-2:.2:2, -2:.25:2, -2:.16:2);
	v = x .* exp(-x.^2 - y.^2 - z.^2);
	slice(x,y,z,v,[-1.2 .8 2],2,[-2 -.2])
	image

stem3(Z)	Draw the sequence Z as a stem graph in the (x,y) plane.

stem3(X,Y,Z)	Draws the sequence Z at the values specified by X and Y
stem3(...,'fill')	Fill the circles at the tips of the stems with colour.
stem3(...,S)	Make the stem chart with S specifications (colour, ...).

```
>> X = linspace(0,1,10);

Y = X./2;

Z = sin(X) + cos(Y);

stem3(X,Y,Z,'fill')
```

image

image image
image

3.6 SPECIALISED GRAPHICS

image

M ATLAB provides commands for plotting area plots, box plots, three-dimensional pie charts, Pareto charts and other specialised plots. The following table presents the syntax of these commands.

area(Y)
area(X,Y)
area(...,ymin)
	Make the area graph relative to the vector of frequencies Y
	Make the area graph relative to the vector of frequencies Y whose elements are given through the vector X.
	Specifies the lower boundary in the y-direction of the fill area.
	>> Y = [1, 5, 3;3, 2, 7;1, 5, 3;2, 6, 1];area(Y)
	image

box on, box off	Enables and disables boxes on axes for 2-D and 3-D charts

comet(y)
comet(x,y)
	Make the kite plot relative to the vector of frequencies Y
	Make the kite graph relative to the vector of frequencies Y whose elements are given through the vector X.
	>> t = -pi:pi/200:pi;comet(t,tan(sin(t))-sin(tan(t))))
	image

ezcontour(f)	Contour plot of f(x,y) at [-2π, 2 π]x[-2 π, 2 π]
ezcontour(f,domain)	Contour plot of f(x,y) in the given domain
ezcontour(...,n)	Contour plot of f(x,y) on the mesh nxn
	>> ezcontour('sqrt(x^2 + y^2)')
	image
	Contour plot of f(x,y) filled at [-2π, 2 π]x[-2 π, 2 π]
ezcontourf(f)	Contour plot of f(x,y) filled in the given domain
ezcontourf(f,domain)	Contour plot of f(x,y) filled in the nxn-grid
ezcontourf(...,n)	>> ezcontourf('sqrt(x^2 + y^2)')
	image
	Mesh graph of f(x,y) filled in [-2π, 2 π]x[-2 π, 2 π]
ezmesh(f)	Mesh plot of f(x,y) filled in the given domain
ezmesh(f,domain)	Mesh plot of f(x,y) filled on the nxn-grid
ezmesh(...,n)	Mesh graph for x=x(t,u), y=y(t,u), z=z(t,u) t,u ∈[-2 π, 2 π]
ezmesh(x,y,z)	Mesh plot for x=x(t,u), y=y(t,u), z=z(t,u) t,u ∈domain
ezmesh(x,y,z, domain)	Mesh chart on a domain-centred disk
ezmesh(..., 'circ')	>> ezmesh('sqrt(x^2 + y^2)')
	image
ezmeshc(f)	Performs a combination of mesh and contour plotting

ezmeshc(f,domain)
ezmeshc(...,n)
ezmeshc(x,y,z)
ezmeshc(x,y,z, domain)
ezmeshc(..., 'circ')
	>> ezmeshc('sqrt(x^2 + y^2)')
	image

ezsurf(f)
ezsurf(f,domain)
ezsurf(...,n)
ezsurf(x,y,z)
ezsurf(x,y,z, domain)
ezsurf(..., 'circ')
	Produces a coloured surface graph
	>> ezsurf('sqrt(x^2 + y^2)')
	image

ezsurfc(f)
ezsurfc(f,domain)
ezsurfc(...,n)
ezsurfc(x,y,z)
ezsurfc(x,y,z, domain)
ezsurfc(..., 'circ')
	Performs a combination of surface and contour plotting
	>> ezsurfc('sqrt(x^2 + y^2)')
	image

ezplot3(x,y,z)	Parametric curve 3D x=x(t), y=y(t), z=z(t) t ∈[-2 π, 2 π]

ezplot3(x,y,z,domain)	3D parametric curve x=x(t), y=y(t), z=z(t) t ∈domain
ezplot3(..., 'animate')	3D parametric curve with animation >> ezplot3('cos(t)','t.*sin(t)','sqrt(t)') image
ezpolar (f) ezpolar (f, [a,b])	Plot the polar curve r=f(c) with c ∈[0, 2 π]. Plot the polar curve r=f(c) with c ∈[a, b]. ezpolar('sin(2*t).*cos(3*t)',[0 pi]) image

pareto (Y)
pareto(X,Y)
	Make the Pareto chart relative to the vector of frequencies Y
	Make the Pareto chart relative to the vector of frequencies Y whose elements are given through the vector X.
	>> code_lines = [200 120 555 608 1024 101 57 687];
	encoders = ...
	{'Fred','Ginger','Norman','Max','Julia','Wally','Heidi','Pat'};
	pareto(code_lines, encoders)
	title('Lines of code per programmer')
	image

foot3(X)	Three-dimensional pie chart for the frequencies X
pie3(X,explode)	Three-dimensional pie chart broken down into three-dimensional sectors

	>> pie3([2 4 3 5],[0 1 1 0],{'North','South','East','West'}) image
plotmatrix (X,Y)	Scatter plot of X vs. Y columns >> x = randn(50,3); y = x*[-1 2 1;2 0 1;1 -2 3;]';plotmatrix(y) image

stairs(Y)
stairs(X,Y)
stairs(...,LineSpec)
	Ladder chart with the elements of Y
	Ladder graph of the elements of Y corresponding to those of X
	Line specifications for the ladder chart
	>> x = linspace(-2*pi,2*pi,40);stairs(x,sin(x))
	image

scatter(X,Y,S,C)
scatter(X,Y)
scatter(X,Y,S)
scatter(...,marker)
scatter(...,'filled')
	Scatter plot for the vectors (X,Y) according to the colours C and the area of each marker S. You can also obtain the filled graph (fill option) and use different types of markers.

scatter3(X,Y,Z,S,C)
scatter3(X,Y,Z)
scatter3(X,Y,Z,S)
	Three-dimensional scatter plot for the vectors (X,Y,Z) according to the colours C and the area of each marker S. It is also possible to obtain the filled graph (fill option) and to use different types of markers.

scatter3(...,marker)
scatter(...,'filled')
	>> x=(0:0.1:4);
	>> scatter(x,cos(x))
	image
	>> x=(0:0.1:2*pi);
	>> scatter3(x,cos(x),sin(x))
	image

image image
image

3.7 2D AND 3D GRAPHICS OPTIONS

image

M ATLAB presents many possibilities for handling graphical options such as titles, labels, placement, viewpoint, etc. The following table presents the most important ones.

title('text')	Adds text as chart title at the top of the chart in 2-D and 3-D charts
xlabel('text')	Places text next to x-axis in 2-D and 3-D graphics
ylabel('text')	Places the text next to the y-axis in 2-D and 3-D graphics.
zlabel('text')	Places text next to z-axis on a 3-D graphic
clabel(C,h)	Rotate labels and place them inside the contour lines.
clabel(C,h,v)	Creates labels only for the contour levels given by the vector v and rotates and places them inside the contour lines.

datetick(axis)

Label the marks on the specified axis ('x', 'y' or 'z') based on the maximum and minimum of the specified axis.

datetick(axis, date)

Label the specified axis marks ('x', 'y' or 'z') with the given date format (an integer between 1 and 28).

legend('string1', string2',...)

Places the legends specified by the strings in n consecutive graphs.

legend(h, 'string1', string2',...)

Places the legends specified by the strings in the managed graphics according to the vector h

legend('off'),

Removes legends from current axes

text(x,y,'text')

Places the text at the point (x,y) within the 2-D graphic.

text(x,y,z,'text')

Places the text at the point (x,y,z) on the 3-D graphic.

gtext('text')

Allows text to be positioned at a point selected with the mouse within a 2-D graphic.

grid

Places grids on the axes of a 2-D or 3-D chart. The grid on option places the grids and greed off

image	removes them. The grid option toggles between on and off
hold	Allows you to keep the existing chart with all its properties, so that the next chart you make will be on the same axes and overlay the existing one. The hold on option activates the option and hold off removes it. The hold option switches between on and off. Valid for 2-D and 3-D
axis([xmin xmax ymin ymax ymax zmin zmax])	Places the maximum and minimum values for the X, Y and Z axes on the current graph.
axis('auto')	Set the axes to the default automatic scale (given by xmin = min(x), xmax = max(x) and y free).
axis(axis)	Freezes the axis scaling at the current limits, so that when another graph is placed on the same axes (with hold on) the scale does not change.
V=axis	Gives the 4-element vector V, containing the scale of the current graph
axis('xy')	Places Cartesian coordinates with the origin at the bottom left of the graph.
Axis('tight')	Places the limits of the axes in the range of the data.

axis('ij')	Places coordinates with the origin at the top left of the graph
axis('square')	Turns the plotting rectangle into a square, so that the figures are bulged.
axis('equal')	Set the same scale factor for both axes.
axis('normal')	Remove the square and equal options
axis('off')	Remove labels and marks from axes and grids, keeping the chart title and texts placed on it with text and gtext.
axis('on')	Replace labels, markings and axle grids.
subplot(m,n,p) ________ image	Divide the graph window into mxn subwindows and place the current graph in the p-th window, starting from the top left and counting from left to right until the end of the line, and move on to the next one.
plottyy(X1,Y1,X2,Y2)	Label the graph of X1 versus Y1 on the left, and the graph of X2 versus Y2 on the right.
plottyy(X1,Y1,X2,Y2, function')	Same as the previous command, but the function can be plot, loglog, semilogx, semilogy, item or any h=function(x,y).
	Same as the previous command, but using each function fi for the pair (XiYi).

plottyy(X1,Y1,X2,Y2, 'function1','function2')	
axis([xmin xmax ymin ymax zmin zmax])	It places the variation intervals of the axes at the indicated values. It also accepts the options 'ij', 'square', 'equal', etc, identical to those already seen for two dimensions.
view([x,y,z])	Places the point of view of the figure at the Cartesian coordinate point (x,y,z).
view([az, el])	Places the angle of view of the figure at the point of azimuth (horizontal rotation) 'az' and elevation (vertical elevation) 'el'.
hidden	Controls the presence of hidden lines in the chart. Hidden lines appear with hidden on and disappear with hidden off.
shading	Controls the type of shading of a surface created with the surf, mesh, pcolor, fill and fill3 commands. The shading flat option sets a smooth shading, the shading interp option sets a dense shading and the shading faceted option (default option) sets a normal shading.
colourmap(M)	It places the matrix M as the current colour map. M must have three columns and contain values only between 0 and 1. It can also be a matrix whose

	rows are RGB vectors of the type [r g b]. There are already defined matrices M in MATLAB, which are: bone(p), contrast(p), cool(p), copper(p), flag(p), gray(p), hsv(p), hot(p), jet(p), pink(p), prism(p) and white(p). All matrices have 3 columns and p rows. For example, the syntax colourmap(hot(8)) sets the hot(8) matrix as the current colourmap (the complete colour system of the current figure).
brighten(p)	Adjusts the illumination of the figure. If 0<p<1, the figure will be bright, and if -1<p<0, the figure will be dark. The variation of p is the interval (-1,1), and as values of p approach -1, the figure becomes darker, while as values of p approach 1, the figure becomes brighter.
image(A) ———— image	It produces a two-dimensional image with brightnesses proportional to the elements of the matrix A, and is used to display photographs and drawings adapted to the brightnesses given in the matrix A. Each element (m,n) of the matrix A affects one cell of the drawing.
pcolor(A) ———— image	It produces a two-dimensional figure with colours proportional to the elements of the matrix A, and is used to display abstract mathematical objects with varying colours. Each element (m,n) of the matrix A affects a grid of the figure.
caxis([cmin cmax]) ————	Places the minimum and maximum values of the colour scale (defined by colormap and intrinsically related to the divisions made on the axes, via the

image	grids) for a chart. Therefore, it allows to use only a subset of colours of the one defined by colormap for the figure.
figure(h) or h=figure	Creates the figure as an object of name h, and places it as an ordinary figure. The command gcf(h) is used to refer any property to the figure h. The close(h) command closes the figure h. The command whitebg(h) changes the background colour of the figure h. The clf command closes the current figure. The graymon command sets the grayscale. The command newplot determines the axes to make a new figure. The refresh command redraws the figure
axes(e) or e=axes ________ image	Creates the axes as an object with name e in the current figure. The command gca(e) is used to refer any property to the axes e. The command cla is used to delete all objects referring to the current axes.

l=line(x,y) or
l=line(x,y,z)
	Creates, as an object of name l, the line joining the points (X,Y) in the plane or (X,Y,Z) in space.

p=patch(X,Y,C) or
patch(X,Y,Z,C)
	Creates an opaque polygonal area that is defined by the set of points (X,Y) in the plane or (X,Y,Z) in space, and whose colour is given by C, as an object of name p

s=surface(X,Y,Z,C)	Creates the parametric surface defined by X, Y and Z and whose colour is given by C as an object with the name s

i=image(C)	Creates the image defined by the given colours in the matrix C as an object of name i
t=text(x,y,'string') or t=text(x,y,z,'string')	Creates the text defined by the string, located at point (x,y) in the plane, or at point (x,y,z) in space.
set(h, 'property1', property2', ...)	Places the specified properties on the h object (gca for axis boundaries, gcf for colours, gco, gcbo, gcbd, etc.).
get(h, 'property')	Returns the current value of the given property for the object h
object=gco	Returns the current object of the current figure
rotate(h, [a, e],α, [p,q,r])	Rotate the object h an angleα , along the azimuthal axes a and elevation e, the origin being the point (p,q,r)
reset(h)	Updates all properties assigned to the object h and sets its default properties
delete(h)	Deletes the object h

I n addition, the most typical properties to be placed on MATLAB graphics objects are the following:

Object	Properties	Possible values
Figure	Colour (background colour)	y', 'm', 'c', 'r', 'g', 'b', 'w', 'k'.
	ColorMap (map colour)	hot(p), gray(p), pink(p), ...
	Position (position on screen)	[left, button, width, height].
	Name	chain with the name
	MinColorMap (min. no. of colour)	minimum number of colours for the map
	NextPlot (next plot mode)	new, add, replace
	NumberTitle (no. in figure)	on, off
	Units (units of measurement)	pixels, inches, centimeters, points
	Resize (figure size with mouse)	on (can be changed), off (no)
Axes	Box (box for the graphic)	on, off

Colour (colour of axles)	y', 'm', 'c', 'r', 'g', 'b', 'w', 'k'.
GridLineStyle (line for mesh)	'-', '—', ':', '-.'
Position (position on screen)	[left, button, width, height].
TickLength (length between tick marks)	a numerical value
TickDir (tick address)	in, out
Units (units of measurement)	pixels, inches, centimeters, points
View	[azimuth, elevation] [azimuth, elevation
FontAngle (font angle)	normal, italic, oblique
FontName (font name)	text with the name of the source
FontSize	numerical value
FontWeight (font weight)	light, normal, demi, bold

	DrawMode (drawing mode)	normal, fast
	Xcolor, Ycolor, Zcolor (colour axes)	[min, max] [min, max] [min, max] [min, max] [min, max] [min, max
	XDir,YDir, ZDir (axis direction)	normal (they grow from left to right), reverse
	XGrid,YGrid, Zgrid (grids)	on, off
	XLabel,YLabel, Zlabel (labels)	string with the text of the tags
	XLim,YLim,Zlim (limit values)	[min, max] (range of variation)
	XScale,YScale,ZScale (scales)	linear (linear), log(logarithmic)
	XTick,YTick,ZTick (marks)	[m1,m2,...] (location of marks on axis)
Line	Colour (line colour)	y', 'm', 'c', 'r', 'g', 'b', 'w', 'k'.
	LineStyle (line style)	'-', '—', ':', '-.', '+', '*', '.', 'x'

	LineWidth (line width)	numerical value
	Visible (visible or not visible line on screen)	on, off
	Xdata,Ydata,Zdata (coordinate.)	set of coordinates of the line
Text	Colour (text colour)	y', 'm', 'c', 'r', 'g', 'b', 'w', 'k'.
	FontAngle (font angle)	normal, italic, oblique
	FontName (font name)	text with the name of the source
	FontSize	numerical value
	FontWeight (font weight)	light, normal, demi, bold
	HorizontalAlignment (hor. adjustment)	left, centre, right
	VerticalAlignment (vertical adjustment)	top, cap, middle, baseline, bottom
	Position (position on screen)	[x, y, z] (location point)

	Rotation (text orientation)	0,± 90, ±180, ±270
	Units (units of measurement)	pixels, inches, centimeters, points
	String (text string to be placed)	string with the text
Surface	Cdata (colour of each point)	colour matrix
	Edgecolor (grid colour)	'y', 'm',..., none, flat, interp
	Facecolor (colour of faces)	'y', 'm',..., none, flat, interp
	LineStyle (line style)	'-', '—', ':', '-.', '+', '*', '.', 'x'
	LineWidth (line width)	numerical value
	MeshStyle (lines in rows and col.)	row, colum, both
	Visible (line visible or not on screen)	on, off
	Xdata,Ydata,Zdata	set of surface coordinates

	(coordinate.)	
Patch	Cdata (colour of each point)	colour matrix
	Edgecolor (axle colour)	'y', 'm',..., none, flat, interp
	Facecolor (colour of faces)	'y', 'm',..., none, flat, interp
	LineWidth (line width)	numerical value
	Visible (visible or not visible line on screen)	on, off
	Xdata,Ydata,Zdata (coordinate.)	set of surface coordinates
Image	Cdata (colour of each point)	colour matrix
	Xdata,Ydata (coordinates)	set of image coordinates

Let us look at some illustrative examples.

```
>> x=linspace(0,2,30);

y=sin(x.^2);
```

```
plot(x,y)

text(1,0.8, 'y=sin(x^2)')

hold on

z=log(sqrt(x));

plot(x,z)

text(1,-0.1, 'y=log(sqrt(x))')

xlabel('X-axis');

ylabel('Y-axis');

title('Sine and logarithmic graph');
```

image

```
>> subplot(2,2,1);

ezplot('sin(x)',[-2*pi 2*pi])

subplot(2,2,2);

ezplot('cos(x)',[-2*pi 2*pi])

subplot(2,2,3);

ezplot('csc(x)',[-2*pi 2*pi])

subplot(2,2,4);

ezplot('sec(x)',[-2*pi 2*pi])
```

image

```
>> [X,Y]=meshgrid(-2 :0.05:2);
```

```
Z=X.^2-Y.^2;

subplot(2,2,1)

surf(X,Y,Z)

subplot(2,2,2)

surf(X,Y,Z),view(-90,0)

subplot(2,2,3)

surf(X,Y,Z),view(60,30)

subplot(2,2,4)

surf(X,Y,Z),view(-10,30)
```

image

```
>> [X,Y]=meshgrid(-2 :0.05:2);

Z=X.^2-Y.^2;

surf(X,Y,Z),shading interp,brighten(0.75),colourmap(gray(5))
```

image

E xercise 1. Represent the surface defined by the equation:

image

```
>> [x,y]=meshgrid(0:0 .05:2,-2:0.05:2);

>> z=y.^2.*(x-1).^2./(y.^2+(x-1).^2);

>> mesh(x,y,z),view([-23,30])
```

image

We could also have represented the surface as follows:

```
>> ezsurf('y^2*(x-1)^2/(y^2+(x-1)^2)')
```

image

Exercise 2. Let f:R^2 R→ be a function defined by:

image

image

Represent it graphically in an environment of (0,0).

```
>> [x,y]=meshgrid(-1/100:0.0009:1/100,-1/100:0.0009:1/100);

>> z=(1-cos(x)).*sin(y)./(x.^3+y.^3);

>> surf(x,y,z)

>> view([50,-15])
```

image

Plot the polar curves on the same graph adjacent to each other:

image

Also represent their intersection.

```
>> a=0:.1:2*pi;

>> subplot(1,2,1)

>> r=sqrt(cos(2*a));

>> polar(a,r)
>> title(' r = sqrt(Cos(2a))')

>> subplot(1,2,2)

>> r=sin(2*a);

>> polar(a,r)

>> title(' r = Sen(2a)')
```

image

To see the intersection of the two curves, we plot them both on the same axes.

```
>> a=0:.1:2*pi;

>> r=sqrt(cos(2*a));

>> polar(a,r)

>> hold on;

>> r=sin(2*a);

>> polar(a,r)
```

image

Exercise 4. Represent the volume generated by rotating the cubic $12x - 9x^2 + 2x^3$ around the OX axis, between x=0 and x=5/2.

The surface of revolution will have the equation $y^2 + z^2 = (12x-9x^2 + 2x^3)^2$, and to graph it can be parametrised as follows:

$x = t$, $y = \text{Cos}(u)*(12t - 9t^2 + 2t^3)$, $z = \text{Sen}(u)*(12t - 9t^2 + 2t^3)$

```
" t=(0:.1:5/2);
```

```
"u=(0:.5:2*pi);

" x=ones(size(u))'*t;

» y=cos(u)'*(12*t-9*t.^2+2*t.^3);

» z=sin(u)'*(12*t-9*t.^2+2*t.^3);

"surf(x,y,z)
```

image

R epresent the volumes generated by rotating the ellipse: image around the OX axis and around the OY axis.

We represent the generated figures on the same graph, but only in their positive half. The final volume will be twice the volume represented. The equation of the figure of revolution around the OX axis is $y^{2} + z^{2} = 9(1-x^{2}/4)$, and is parametrised as follows:

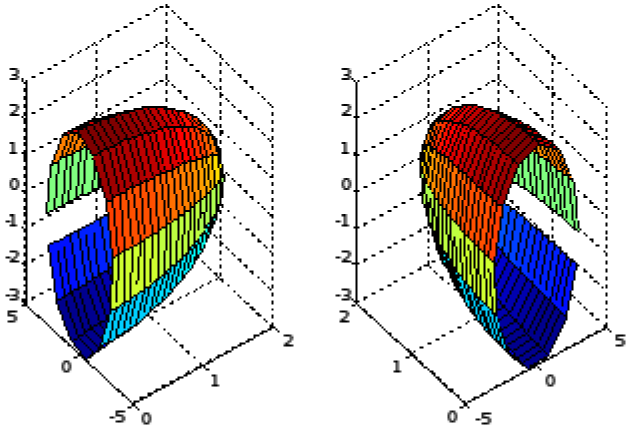
$x=t$, $y = 3\ \text{Cos}(u)\ (1-t^{2}/4)^{1/2}$, $z = 3\ \text{Sen}(u)\ (1-t^{2}/4)^{1/2}$

The equation of the figure of revolution around the OY axis is written as $x^{2} + z^{2} = 4(1-y^{2}/4)$, and is parametrised as follows:

$x = 3\ \text{Cos}(u)\ (1-t^{2}/9)^{1/2}$, $y=t$, $z = 3\ \text{Sen}(u)\ (1-t^{2}/9)^{1/2}$

```
" t=(0:.1:2);

"u=(0:.5:2*pi);

" x=ones(size(u))'*t;

» y=cos(u)'*3*(1-t.^2/4).^(1/2);

» z=sin(u)'*3*(1-t.^2/4).^(1/2);

" subplot(1,2,1)
```

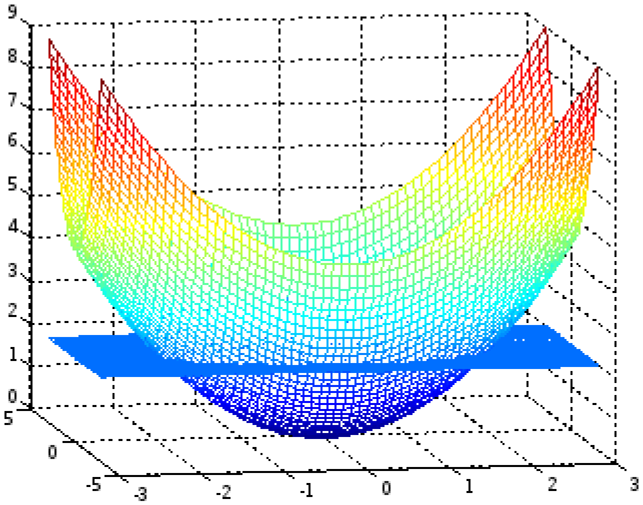
```
"surf(x,y,z)
" subplot(1,2,2)
» x=cos(u)'*3*(1-t.^2/4).^(1/2);
" y=ones(size(u))'*t;
» z=sin(u)'*3*(1-t.^2/4).^(1/2);
"surf(x,y,z)
```



E xercise 6. Represent the intersection of the paraboloid $x^2 + y^2 = 2z$ with the plane z=2.

```
" [x,y]=meshgrid(-3:.1:3);
"z=(1/2)*(x.^2+y.^2);
"mesh(x,y,z)
" hold on;
" z=2*ones(size(z));
"mesh(x,y,z)
```

```
"view(-10,10)
```

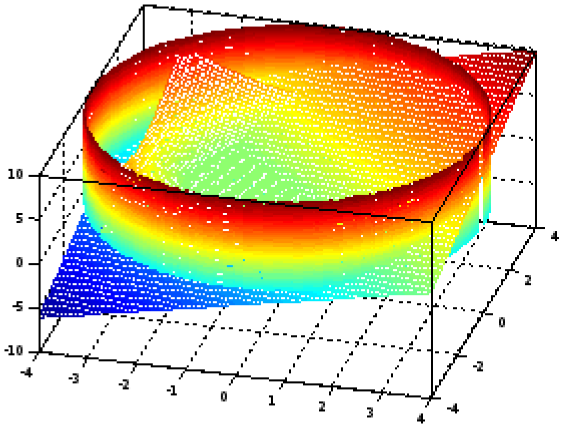


R epresent the volume in the first octant between the plane OXY, the plane z=x+y+2 and the cylinder x ^{2} + y ^{2} = 16.

We make a graphical representation of the enclosure, with the plane in Cartesian and the cylinder parameterised.

```
"t=(0:.1:2*pi);

" u=(0:.1:10);

"x=4*cos(t)'*ones(size(u));

"y=4*sin(t)'*ones(size(u));

" z=ones(size(t))'*u;

"mesh(x,y,z)

" hold on;

" [x,y]=meshgrid(-4:.1:4);

"z=x+y+2;
```

```
"mesh(x,y,z)

" set(gca,'Box','on');

" view(15,45)
```

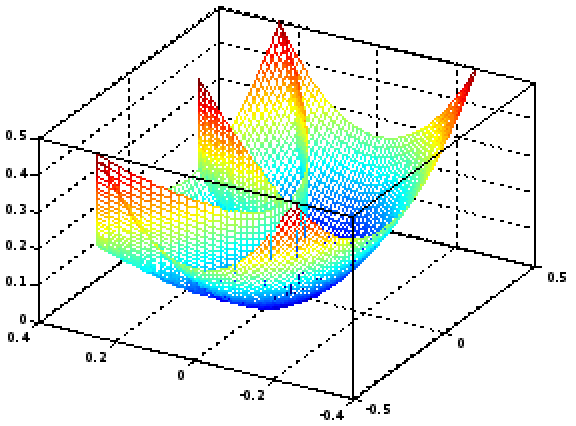


R epresent the volume bounded by the paraboloid $x^2 + 4y^2 = z$, and laterally by the cylinders of equations $y^2 = x$ and $x = {}^2 y$.

```
» [x,y]=meshgrid(-1/2:.02:1/2,-1/4:.01:1/4);

"z=x.^2+4*y.^2;

"mesh(x,y,z)

" hold on;

"y=x.^2;

"mesh(x,y,z)

" hold on;

"x=y.^2;

"mesh(x,y,z)
```

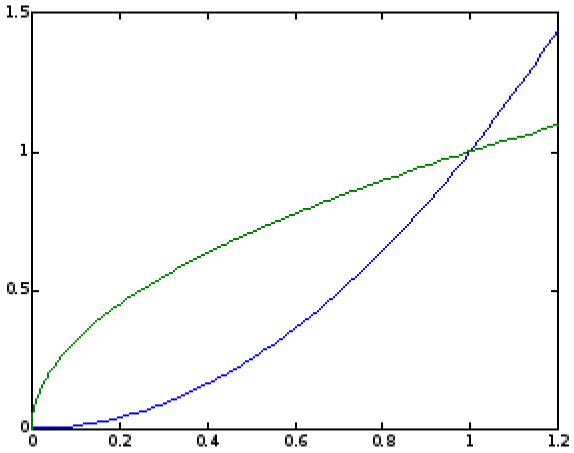
```
" set(gca,'Box','on')
```

```
"view(-60,40)
```

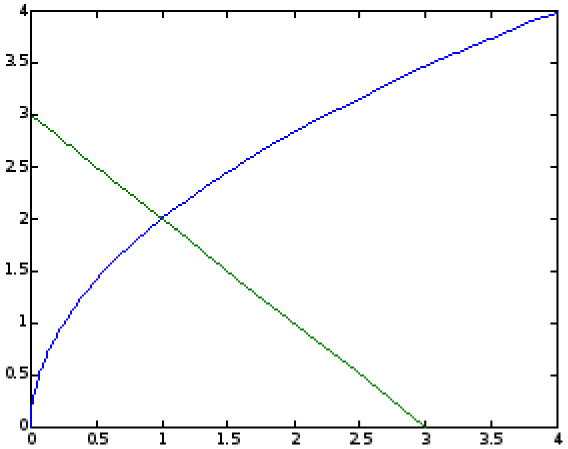


R epresent on the same axes the parabolas $y^2 = x$ and $x^2 = y$. Also represent on the same axes the parabola $y^2 = 4x$ and the line x+y=3.

```
>> fplot('[x^2,sqrt(x)]',[0,1.2])
```



```
» fplot('[(4*x)^(1 /2),3-x]',[0,4,0,4])
```



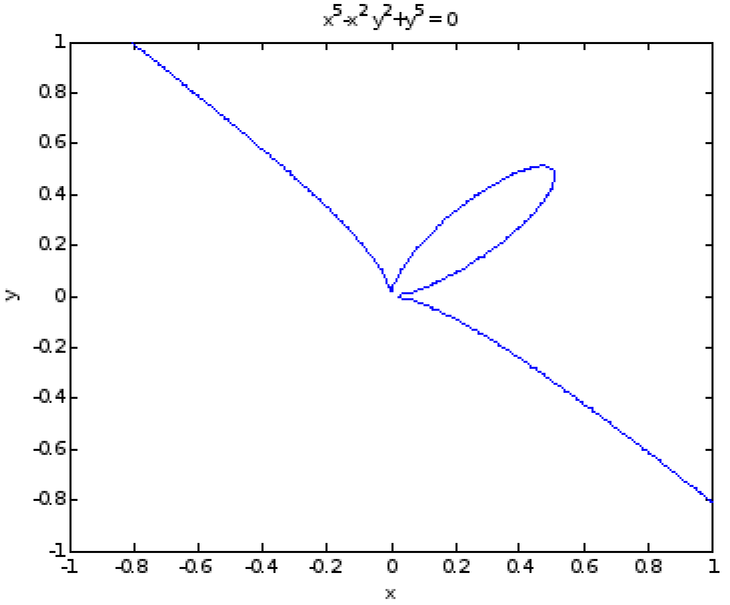
E xercise 10. Represent the curvature in implicit coordinates of equations:

$x^5 - x^2 y^2 + y^5 = 0$

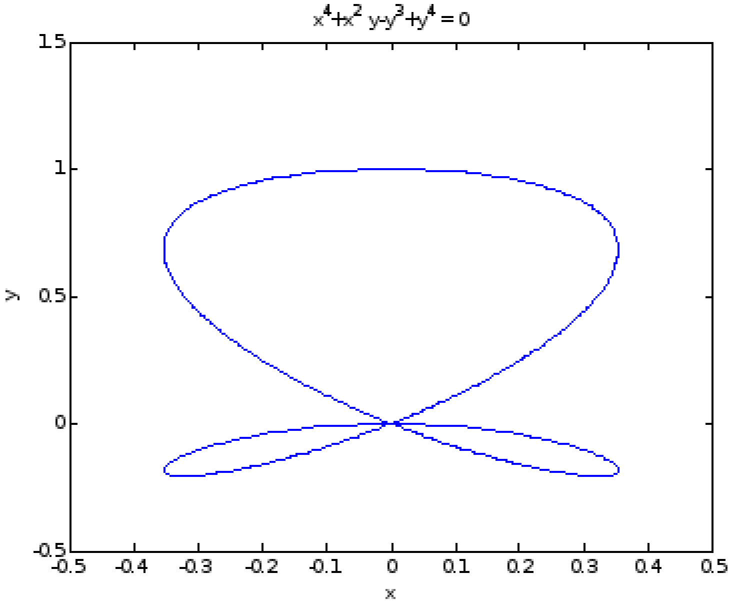
$x^4 + x^2 y - y^5 + y^4 = 0$

____

```
>> ezplot('x^5-x^2*y ^2+y^5', [-1,1,-1,1])
```



```
>> ezplot('x^4+x^2*y -y^3+y^4', [-1/2,1/2,-1/2,3/2])
```

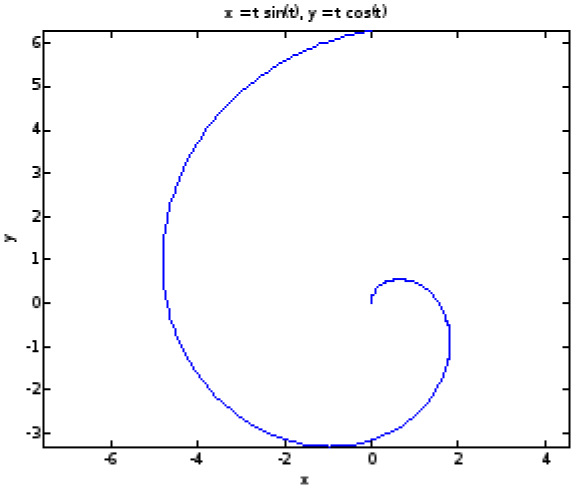


E xercise 11. Represent the curve in parametric coordinates:

x(t) = tSen(t)

y(t) = tCos(t)

```
>> ezplot('t*sin(t)' ,'t*cos(t)')
```



E xercise 12. Represent the curve in polar coordinates:

ρ=1− cos(θ)

```
>> ezpolar('1-cos(t)' )
```

image

E xercise 13. Represent the warped curve of paarametric equations:

x=cos(t) y=sen(t) z=t

```
>> ezplot3('cos(t)','sin (t)','t',[0,6*pi])
```

image

Chapter 4.

ALGEBRAIC EXPRESSIONS, POLYNOMIALS, EQUATIONS AND SYSTEMS

4.1 EXPANSION, SIMPLIFICATION AND FACTORISATION OF ALGEBRAIC EXPRESSIONS

M ATLAB incorporates a wide range of commands that allow you to work with algebraic expressions. It is possible to simplify an expression as much as possible, expand it or factor it appropriately. The following table presents the most common commands for working with algebraic expressions.

expand(expr)

Expands an algebraic expression as much as possible, fully realising products and powers, until the result is presented as a sum of terms. Applies multiple angle rules for trigonometric expressions and formally applies the properties of logarithmic and exponential functions. Also decomposes algebraic fractions of polynomial numerator into sums of fractions.

```
>> syms x y z t a b
>> pretty(expand((x+2)*(x+3))))
2
x + 5 x + 6
>> pretty(expand((x+2)/(x+3))))
x 2
—-—+—-—
x + 3 x + 3
>> pretty(expand(cos(x+y)))
cos(x) cos(y) - sin(x) sin(y)
```

factor(expr)

Writes an expanded algebraic expression as a product of factors (inverse of expand). The factorisation defaults to the body defined by the coefficients of the expression. For algebraic fractions, *factor numerator and denominator and simplify common factors.*

```
>> syms x y
   >> pretty(factor(6*x^2+18*x-24))
   6 (x + 4) (x - 1)
   >> pretty(factor((x^3-y^3)/(x^4-y^4))))
   2 2
   x + x y + y
                              ________________
2 2
   (x + y) (x + y )
   >> pretty(factor(x^3+y^3))
   2 2
   (x + y) (x - x y + y )
```

simplify(expr)

Simplifies an algebraic expression as much as possible by performing final sums and also performs addition of algebraic fractions.

```
>> syms x y a b c
>> simplify(sin(x)^2 + cos(x)^2)*2
ans =
2
>> simplify(log(log(exp(a+log(exp(c)))))
ans =
log(exp(a + c))
```

simple(expr)

Simplifies the algebraic expression as much as possible by showing the rules that are used.

```
>> syms a positive;
f = (1/a^3 + 6/a^2 + 12/a + 8)^(1/3);
>> simplify(f)
ans =
(8*a^3 + 12*a^2 + 6*a + 1)^(1/3)/a
```

```
   >> simple(f)
   simplify:
   (2*a + 1)/a

r adsimp:
   (12/a + 6/a^2 + 1/a^3 + 8)^(1/3)

s implify(100):
   1/a + 2

c ombine(sincos):
   (12/a + 6/a^2 + 1/a^3 + 8)^(1/3)

c ombine(sinhcosh):
   (12/a + 6/a^2 + 1/a^3 + 8)^(1/3)

c ombine(ln):
   (12/a + 6/a^2 + 1/a^3 + 8)^(1/3)

f actor:
   (12/a + 6/a^2 + 1/a^3 + 8)^(1/3)

e xpand:
   (12/a + 6/a^2 + 1/a^3 + 8)^(1/3)
```

```
c ombine:
    (12/a + 6/a^2 + 1/a^3 + 8)^(1/3)

r ewrite(exp):
    (12/a + 6/a^2 + 1/a^3 + 8)^(1/3)

r ewrite(sincos):
    (12/a + 6/a^2 + 1/a^3 + 8)^(1/3)

r ewrite(sinhcosh):
    (12/a + 6/a^2 + 1/a^3 + 8)^(1/3)

r ewrite(tan):
    (12/a + 6/a^2 + 1/a^3 + 8)^(1/3)

m wcos2sin:
    (12/a + 6/a^2 + 1/a^3 + 8)^(1/3)

c ollect(a):
    (12/a + 6/a^2 + 1/a^3 + 8)^(1/3)
    ans =
    1/a + 2
    >> g=simple(f)
    g =
    1/a + 2
```

collect(expr)

Presents the result of the expression by grouping terms in powers of their variables.

```
>> syms x;
f = x*(x*(x*(x - 6) + 11) - 6;
>> collect(f)
ans =
x^3 - 6*x^2 + 11*x - 6
>> f = (1+x)*t + x*t;
>> collect(f)
ans =
(2*t)*x + t
```

horner(expr)

Factor the expression in its Horner form

```
>> syms x;
f = x^3 - 6*x^2 + 11*x - 6;
>> horner(f)
ans =
x*(x*(x - 6) + 11) - 6
```

4.2 POLYNOMY

M ATLAB implements specific commands for working with polynomials in tasks such as finding their roots, evaluation, differentiation and interpolation. The following table presents the syntax and examples of the most important commands involving polynomials.

poly2sym(vector)

Converts the vector into a symbolic polynomial whose coefficients are the components of the vector (from highest to lowest power).

```
>> poly2sym([3 5 0 8 9])
ans =
3*x^4 + 5*x^3 + 8*x + 9
```

poly2sym(vector,'v')

Converts the vector to a symbolic polynomial in the variable v whose coefficients are the components of the vector

```
>> poly2sym([3 5 0 8 9],'z')
ans =
3*z^4 + 5*z^3 + 8*z + 9
```

sym2poli(polynomial)

Converts the polynomial to a vector whose components are its coefficients

```
>> syms x
>> sym2poly(x^5-3*x^4+2*x^2-7*x+12)
ans =
1 -3 0 2 -7 12
```

q=conv(u,v)

Give the coefficients of the polynomial product of the two polynomials whose coefficients are given by the vectors u and v.

```
>> u=[3 -1 4 2];v=[2 1 4 4 6 8 3];
>> p=conv(u,v)
```

```
p =
6 1 19 22 36 33 41 28 6
>> poly2sym(p)
ans =
6*x^8 + x^7 + 19*x^6 + 22*x^5 + 36*x^4 + 33*x^3 + 41*x^2 + 28*x + 6
```

[q,r] = deconv(v,u)

Give the polynomial quotient and remainder of the division between the polynomials u and v. We have v = conv(u,q)+r

```
>> [q,r]=deconv(v,u)
q =
0.6667 0.5556 0.6296
r =
0 0 0 3.0741 4.3704 1.7407
>> poly2sym(q)
ans =
(2*x^2)/3 + (5*x)/9 + 17/27
>> poly2sym(r)
ans =
(83*x^2)/27 + (118*x)/27 + 47/27
```

p = poly(r)

Gives the coefficients of the polynomial p whose roots are the vector r

```
>> p=poly(u)
p =
1 -8 17 2 -24
>> poly2sym(p)
ans =
x^4 - 8*x^3 + 17*x^2 + 2*x - 24
```

k = polyder(p)

Gives the coefficients of the polynomial k derived from the polynomial p

k = polyder(a,b)

[q,d] = polyder(b,a)

Give the coefficients of the derivative of the product of a and b.

Give the numerator q and denominator d of the derivative of a/b.

```
 >> polyder([1 -8 17 2 -24])
 ans =
 4 -24 34 2
 >> poly2sym([1 -8 17 2 -24])
 ans =
 x^4 - 8*x^3 + 17*x^2 + 2*x - 24
 >> poly2sym(polyder([1 -8 17 2 -24]))
 ans =
 4*x^3 - 24*x^2 + 34*x + 2
 >> u=[3 -1 4 2];v=[2 1 4 4 6 8 3];
 >> k=polyder(u,v)
 k =
 48 7 114 110 144 99 82 28
 >> poly2sym(k)
 ans =
 48*x^7 + 7*x^6 + 114*x^5 + 110*x^4 + 144*x^3 + 99*x^2 + 82*x + 28
 >> [q,d]=polyder(u,v)
 q =
 -12 3 -30 -10 8 -29 -30 -4



d =
 4 4 17 32 60 76 106 120 100 48 9
```

p = polyfit(x,y,n)	*Polynomial of degree n fitting the point cloud (x,y)*
[p,S] = polyfit(x,y,n)	*Polynomial of degree n fitting the points (x,y) with variance S*
[p,S,u] = polyfit(x,y,n)	*Polynomial of degree n fitting the (x,y) points with variance S and mean u for the error of fit*

```
 >> u=[3 -1 4 2];v=[2 1 4 6];
 >> p=poly2sym(polyfit(u,v,3))
 p =
 (53*x^3)/60 - (99*x^2)/20 + (119*x)/30 + 54/5
 >> [p,S,u]=polyfit(u,v,3)
 p =
 8.9050 1.6333 -11.3053 6.0000



S =
 A: [4x4 double].
 df: 0
 normr: 1.2686e-014



u =
 2.0000
 2.1602
```

y = polyval(p,x) *Evaluate the polynomial p in x*

y = polyval(p,x,[ ],u) *Evaluates the polynomial p in x with mean u for the error*

[y,delta] = polyval(p,x,S) *Evaluate the polynomial p in x with variance S for the error y with an error y ±delta in the result.*

[y,delta] = polyval(p,x,S,u) *Evaluate the polynomial p in x with mean u and variance S for the error y with an error y ±delta in the result.*

```
 >> p=[2 0 -1 7 9]
 p =
 2 0 -1 7 9
 >> poly2sym(p)
 ans =
 2*x^4 - x^2 + 7*x + 9
```

```
>> polyval(p,10)
ans =
19979
```

Y = polyvalm(p,X)

Evaluates the polynomial p in the matrix variable X

```
>> X=[1 2 3;4 5 6;7 8 9]
X =
1 2 3
4 5 6
7 8 9
>> p=[2 0 -1 7 9]
p =
2 0 -1 7 9
>> A=polyval(p,X)
A =
17 51 183
533 1269 2607
4811 8193 13113
```

[r,p,k] = residue(b,a)

Gives residues, poles and direct terms of the rational expansion of b/a

[b ,a] = residue(r,p ,k)

Converts rational expansion to polynomials of coefficients b and a

```
>> u=[3 -1 4 2];v=[2 1 4 4 6 8 3];
>> [r,p,k]=residue(v,u)
r =
0.4751 - 0.6032i
0.4751 + 0.6032i
0.0745
```

```
p =
  0.3705 + 1.2240i
```

```
     0.3705 - 1.2240i
     -0.4076



k =
     0.6667 0.5556 0.6296
     >> [v,u]=residue(r,p,k)
     v =
     0.6667 0.3333 1.3333 2.0000 2.6667 1.0000
     u =
     1.0000 -0.3333 1.3333 0.6667
```

r = roots(c)

Gives the column vector r of roots of the polynomial with coefficients c

```
     >> v=[0.6667 0.3333 1.3333 2.0000 2.6667 1.0000];
     >> r=roots(v)
     r =
     0.6662 + 1.4813i
     0.6662 - 1.4813i
     -0.6662 + 0.8326i
     -0.6662 - 0.8326i
     -0.5000
```

4.3 POLYNOMIAL INTERPOLATION

M ATLAB implements specific commands, both algebraic and graphical, for working with polynomial interpolation. The following table presents the syntax and examples of the most important commands for polynomial interpolation.

Yi = interp1(X,Y,Xi)	*Gives the vector Yi such that (Xi,Yi) is the total set of points found by one-dimensional interpolation of the given set of points (X,Y).*
Yi = interp1(Y,Xi)	*Assume that X=1:N, where N is the length of Y*
Yi = interp1(X,Y,Xi,method)	*Performs interpolation using the given method, which can be nearest (nearest neighbour), linear (linear), cubic (Hermite cubic), v5cubic (MATLAB 5 cubic), spline and pchip (Hermite cubic).*
Y i = interp1(X,Y,Xi ,method,ext)	*Additionally, it performs extrapolation for the value of x=ext external to the range of variation of x used for interpolation.*

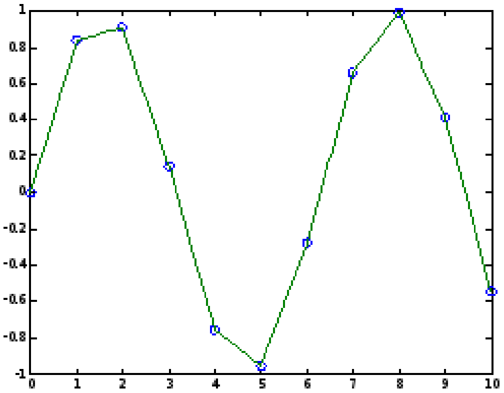
In the following example, 20 points (x,y) are interpolated according to the function *y=sen(x)* for equally spaced values of *x* between 0 and 10.

```
>> x = 0:10; y = sin(x); xi = 0:.5:10;yi = interp1(x,y,xi); points=[xi',yi'].
points = points = points = points = points = points = points = points = points = points = points = points
0 0
```

```
0.5000 0.4207
1.0000 0.8415
1.5000 0.8754
2.0000 0.9093
2.5000 0.5252
3.0000 0.1411
3.5000 -0.3078
4.0000 -0.7568
4.5000 -0.8579
5.0000 -0.9589
5.5000 -0.6192
6.0000 -0.2794
6.5000 0.1888
7.0000 0.6570
7.5000 0.8232
8.0000 0.9894
8.5000 0.7007
9.0000 0.4121
9.5000 -0.0660
10.0000 -0.5440
```

We can represent the points as follows:

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



Zi = interp2(X,Y,Z,Xi,Yi) *Gives the vector Zi such that (Xi,Yi, Zi) is the total set of points found by two-dimensional interpolation of the given set of points (X,Y,Z).*

Z i = interp2(Z,Xi,Yi)

Assume that X=1:n and Y=1:m and (n,m) = size of Z

Zi = interp2(Z,n)

Two-dimensional recursive interpolation n times

Zi = interp2(X,Y,Z,Xi,Yi, *method*)

Interpolation using nearest neighbour, linear, cubic (cubic Hermite) and spline methods

In the following example we consider a set of years, lengths of service and salaries and try to find by interpolation the salary earned in 1975 by an employee with 15 years of service.

```
>> years = 1950:10:1990;
services = 10:10:30;
wages = [150,697 199,592 187,625
179.323 195.072 250.287
203.212 179.092 322.767
226.505 153.706 426.730
249.633 120.281 598.243];
w = interp2(services,years,wages,15,1975)
w =
190.6288
```

Vi = interp3(X,Y,Z,V,Xi,Yi,Zi)

Gives the vector Vi such that (Xi,Yi, Zi) is the total set of points found by three-dimensional interpolation of the set of points resulting from applying the three-dimensional function V to the points (X,Y,Z).

V i = interp3(V,Xi,Yi ,Zi)

Assume that X=1:n, Y=1:m, Z=1:p and (n,m,p) = size of V

Vi = interp3(V,n)

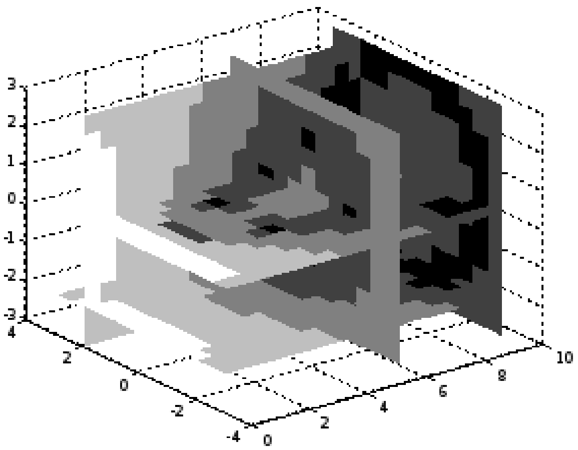
Three-dimensional recursive interpolation n times

Vi = interp3(...,method)

Interpolation using the specified methods

In the following example, the volumetric point matrix *w=f(x,y,z)* of three-dimensional interpolation between the points (*x,y,z,t*) given by the values taken by the predefined MATLAB function ' *flower* ' for values of *x* between 0.1 and 10 half a point apart and for values of *y* and *z* between -3 and 3 half a point apart is calculated and plotted.

```
>> [x,y,z,v] = flow(10);
[xi,yi,zi] = meshgrid(.1:.5:10,-3:.5:3,-3:.5:3);
vi = interp3(x,y,z,v,xi,yi,zi);
slice(xi,yi,zi,vi,[6 9.5],2,[-2 .2]), shading flat
```



Y = interpft(X,n)

Unidimensional interpolation using the FFT method. It gives the vector Y containing the values of the periodic function X sampled at n equally spaced points. The original

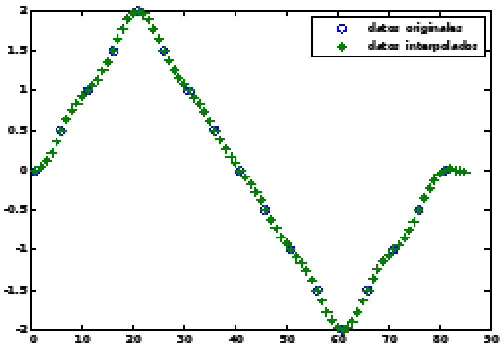
y = interpft(x,n,dim)

vector X is transformed to the Fourier frequency domain using the fast Fourier transform FFT

Operates along the specified dimension

An example is presented where the original points are compared with the points interpolated through the FFT method.

```
>> y = [0:.5:2 1.5:-.5:-2 -1.5:.5:0]; % Equally spaced dots
factor = 5; % Interpolation with a factor of 5
m = length(y)*factor;
x = 1:factor:m;
xi = 1:m;
yi = interpft(y,m);
plot(x,y,'o',xi,yi,'*')
legend('original data','interpolated data')
```



Vi = interpn(X,Y,Z,...V,Xi,Yi,Zi...)

Gives the vector Vi such that (Xi,Yi, Zi,...) is the total set of points found by n-dimensional interpolation of the set of points resulting from applying the three-dimensional function V to the points (X,Y,Z,...).

V i = interpn(V,Xi,Yi ,Zi)

Vi = interpn(V,n)

Vi = interpn(...,method)

Assume X=1:n, Y=1:m, Z=1:p,... and (n,m,p,...) = size of V

Recursive n-dimensional interpolation n times

Interpolation using the specified methods

Yi = pchip(X,Y,Xi)

Gives the vector Yi such that (Xi,Yi) is the total set of points found by piecewise Hermite cubic polynomial interpolation (PCHIP) of the given set of points (X,Y).

pp = pchip(X,Y)

Performs interpolation at intermediate points

Yi = spline(X,Y,Xi)

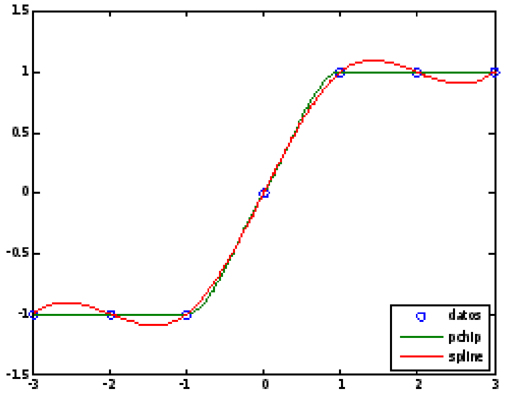
Gives the vector Yi such that (Xi,Yi) is the total set of points found by cubic spline interpolation of the given set of points (X,Y).

pp = spline(X,Y)

Performs interpolation at intermediate points

An example is presented where the original points are compared with the points interpolated via *pchip* and *spline* method.

```
>> x = -3:3;
y = [-1 -1 -1 0 1 1 1];
t = -3:.01:3;
plot(x,y,'o',t,[pchip(x,y,t); spline(x,y,t)])
legend('data','pchip','spline',4)
```



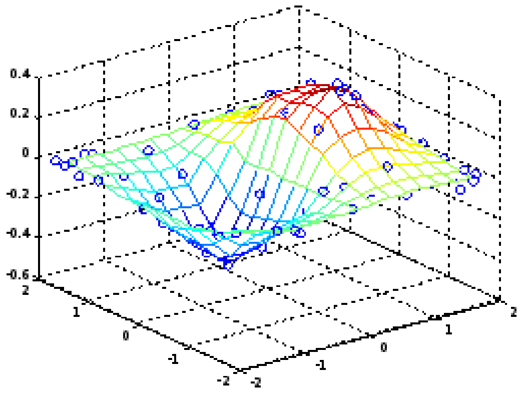
Zi = griddata(X,Y,Z,Xi,Yi) *It fits a surface of the form Z=f(X,Y) in the space of vectors (X,Y,Z). The vector Zi determines the interpolation points (Xi,Yi,Zi) between the given points (X,Y,Z). An inverse distance method is used for interpolating*

[X i,Yi,Zi] = griddata (X,Y,Z,Xi,Yi) *In addition to Xi, gives the row and column vectors Yi and Zi*

Interpolation using the specified methods

[...] = griddata(...,method) The following is an example of an interpolation adjustment of a surface.

```
x = rand(100,1)*4-2; y = rand(100,1)*4-2;
z = x.*exp(-x.^2-y.^2);
ti = -2:.25:2;
[xi,yi] = meshgrid(ti,ti);
zi = griddata(x,y,z,xi,yi);
mesh(xi,yi,zi), hold on, plot3(x,y,z,'o'),
hold off
```

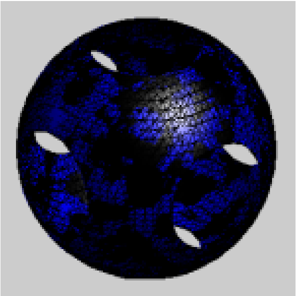


W = griddata3(X,Y,Z,V,Xi,Yi,Zi) *It fits a hypersurface of the form W=f(X,Y,Z) in the space of vectors (X,Y,Z,V). The vector W determines the interpolation points (Xi,Yi,Zi,Vi) between the given points (X,Y,Z,V). An inverse distance method is used for interpolating*

W = griddata3(...,'method') *Interpolation using the specified methods*

The following is an example of an interpolation adjustment of a hypersurface

```
>> x = 2*rand(5000,1)-1; y = 2*rand(5000,1)-1; z = 2*rand(5000,1)-1;
v = x.^2 + y.^2 + z.^2;
d = -0.8:0.05:0.8;
[xi,yi,zi] = meshgrid(d,d,d);
w = griddata3(x,y,z,v,xi,yi,zi);
p = patch(isosurface(xi,yi,zi,w,0.8));
isonormals(xi,yi,zi,w,p);
set(p,'FaceColor','blue','EdgeColor','none');
view(3), axis equal, axis off, camlight, lighting phong
```



4.4 SOLVING EQUATIONS AND SYSTEMS

M ATLAB provides multiple commands for solving equations and systems. The following sections present the syntax and characteristics of these methods.

image image
image

4.4.1 General methods

image

T he following are the most common commands in MATLAB for solving equations and systems

solve('equation','x')	Solve the equation in the variable x The equation p*sen(x)=r is solved. >> solve('p*sin(x) = r') ans = asin(r/p) pi - asin(r/p)
syms x ; solve(ecu(x),x)	Solve the equation equation(x)=0 in the variable x The equation p*cos(x)=r is solved. >> syms x r; solve(p*cos(x)-r, x) ans = acos((8192*r)/1433639) -acos((8192*r)/1433639)

solve('ec1,ec2,...,ecn', 'x1, x2,...,xn')	Solve n simultaneous equations eq1,...,eqn (system in the variables x1,...,xn) >> [x,y] = solve('x^2 + x*y + y = 3','x^2 - 4*x + 3 = 0') x = 1 3 ________ image y = 1 -3/2

syms x1 x2 x2...xn;
solve(ec1,ec2,...,ecn, x1, x2,...,xn)
	Solve n simultaneous equations eq1,...,eqn (system in the variables x1,...,xn)
	>> syms x y; [x,y] = solve(x^2 + x*y + y - 3, x^2 - 4*x + 3)
	x =
	1
	3
	________
	image

	y = 1 -3/2
X = linsolve(A,B)	Solve A*X=B for a square matrix A, where B and X are matrices We solve the system: 2 x + y + z + t = 1 x + 2 y + z + t = 2 x + y +2 z + t = 3 x + y + z +2 t = 4 >> A=[2,1,1,1;1,2,1,1;1,1,2,1;1,1,1,2];B=[1,2,3,4]'; linsolve(A,B) ans = -1 0 1 2

x = lscov(A,B)
x = lscov(A,B,V)
	Solve A*x=B in least squares sense

Solve A*x=B in the least squares sense with error covariance matrix proportional to V. That is, give the vector x such that Ax=b+e with e→N(0,V).

Gives x (vector) that minimises (A*x-B)'*inv(V)*(A*x- B).

```
>> A=[2,1,1,1;1,2,1,1;1,1,2,1;1,1,1,2];B=[1,2,3,4]';

lscov(A,B)

ans =

-1

0

1

2
```

Solve the system A*X=B

```
>> A=[2,1,1,1;1,2,1,1;1,1,2,1;1,1,1,2];B=[1,2,3,4]';

A

ans =
```

X = A

```
-1.0000

-0.0000

1.0000

2.0000
```

X = A/B	Solve the system X*A=B
roots(A)	Give the roots of the polynomial whose coefficients construct the vector A (from highest to lowest order). As an example we find the roots of the polynomial x ^{4} +2x +3x +3x+4 ^{3} >> roots([1 2 3 4]) ans = -1.6506 -0.1747 + 1.5469i -0.1747 - 1.5469i
poly(V)	Gives the polynomial whose roots are the vector V >> poly([1 2 3 4]) ans = 1 -10 35 -50 24
x=fzero(function,x0)	Find a zero of the function near x0 >> X = fzero('sin(x)-1', pi/2) X = 1.5708

[x, feval]=fzero(fun,x0)	It also gives the value of the objective function at x >> [X f] = fzero('sin(x)-1', pi/2) X = 1.5708 f = 0

4.4.2 BICONJUGATE GRADIENT METHOD

The following are the commands provided by MATLAB for solving equations and systems using the biconjugate gradient method.

x = bicg(A,b)

Try to solve the system Ax=b by the method of biconjugate gradients.

```
>> A=[pi 2*pi 3*pi -pi; 1 0 -1 2; exp(1) exp(2) exp(3) exp(4); i 2i 3i -i];
>> B=[1 2 3 4]';
>> bicg(A,B)
bicg stopped at iteration 4 without converging to the desired tolerance 1e-006
because the maximum number of iterations was reached. The iterate returned (number 0) has relative residual 1
ans =
0
0
0
0
```

Solve Ax=b specifying the tolerance

Solve Ax=b specifying tolerance and maximum number of iterations

```
>> bicg(A,B, 1e-009,100)
ans =
1.0e+016 *
4.1449 - 0.7033i
-7.1997 + 1.2216i
3.2729 - 0.5553i
```

bicg(A,b,tol)

bicg(A,b,tol,maxit)

```
-0.4360 + 0.0740i
```

*Solve the system inv(M)*A*x = inv(M)*b*

*Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2*

*Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2 and initial value x0*

Solve the system and f indicates the result (0=convergence, 1=non-convergence, 2=conditional convergence, 3=stagnation and 4=very extreme numbers).

```
>> [x,f]=bicg(A,B, 1e-009,100)
x =
1.0e+016 *
4.1449 - 0.7033i
-7.1997 + 1.2216i
3.2729 - 0.5553i
-0.4360 + 0.0740i
f =
3
```

bicg(A,b,tol,maxit,M)

bicg(A,b,tol,maxit,M1,M2)

bicg(A,b,tol,maxit,M1,M2,x0)

[x,f] = bicg(A,b,...)

x = bicgstab(A,b)

Try to solve the system Ax=b by the stabilised biconjugate gradient method.

```
>> bicgstab(A,B)
bicgstab stopped at iteration 4 without converging to the desired tolerance 1e-006
because the maximum number of iterations was reached.
The iterate returned (number 4) has relative residual 0.88
ans =
1.0e+011 *
0.6696 - 0.4857i
-1.1631 + 0.8437i
0.5287 - 0.3835i
-0.0704 + 0.0511i
```

Solve Ax=b specifying the tolerance

Solve Ax=b specifying tolerance and maximum number of iterations

```
>> bicg(A,B, 1e-009,100)
ans =
1.0e+016 *
4.1449 - 0.7033i
-7.1997 + 1.2216i
3.2729 - 0.5553i
```

bicgstab(A,b,tol)

bicgstab(A,b,tol,maxit)

```
-0.4360 + 0.0740i
```

*Solve the system inv(M)*A*x = inv(M)*b*

*Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2*

*Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2 and initial value x0*

Solve the system and f indicates the result (0=convergence, 1=non-convergence, 2=conditional convergence, 3=stagnation and 4=very extreme numbers).

*It also returns the relative residual norm(b-A*x)/norm(b).*

Returns also the number of iterations

```
>> [x,f,r,i]=bicg(A,B, 1e-006,100)
x =
1.0e+016 *
4.1449 - 0.7033i
-7.1997 + 1.2216i
3.2729 - 0.5553i
-0.4360 + 0.0740i
f =
3
r =
26.0415
i =
18
```

```
    bicgstab(A,b,tol,maxit,M)

    bicgstab(A,b,tol,maxit,M1,M2)

    bicgstab(A,b,tol,maxit,M1,M2,x0)

    [x,f] = bicgstab(A,b,...)



[x ,f,relres] = bicgstab (A,b,...)

    [x,f,relres,iter] = bicgstab(A,b,...)
```

image image
image

4.4.3 Conjugate gradient method

image

T he following are the commands provided by MATLAB for solving equations and systems using the conjugate gradient method.

x = pcg(A,b)	Try to solve the system Ax=b by the preconditioned conjugate gradient method.
pcg(A,b,tol)	Solve Ax=b specifying the tolerance
pcg(A,b,tol,maxit)	Solve Ax=b specifying tolerance and maximum number of iterations
pcg(A,b,tol,maxit,M)	Solve the system inv(M)*A*x = inv(M)*b
pcg(A,b,tol,maxit,M1,M2)	Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2
pcg(A,b,tol,maxit,M1,M2,x0)	Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2 and initial value x0
[x,f] = pcg(A,b,...) ———— image	Solve the system and f indicates the result (0=convergence, 1=non-convergence, 2=conditional convergence, 3=stagnation and 4=very extreme numbers).
[x ,f,relres] = pcg(A ,b,...)	It also returns the relative residual norm(b-A*x)/norm(b).
[x,f,relres,iter] = pcg(A,b,...)	Returns also the number of iterations

```
>> A=[pi 2*pi 3*pi -pi; 1 0 -1 2; exp(1)
exp(2) exp(3) exp(4); i 2i 3i -i];

>> B=[1 2 3 4]';

>> [x,f,r,i]=pcg(A,B, 1e-006,1000)

x =

0

0

0

0

f =

4

r =

1

i =

0
```

x = lsqr(A,b)	Try to solve the system Ax=b by the conjugate gradient method in normal equations.
lsqr(A,b,tol)	Solve Ax=b specifying the tolerance
lsqr(A,b,tol,maxit)	Solve Ax=b specifying tolerance and maximum number of iterations
lsqr(A,b,tol,maxit,M)	
lsqr(A,b,tol,maxit,M1,M2)	

lsqr(A,b,tol,maxit,M1,M2,x0)

[x,f] = lsqr(A,b,...)

---

image

[x ,f,relres] = lsqr (A,b,...)

[x,f,relres,iter] = lsqr(A,b,...)

Solve the system inv(M)*A*x = inv(M)*b

Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2

Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2 and initial value x0

Solve the system and f indicates the result (0=convergence, 1=non-convergence, 2=conditional convergence, 3=stagnation and 4=very extreme numbers).

It also returns the relative residual norm(b-A*x)/norm(b).

Returns also the number of iterations

```
>> A=[pi 2*pi 3*pi -pi; 1 0 -1 2; exp(1) exp(2) exp(3) exp(4); i 2i 3i -i];

>> B=[1 2 3 4]';

>> [x,f,r,i]=lsqr(A,B, 1e-006,1000)

x =

1.1869 - 0.0910i

0.4295 - 0.0705i

-0.5402 - 0.0362i

0.1364 + 0.0274i
```

---

image

f =

0

---

image

r =

0.6981

---

image

i =

3

4.4.4 RESIDUAL METHOD

T he following are the commands provided by MATLAB for solving equations and systems by the residual method.

x = qmr(A,b)	*Try to solve the system Ax=b by the quasiminimal residual method.*
qmr(A,b,tol)	*Solve Ax=b specifying the tolerance*
qmr(A,b,tol,maxit)	*Solve Ax=b specifying tolerance and maximum number of iterations*
qmr(A,b,tol,maxit,M)	*Solve the system inv(M)*A*x = inv(M)*b*
qmr(A,b,tol,maxit,M1,M2)	*Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2*
qmr(A,b,tol,maxit,M1,M2,x0)	*Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2 and initial value x0*
[x,f] = qmr(A,b,...)	*Solve the system and f indicates the result (0=convergence, 1=non-convergence, 2=conditional convergence, 3=stagnation and 4=very extreme numbers).*
[x ,f,relres] = qmr(A ,b,...)	*Gives also the relative residual norm(b-A*x)/norm(b)*
[x,f,relres,iter] = qmr(A,b,...)	

	Returns also the number of iterations >> A=[pi 2*pi 3*pi -pi; 1 0 -1 2; exp(1) exp(2) exp(3) exp(4); i 2i 3i -i]; >> B=[1 2 3 4]'; >> [x,f,r,i]=qmr(A,B, 1e-006,1000) x = 1.0e+016 * 0.4810 - 4.0071i -0.8356 + 6.9603i 0.3798 - 3.1640i -0.0506 + 0.4215i f = 3 r = 19.5999 i = 11

x = gmres(A,b)
gmres(A,b,tol)
gmres(A,b,tol,maxit)
gmres(A,b,tol,maxit,M)
gmres(A,b,tol,maxit,M1,M2)
gmres(A,b,tol,maxit,M1,M2,x0)
[x,f] = gmres(A,b,...)
 	
[x ,f,relres] = gmres (A,b,...)
[x,f,relres,iter] = gmres(A,b,...)
	Try to solve the system Ax=b by the generalised minimum residual method.
	Solve Ax=b specifying the tolerance
	Solve Ax=b specifying tolerance and maximum number of iterations
	Solve the system inv(M)*A*x = inv(M)*b
	Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2
	Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2 and

initial value x0

Solve the system and f indicates the result (0=convergence, 1=non-convergence, 2=conditional convergence, 3=stagnation and 4=very extreme numbers).

*It also returns the relative residual norm(b-A*x)/norm(b).*

Returns also the number of iterations

```
>> A=[pi 2*pi 3*pi -pi; 1 0 -1 2; exp(1) exp(2) exp(3) exp(4); i 2i 3i -i];
>> B=[1 2 3 4]';
>> [x,f,r,i]=gmres(A,B)
x =
1.5504 + 0.0085i
-0.2019 - 0.2433i
-0.2532 + 0.0423i
0.0982 + 0.0169i
f =
3
r =
0.6981
i =
1 4
```

x = minres(A,b)

minres(A,b,tol)

minres(A,b,tol,maxit)

minres(A,b,tol,maxit,M)

minres(A,b,tol,maxit,M1,M2)

Try to solve the system Ax=b by the minimum residual method.

Solve Ax=b specifying the tolerance

Solve Ax=b specifying tolerance and maximum number of iterations

minres(A,b,tol,maxit,M1,M2,x0)

[x,f] = minres(A,b,...)

[x ,f,relres] = minres (A,b,...)

[x,f,relres,iter] = minres(A,b,...)

*Solve the system inv(M)*A*x = inv(M)*b*

*Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2*

*Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2 and initial value x0*

Solve the system and f indicates the result (0=convergence, 1=non-convergence, 2=conditional convergence, 3=stagnation and 4=very extreme numbers).

*It also returns the relative residual norm(b-A*x)/norm(b).*

Returns also the number of iterations

```
>> A=[pi 2*pi 3*pi -pi; 1 0 -1 2; exp(1) exp(2) exp(3)
exp(4); i 2i 3i -i];
>> B=[1 2 3 4]';
>> [x,f,r,i]=minres(A,B, 1e-006,1000)
x =
0.0748 - 0.0070i
-0.0761 - 0.0001i
0.5934 - 0.1085i
-0.1528 + 0.0380i
f =
1
r =
0.0592
i =
1000
```

4.4.5 SYMMETRIC AND NON-NEGATIVE LEAST SQUARES METHODS

T he following are the commands provided by MATLAB for solving equations and systems by the symmetric and non-negative least squares methods.

x = symmlq(A,b)

symmlq(A,b,tol)

symmlq(A,b,tol,maxit)

symmlq(A,b,tol,maxit,M)

symmlq(A,b,tol,maxit,M1,M2)

symmlq(A,b,tol,maxit,M1,M2,x0)

[x,flag] = symmlq(A,b,...)

[x ,flag,relres] = symmlq (A,b,...)

[x,flag,relres,iter] = symmlq(A,b,...)

Try to solve the system Ax=b by the symmetric LQ method.

Solve Ax=b specifying the tolerance

Solve Ax=b specifying tolerance and maximum number of iterations

*Solve the system inv(M)*A*x = inv(M)*b*

*Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2*

*Solve the system inv(M)*A*x = inv(M)*b with M=M1*M2 and initial value x0*

Solve the system and indicate the result (0=convergence, 1=non-convergence, 2=conditional convergence, 3=stagnation and 4=very extreme numbers).

*It also returns the relative residual norm(b-A*x)/norm(b).*

Returns also the number of iterations

```
>> A=[pi 2*pi 3*pi -pi; 1 0 -1 2; exp(1) exp(2) exp(3) exp(4); i 2i 3i -i];
>> B=[1 2 3 4]';
>> [x,f,r,i]=symmlq(A,B, 1e-006,1000)
x =
0.0121 - 0.0004i
0.0035 - 0.0001i
0.1467 - 0.0061i
0.0001 + 0.0039i



f =
1
r =
0.8325
i =
3
```

x = lsqnonneg(C,d)	*Gives x minimising norm(C*x-d) subject to x ≥0 by least squares (real C and d)*
x = lsqnonneg(C,d,x0)	*Gives x that minimises norm(C*x-d) subject to x ≥0, but with x=x0 ≥0 as initial value.*
x = lsqnonneg(C,d,x0,opt)	*Gives x that minimises norm(C*x-d) subject to x ≥0, with x=x0≥ 0 as the initial value and with the option opt. The options*

	are TolX for tolerance and Display for displaying the output ('off' does not display the output, 'final' displays the end of the output and 'notify' displays the output only if there is no convergence).
[x,resnorm] = lsqnonneg(...)	*Gives the solution and the value of the 2-square norm(C*x-d)^2*
[x,resnorm,residual] = lsqnonneg(...)	*Gives the solution and the residual C*x-d*
[x,resnorm,residual,f] = lsqnonneg(...)	*Gives the solution, the residual C*x-d and a positive or null f-value depending on whether the solution converges or not.*
[x,resnorm,residual,f,out,lambda] = lsqnonneg(...)	*Gives the solution, the C*x-d residual, the f-value, the algorithm used and the vector of Lagrange lambda multipliers.*

```
>> A=[1 2 3;5 7 1;2 3 6]; B=[1 3 5]'; lsqnonneg(A,B)
ans =
0.4857
0
0.5714
```

4.5 SOLVING LINEAR EQUATIONS AND SYSTEMS

In the previous paragraphs we have studied equations and systems in general. To work in this field it is possible to use the commands we have seen so far, but MATLAB dedicates a special area to linear equations and systems by implementing specific commands for solving them. The following table shows these commands.

X = linsolve(A,B)

*Solve the linear system A*X=B*

We solve the system:

$$2x + y + z + t = 1$$

$$x + 2y + z + t = 2$$

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

$$x + y + z + 2t = 4$$

```
>> A=[2,1,1,1;1,2,1,1;1,1,2,1;1,1,1,2];B=[1,2,3,4]';
linsolve(A,B)
ans =
-1
0
1
2
```

[X,R] = linsolve(A,B)

*Solves the linear system A*X=B and additionally returns the inverse of the condition number of A if A is square and returns its rank if A is not square.*

```
>> A=[2,1,1,1;1,2,1,1;1,1,2,1;1,1,1,2];B=[1,2,3,4]';
[X,R]=linsolve(A,B)
X =
-1
```

	0 1 2 R = 0.1429
X = linsolve(A,B,options)	*Solve the linear system A*X=B using different options for the matrix A (UT for upper triangular, LT for lower triangular, SYM for real or complex Hermitian symmetric, RECT for general rectangular, POSDEF for positive definite, UHESS for upper Hessemberg and TRANSA for transpose).*
rank(A)	*Rank of matrix A* >> rank(A) ans = 4
det (A)	*Determinant of the square matrix A* >> det(A) ans = 5
Z=null (A,'r')	*Rational basis for the core of A*

Systems of linear equations can be converted to matrix form and solved using matrix calculus. A system can be written in the form *M . X = B* , where *X* is the vector of variables, *B* is the vector of independent terms and *M* is the coefficient matrix of the system. If *M* is a square matrix and the determinant of the matrix *M* is non-zero, *M* is invertible, and the unique solution of the system can be written in the form: $X=M^{-1}B$. In this case, the commands *solve* , *linsolve* , *lscov, bicg, pcg, lsqr* , *gmr, gmres, minres, symmlq* or *M* , described above, provide the solution.

If the determinant of *M* is zero, the system has infinitely many solutions, so that there are rows or columns in *M* that are linearly dependent. In this case, the number of redundant equations will be calculated to find out how many variables we have to fix to give the infinite solutions. If the matrix *M* is rectangular (not square), the system may be indeterminate (number of equations less than the number of variables), overdetermined (number of equations greater than the number of variables) or non-singular (number of equations equal to number of variables and determinant of *M* non-zero). An indeterminate system can have infinitely many solutions or none at all, and the same happens to an overdetermined system. If there is no solution for a system, it is called inconsistent (incompatible), and if at least one solution exists, it is called consistent (compatible). The system *M* . *X* = *B* is called *homogeneous* when the vector *B* is the null vector. The system will be of the form *M . X = 0* . If the determinant of *M* is non-zero, the only solution of the system is the null vector (obtained with the command *linsolve*). If the determinant of *M* is null, the system has infinite solutions (it is indeterminate compatible). The solutions will be found by means of the *solve* or *linsolve, lsqr* or other commands already seen previously for general linear systems .

A fundamental tool for the analysis and solution of systems of equations is the *Rouché Frobenius theorem* . This theorem states that a system of *m* equations with *n* unknowns has some solution if, and only if, the rank of the matrix of coefficients coincides with the rank of the matrix expanded with the column vector of the independent terms of the system. If the two ranks are equal and equal to the number of unknowns, the system has a unique solution. If the two ranks are equal, but less than the number of unknowns, the system has infinite solutions. If they are different, the system has no solution.

Let *A* be the coefficient matrix of the system and *B* the extended matrix with the vector of independent terms.

If *range(A) ≠range(B)* , the system is incompatible (unsolvable).

If *rank(A)=rank(B) < n* , the system is indeterminate compatible (has infinitely many solutions).

If *rank(A)=rank(B) = n* , the system is determinate compatible (has a unique solution).

This theorem makes it possible to analyse the solutions of a system of equations before solving it.

We already know that another special type of linear systems are *homogeneous systems* . The system *A.X=B* is said to be homogeneous if the vector of independent terms *B* is zero, so that any homogeneous system will be of the form *A.X=0* . In a homogeneous system, the rank of the coefficient matrix and the rank of the matrix extended with the column of independent terms always coincide. If we apply the Rouché-Frobenius theorem, a homogeneous system will have a unique solution when the determinant of the matrix *A* is non-zero. This unique solution is the null vector, since the null solution is always found in homogeneous systems. A homogeneous system will have infinitely many solutions when the determinant of the matrix *A* is zero. In this case, the infinite solutions are calculated in the same way as in general systems (*solve* command), or also using the *null* function *(A).*

As a first example, we solve the system:

$$2x + y + z + t = 1$$

$$x + 2y + z + t = 1$$

$$x + y + 2z + t = 1$$

$$x + y + z + 2t = 1$$

We will find the rank of the system matrix and of the extended matrix with the column of independent terms.

```
>> A=[2,1,1,1;1,2,1,1;1,1,2,1;1,1,1,2];
>> B=[2,1,1,1,1;1,2,1,1,1;1,1,2,1,1;1,1,1,2,1];
>> [rank(A), rank(B)] >> [rank(A), rank(B)].
ans =
4 4
```

We observe that the ranks of the two matrices coincide with the number of unknowns. The Rouché-Frobenius theorem then ensures that the system is determinate compatible with a unique solution. We can calculate the solution as follows:

```
>> B=[1 1 1 1]';
>> linsolve(A,B)
ans =
0.2000
0.2000
0.2000
0.2000
```

The system could also have been solved with the following commands:

```
>> lscov(A,B)
ans =
0.2000
0.2000
0.2000
0.2000
```

```
>> bicg(A,B)
bicg converged at iteration 1 to a solution with relative residual 0
ans =
0.2000
0.2000
0.2000
0.2000
>> pcg(A,B)
pcg converged at iteration 1 to a solution with relative residual 0
ans =
0.2000
0.2000
0.2000
0.2000
>> lsqr(A,B)
lsqr converged at iteration 1 to a solution with relative residual 0
ans =
0.2000
0.2000
0.2000
0.2000
>> qmr(A,B)
qmr converged at iteration 1 to a solution with relative residual 0
ans =
0.2000
0.2000
0.2000
0.2000
>> gmres(A,B)
gmres converged at iteration 1 to a solution with relative residual 1.5e-016
ans =
0.2000
0.2000
0.2000
0.2000
>> symmlq(A,B)
symmlq converged at iteration 1 to a solution with relative residual 0
ans =
0.2000
0.2000
0.2000
0.2000
```

As a second example, we solve the system:

$$x + 2y + 3z = 6$$

$$x + 3\ y + 8z = 19$$

$$2x + 3y + z = -1$$

$$5x + 6\ y + 4z = 5$$

We will find the rank of the system matrix and of the extended matrix with the column of independent terms.

```
>> A=[1,2,3;1,3,8;2,3,1;5,6,4];
>> B=[1,2,3,6;1,3,8,19;2,3,1,-1;5,6,4,5];
>> [rank(A), rank(B)] >> [rank(A), rank(B)].
ans =
3 3
```

We observe that the ranks of the two matrices coincide with the number of unknowns. The Rouché-Frobenius theorem then ensures that the system is determinate compatible with a unique solution. We can calculate the solution as follows:

```
>> A=[1,2,3;1,3,8;2,3,1;5,6,4];
>> B=[6 19 -1 5]';
>> linsolve(A,B)
ans =
1.0000
-2.0000
3.0000
```

As a third example, we solve the system:

$$x + 2y - z = 0$$

$$2x - y + z = 0$$

$$3x + y = 0$$

As we are dealing with a homogeneous system, we will calculate the determinant of the coefficient matrix of the system.

```
>> A=[1,2,-1;2,-1,1;3,1,0];
>> det(A)
ans =
5.5511e-016
```

Since the determinant is practically zero, the homogeneous system will have infinitely many solutions, which are calculated with the *solve* command as shown below.

```
    >> [x,y,z]=solve('x+2*y-z, 2*x-y+z, 3*x+y', 'x,y,z')
    x =
    -z1/5



y =
    (3*z1)/5
```

```
z =
   z1
```

It is observed that the infinite solutions depend on the parameter *z1* and can be written as follows:

{(-z1/5, 3z1/5, *z1*)}z1 image *R*

Exercise 1. Expand the following algebraic expressions:

image

```
" syms x y z t a b
"pretty(expand((x+1)*(x+2))))
2
x + 3 x + 2
" pretty(expand((x+1)/(x+2)))
x 1
———+———-
x + 2 x + 2
"pretty(expand(sin(x+y)))
sin(x) cos(y) + cos(x) sin(y)
"pretty(expand(cos(2*x)))
2
2 cos(x) - 1
"pretty(expand(exp(a+log(b))))
exp(a) b
" pretty(expand(log(x/(1-x)^2))))
                                   log(x) - 2 log(1 - x)
"pretty(expand((x+1)*(y+z)))
x y + x z + y + z
```

Factor the following algebraic expressions:

image

```
>> syms x y
    >> pretty(factor(6*x^2+18*x-24))
    6 (x + 4) (x - 1)
    >> pretty(factor(x^4-y^4))
    2 2
    (x - y) (x + y) (x + y )
    >> pretty(factor(x^3+y^3))
    2 2
    (x + y) (x - x y + y )
    >> pretty(factor((x^3-y^3)/(x^4-y^4)))
    2 2
    x + x y + y
______________-__

2 2
    (x + y) (x + y )
```

E xercise 3. Simplify the following algebraic expressions:

image

```
>> syms x y a b c
    >> simplify(sin(x)^2 + cos(x)^2)
    ans =
    1
    >> pretty(simplify(exp(a+log(b*exp(c)))))
    b exp(a + c)
    >> pretty(sym(simple(cos(3*acos(x)))))
    3
    4 x - 3 x
    >> pretty(simple( (x^2-y^2)/(x-y)^3 ))
    x + y
```

```
2
     (x - y)
```

Group x terms in the following algebraic expressions:

$f(x) = a^{3} x - x + a^{3} x + a$, $p(x)= y/x+2z/x+x+x^{1/3} -y*x^{1/3}$, $q(x)= (x+1)(x+2)$

Grouping terms in sin(x) in the expression: y(sin(x)+1)+sen(x)

Grouping terms in Ln(x) in the expression: f=aLn(x)-xLn(x)-x

Group terms in (x,y) in the expression: $p=xy+zxy+yx^{2} +zyx^{2} +x+zx+zx$

```
>> syms a x y z
     >> pretty(collect(a^3*x-x+a^3+a, x))
     3 3
     (a - 1) x + a + a
     >> pretty(collect(y/x+2*z/x+x^(1/3)-y*x^(1/3), x)))
     4 4

-    -

3 3
```

```
   y + 2 z - x y + x
———————————-——
X
   >> pretty(collect( (x+1)*(x+2) ))
   2
   x + 3 x + 2
   >> p = x*y+z*x*y+y*x^2-z*y*x^2+x+z*x;
   >> pretty(collect( p, [x,y] ))
   2
   (1 - z) x y + (z + 1) x y + (z + 1) x
   >> f = a*log(x)-log(x)*x-x;
   >> pretty(collect(f,log(x)))
   (a - x) log(x) - x
```

Exercise 5. Combine terms as much as possible in the following expression:

aln(x)+3ln(x)-ln(1-x)+ln(1+x)/2

Carry out the simplification assuming that a is real and x>0.

```
   >> pretty(sym(simple(a*log(x)+3*log(x)-log(1-x)+log(1+x)/2))))
   log(x + 1)
   ——————————- log(1 - x) + 3 log(x) + a log(x)
   2
   >> x = sym('x', 'positive')
   x =
   x
   >> a = sym('a', 'real')
   a =
   a
   >> pretty(sym(simple(a*log(x)+3*log(x)-log(1-x)+log(1+x)/2))))
   / x - 1 \
   - log| -————————————|
   | 3 a 1/2 |
   \ x x (x + 1) /
```

Develop and simplify as much as possible the following trigonometric expressions:

a) Sin[3x]Cos[5x]

(b) [(Cot[a]) ^{2} *+(Sec[a])* ^{2} *-(Csc[a])* ^{2}

c) Sin[a]/(1+Cot[a] ^{2} *)-Sin[a]* ^{3}

```
>> pretty(simple(expand(sym(sin(3*x)*cos(5*x)))))
sin(8 x) sin(2 x)

—————-—————

2 2
>> pretty(simple(expand(((cot(a))^2+(sec(a))^2-(csc(a))^2))))
1
———-——- 1
2
cos(a)
>> pretty(simple(expand(expand(sin(a)/(1+cot(a)^2)-sin(a)^3)))))
0
```

Exercise 7. Carry out the following algebraic operations, simplifying the result as much as possible:

```
x/(x+y)-y/(x-y)+2*x*y/(x^2-y^2)
(1+a^2)/b + (1-b^2)/a - (a^3-b^3)/(a*b)
```

```
>> pretty(simple(expand (x/(x+y)-y/(x-y)+2*x*y/(x^2-y^2))))
1
```

```
>> pretty(simple(expand((1+a^2)/b + (1-b^2)/a - (a^3-b^3)/(a*b))))
1   1
- + -
a   b
```

Exercise 8. Simplify the following algebraic fractions as much as possible:

a^3-a^2*b+a*c^2-b*c^2)/(a^3+a*c^2+a^2*b+b*c^2)

((x^2-9)*(x^2-2*x+1)*(x-3))/((x^2-6*x+9)*(x^2-1) *(x-1))

```
>>pretty(simple(factor(a^3-a^2*b+a*c^2-b*c^2)/(a^3+a*c^2+a^2*b+b*c^2)))))
a - b

—-—

a + b
>>pretty(simple(factor((x^2-9)*(x^2-2*x+1)*(x-3))/((x^2-6*x+9)*(x^2-1) *(x-1))))
2
—-—+ 1
x + 1
```

Exercise 9. Calculate the roots of the following polynomials:

image

Evaluate the first polynomial in the unit matrix of order 3, the second in the ones matrix of order 3 and the third in a uniform random matrix of order 3.

Find the coefficients of the polynomials derived from the given polynomials, as well as the polynomials themselves.

```
>> p1 = [1 -6 -72 -27];r=roots(p)
r =
12.1229
-5.7345
-0.3884
```

```
>> p2 = [2 -3 4 -5 11];r=roots(p)
r =
1.2817 + 1.0040i
1.2817 - 1.0040i
-0.5317 + 1.3387i
-0.5317 - 1.3387i
>> p3 = [1 0 0 0 0 0 0 0 0 0 0 0 0 1];r=roots(p)
r =
-1.0000
-0.8413 + 0.5406i
-0.8413 - 0.5406i
-0.4154 + 0.9096i
-0.4154 - 0.9096i
0.1423 + 0.9898i
0.1423 - 0.9898i
0.6549 + 0.7557i
0.6549 - 0.7557i
0.9595 + 0.2817i
0.9595 - 0.2817i
>> Y1=polyval(p1,eye(3))
Y1 =
-104 -27 -27
-27 -104 -27
-27 -27 -104
>> Y2=polyval(p2,ones(3))
Y2 =
9 9 9
9 9 9
9 9 9
>> Y3=polyval(p3,rand(3))
Y3 =
1.1050 1.3691 1.0000
1.3368 1.0065 1.0013
1.0000 1.0000 1.6202
>> d1=polyder(p1)
d1 =
3 -12 -72
>> pretty(poly2sym(d1,'x'))
2
3 x - 12 x - 72
>> d2=polyder(p2)
d2 =
8 -9 8 -5
>> pretty(poly2sym(d2,'x'))
3 2
8 x - 9 x + 8 x - 5
>> d3=polyder(p3)
d3 =
11 0 0 0 0 0 0 0 0 0 0
>> pretty(poly2sym(d3,'x'))
10
11 x
```

Consider the points of the interval [0 5] separated by one tenth. Interpolate the error function at these points and fit a polynomial of degree 6 to them. Plot the original curve and the interpolated curve on the same graph.

```
>> x = (0: 0.1: 5)';
y = erf(x);
f = polyval(p,x);
>> p = polyfit(x,y,6)
p =
0.0012 -0.0173 0.0812 -0.0791 -0.4495 1.3107 -0.0128
>> f = polyval(p,x);
plot(x,y,'o',x,f,'-')
axis([0 5 0 2])
```

C alculate the second degree interpolating polynomial passing through the points (-1,4), (0,2), and (1,6) in the least squares sense.

```
» x=[-1,0,1];y=[4,2,6];p=poly2sym(polyfit(x,y,2))
p =
3*x^2+x+2
```

Plot 200 cubic interpolation points between the points (x,y) given by the values taken by the exponential function e ^{x} for 20 values of x equally spaced between 0 and 2. Use cubic interpolation.

First, we define the given 20 points *(x,y)* , equally spaced between 0 and 2:

```
" x=0:0.1:2;
"y=exp(x);
```

We now find the 200 cubic interpolation points *(xi,yi)* , equally spaced between 0 and 2, and plot them on a graph, together with the initial 20 *(x,y)* points (with asterisks).

```
"xi=0:0.01:2;
"yi=interp1(x,y,xi,'cubic');
" plot(x,y,'*',xi,yi)
```

O btain 25 points of approximation by interpolation of the parametric function X=Cosh(t), Y=Senh(t), Z=Tanh(t) for values of t between 0 and *π/6, on the set of points defined for values of t in the interval (i π/6) with 0 i 6≤≤*

First, we define the given 25 points *(x,y,z)* , equally spaced between 0 and *π* ***/6.***

```
>> t=0:pi/150:pi/6;
>> x=cosh(t);y=sinh(t);z=tanh(t) ;
```

Now we find the 25 interpolation points (x_i, y_i, z_i) , for values of the parameter *t* equally spaced between 0 and *π* ***/6.***

```
"xi=cosh(t);yi=sinh(t);
"zi=griddata(x,y,z,xi,yi) ;
" points=[xi',yi',zi']
points = points = points = points = points = points = points = points = points = points = points = points
1.0000 0 0
1.0002 0.0209 0.0209
1.0009 0.0419 0.0419
1.0020 0.0629 0.0627
1.0035 0.0839 0.0836
1.0055 0.1049 0.1043
1.0079 0.1260 0.1250
1.0108 0.1471 0.1456
1.0141 0.1683 0.1660
1.0178 0.1896 0.1863
1.0220 0.2110 0.2064
1.0267 0.2324 0.2264
1.0317 0.2540 0.2462
1.0373 0.2756 0.2657
1.0433 0.2974 0.2851
1.0498 0.3194 0.3042
1.0567 0.3414 0.3231
```

```
1.0641 0.3636 0.3417

1.0719 0.3860 0.3601

1.0802 0.4085 0.3782

1.0890 0.4312 0.3960

1.0983 0.4541 0.4135

1.1080 0.4772 0.4307

1.1183 0.5006 0.4476

1.1290 0.5241 0.4642

1.1402 0.5479 0.4805
```

Obtain 30 points (xi,yi) approximating the function y=Senh(x) for equally spaced values of x, interpolating them between 20 values of (x,y) given by y=Senh(x) for evenly spaced values of x in the interval (0,2 π), and using the interpolation method based on the Fast Fourier Transform (FFT). Plot the points graphically.

First, we define the 20 values of *x* equally spaced between 0 and 2 *π* .

```
"x=(0:pi/10:2*pi);
```

Now we find the 30 interpolation points *(x,y)* requested.

```
>> y=interpft(sinh(x),30);

>> points=[y',(asinh(y))']

points = points = points = points = points = points = points = points = points = points = points = points

-0.0000 -0.0000

-28.2506 -4.0346

23.3719 3.8451

-4.9711 -2.3067

-7.7918 -2.7503

14.0406 3.3364

-4.8129 -2.2751

-0.8717 -0.7877

11.5537 3.1420
```

```
-3.3804 -1.9323
4.4531 2.1991
11.8616 3.1682
-0.2121 -0.2105
10.9811 3.0914
15.1648 3.4132
6.1408 2.5147
21.2540 3.7502
23.3792 3.8455
18.5918 3.6166
39.4061 4.3672
40.6473 4.3982
42.8049 4.4499
73.2876 4.9876
74.8962 5.0093
89.7159 5.1898
139.0371 5.6279
139.3869 5.6304
180.2289 5.8874
282.4798 6.3368
201.7871 6.0004
>> plot(points)
```

image

F ind the polynomial of degree 3 that best fits the cloud of points (i, i^{2}) even 1 i ≤7≤, in the least squares sense. Find its value for x=10 and plot the fitting curve.

```
>> x=(1:7);y=[1,4,9,16,25,36,49];p=vpa(poly2sym(polyfit(x,y,2)))
p =
x^2 - 0.000000000000000000916169181662272872871686413366801652*x + 0.0000000000000000020761891472015924526365781895317
```

Now we calculate the numerical value of the polynomial *p* at *x=10* .

```
>> subs(p,10)

ans =

100.0000
```

The fit plot is shown below:

```
" ezplot(p,[-5,5])
```

image

F ind solutions to the following equations:

Sen(x)Cos(x)=0, Sen(x)=aCos(x), ax^2+bx+c=0 and Sen(x)+Cos(x)=sqrt(3)/2

```
" solve('sin(x)*cos(x)=0')

ans =

[ 0]

[ 1/2*pi]

[ -1/2*pi]

" solve('sin(x)=a*cos(x)','x')

ans =

tie(a)

" solve('a*x^2+b*x+c=0','x')

ans =

[1/2/a*(-b+(b^2-4*a*c)^(1/2))]

[1/2/a*(-b-(b^2-4*a*c)^(1/2))]

" solve('sin(x)+cos(x)=sqrt(3)/2')

ans =

[1/2*3^(1/2)]

[1/2*3^(1/2)]
```

Find at least two solutions for each of the following two trigonometric and exponential equations:

$$xSen(x)=1/2 \ y \ 2^{x\wedge 3}=4(2^{3x})$$

Initially, we use the *solve* command:

```
>> vpa(solve('x*sin(x)=1/2','x'))
ans =
matrix([[-226.19688152398440474751335389781]])
>> vpa(solve('2^(x^3)=4*2^(3*x)','x')))
ans =
2.0
-1.0
-1.0
```

For the first equation we do not obtain solutions, but for the second one we do. To better analyse the first equation, we make a graphical representation to see approximately the intervals where the possible solutions fall.

```
» fplot('[x*sin(x)-1/2,0]',[0,4*pi])
```

image

Y ou can see that there is a solution between 0 and 2, another between 2 and 4, another between 4 and 8, and so on. Let's calculate three of them with the command *fzero* .

```
>> s1=fzero('x*sin(x)-1/2',2)
s1 =
0.7408
>> s2=fzero('x*sin(x)-1/2',4)
s2 =
2.9726
```

```
>> s3=fzero('x*sin(x)-1/2',6)
s3 =
6.3619
```

Solve each of the following two logarithmic and irrational equations:

$x^{3/2}$ log(x) = x log($x^{3/2}$), sqrt[1-x]+sqrt[1+x] = a

```
>> vpa(solve('x^(3/2 )*log(x)=x*log(x)^(3/2)')))
   ans =
  1.0
  0.31813150520476413531265425158766 - 1.3372357014306894089011621431937*i
```

We then carry out the graphical representation in order to be able to intuit the intervals in which the possible solutions are found. It is found that *x=1 is the* only real solution.

```
>> fplot('[x^(3/2)*log(x),x*log(x)^(3/2)]',[0,3,-1,6])
```

image

N ow, let's solve the irrational equation:

```
>> pretty(sym(solve('sqrt(1-x)+sqrt(1+x)=a','x'))))
+- -+
| 2 1/2 |
| a (4 - a ) |
|——————-——|
| 2 |
| |
| 2 1/2 |
| a (4 - a ) |
| -——————-——|
| 2 |
+- -+
```

Exercise 19. Solve the following system of two equations:

cos(x/12)/exp(x ^{2} /16)=y

-5/4 +y = sin(x $^{3/2}$)

```
>> [x,y]=solve('cos(x/12)/exp(x^2/16)=y','-5/4+y=sin(x^(3/2))')

x =

0.3456974417012631933103328363622 8*i - 0.18864189802267887925036526820236

y =

0.0086520715192230549621145978569268*i + 1.0055146234480859930589058368368
```

Find the intersection of the hyperbolas of equations x ^{2} -y ^{2} =r ^{2} and a ^{2} x ^{2} -b ^{2} y ^{2} =a ^{2} b ^{2} with the parabola z ^{2} = 2px.

```
>>[x,y,z]=solve('a^2*x^2-b^2*y^2=a^2*b^2','x^2-y^2=r^2','z^2=2*p*x','x,y,z')

x =

((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/2)/(2*p)

((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/2)/(2*p)

((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/2)/(2*p)

-((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/2)/(2*p)

((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/2)/(2*p)

-((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/2)/(2*p)

-((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/2)/(2*p)

-((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/2)/(2*p)

y =

a*((b^2 - r^2)/(a^2 - b^2))^(1/2)

-a*((b^2 - r^2)/(a^2 - b^2))^(1/2)

a*((b^2 - r^2)/(a^2 - b^2))^(1/2)

a*((b^2 - r^2)/(a^2 - b^2))^(1/2)

-a*((b^2 - r^2)/(a^2 - b^2))^(1/2)

a*((b^2 - r^2)/(a^2 - b^2))^(1/2)

-a*((b^2 - r^2)/(a^2 - b^2))^(1/2)

-a*((b^2 - r^2)/(a^2 - b^2))^(1/2)
```

```
z =
((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/4)
((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/4)
-((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/4)
((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/4)*i
-((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/4)
-((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/4)*i
((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/4)*i
-((4*a^2*b^2*p^2 - 4*b^2*p^2*r^2)/(a^2 - b^2))^(1/4)*i
```

E xercise 21. Study and solve the system:

$x1 - x2 + x3 = 1$

$4 x1 + 5 x2 - 5 x3 = 4$

$2 x1 + x2 - x3 = 2$

$x1 + 2 x2 - 2 x3 = 1$

```
» A=[1,-1,1;4,5,-5;2,1,-1;1,2,-2]
A =
1 -1 1
4 5 -5
2 1 -1
1 2 -2
» B=[1,-1,1,1;4,5,-5,4;2,1,-1,2;1,2,-2,1]
B =
1 -1 1 1
4 5 -5 4
2 1 -1 2
1 2 -2 1
" [rank(A), rank(B)] [rank(A), rank(B)] [rank(A), rank(B)].
```

```
    ans =
    2 2
```

We see that the rank of *A* and *B* coincide and their value is 2, which is less than the number of unknowns of the system (3). Therefore, the system will have infinite solutions (indeterminate compatible). We try to solve it with the *solve* command:

```
    >>        [x1,x2,x3]=solve('x1-x2+x3=1','4*x1+5*x2-5*x3=4','2*x1+x2-x3=2',        'x1+2*x2-
2*x3=1','x1','x2','x3')
    Warning: 4 equations in 3 variables.
    x1 =
    1



x 2 =
    z



x 3 =
    z
```

The infinite solutions are obtained by giving the parameter *x3* the infinite real values ($\{1,z,z\}\ \forall z \in R$). Note that the trivial solution {1,0,0} is obtained for the value of the parameter equal to zero.

S tudy and solve the system:

$$x + 2y + 3z + t = 6$$

$$x + 3y + 8z + t = 19$$

```
    " A=[1,2,3,1;1,3,8,1]
    A =
    1 2 3 1
    1 3 8 1
    " B=[1,2,3,1,6;1,3,8,1,19]
    B =
    1 2 3 1 6
    1 3 8 1 19
    " [rank(A), rank(B)] [rank(A), rank(B)] [rank(A), rank(B)].
    ans =
    2 2
```

We see that the range of *A* and *B* coincide and their value is 2, which is less than the number of unknowns in the system (4). Therefore, the system has infinite solutions (indeterminate compatible). We try to solve it:

```
    >> [x,y,z,t]=solve('x+2*y+3*z+t=6','x+3*y+8*z+t=19','x','y','z','t')
    Warning: 2 equations in 4 variables. New variables might be introduced.
    x =
    7*z1 - z2 - 20



y =
    z2
    z =
    13 - 5*z1



t =
    z1
```

This time the solution depends on two parameters. By varying *y* and *t* in the real numbers, the infinite solutions of the system are obtained. These solutions form a two-dimensional vector subspace of the two-dimensional real vector space which can be expressed as follows:

{7z1-z2-20,z2,13-5z1,z1} ∀z1,z2 ∈R

Study and solve the system:

$3\ x1 + x2 + x3 - x4 = 0$

$2\ x1 + x2 - x3 + x4 = 0$

$x1 + 2\ x2 + 4\ x3 + 2\ x4 = 0$

$2\ x1 + x2 - 2\ x3 - x4 = 0$

```
» det([3,1,1,-1;2,1,-1,1;1,2,4,2;2,1,-2,-1])
ans =
-30
```

As the determinant of the coefficient matrix is non-zero, the system has only the trivial solution:

```
>> [x1,x2,x3,x4]=solve('3*x1+x2+x3-x4=0','2*x1+x2-x3+x4=0','x1+2*x2-4*x3-2*x4=0','x1-x2-3*x3-5*x4=0','x1','x2','x3','x4')
x1 =
0
x2 =
0
x3 =
0
x4 =
0
```

Study and solve, according to the values of m, the system:

$m\ x + y + z = 1$

x + m y + z = m

x + y + m z = m^2

```
>> syms m
>> A=[m,1,1;1,m,1;1,1,1,m]
A =
[ m, 1, 1]
[ 1, m, 1]
[ 1, 1, m]
>> det(A)
ans =
m^3 - 3*m + 2
>> solve('m^3 - 3*m + 2=0','m')
ans =
-2
1
1
```

The values of *m* that determine the rank of the matrix are -2 and 1.

We now consider the extended matrix as a function of *m* :

```
>> B=[m,1,1,1;1,m,1,m;1,1,m,m^2]
B =
[ m, 1, 1, 1]
[ 1, m, 1, m]
[1, 1, 1, m, m^2]
```

Let's study the case *m = -2* :

```
>> rank(subs(A,{m},{-2}))
ans =
2
>> rank(subs(B,{m},{-2}))
ans =
    3
```

We see that the rank of the two matrices is different, so the system is incompatible (has no solution) for *m=-2*.

We now study the case *m=1* :

```
    >> rank(subs(A,{m},{1}))
    ans =
    1
    >> rank(subs(B,{m},{1}))
    ans =
    1
```

Now the rank of the two matrices is 1, which is less than the number of unknowns. Therefore, the system has infinite solutions (indeterminate compatible). We find them by substituting *m=1* in the initial system:

```
    >> [x,y,z]=solve('x+y+z=1','x','y','z')
    Warning: 1 equations in 3 variables. New variables might be introduced.
    x =
    1 - z2 - z1



y =
    z2



z =
    z1
```

Then, the infinite solutions are obtained by varying the parameters *y* and *z* in the real numbers. The subspace of solutions of dimension 2 is:

{1-z2-z1,z2,z1} ∀z1,z2 ∈R

If we consider the case where *m* is different from -2 and -1, the system has a unique solution, which is given by the *solve* command:

```
>> [x,y,z]=solve('m*x+y+z=1','x+m*y+z=m','x+y+m*z=m^2','x','y','z')
x =
-(m + 1)/(m + 2)
y =
1/(m + 2)
```

```
z =
(m^2 + 2*m + 1)/(m + 2)
```

Study and solve, according to the values of m, the system:

m y = m

(1 + m) x - z = m

y + z = m

```
>> syms m
>> A=[0,m,0;m+1,0,-1;0,1,1,1]
A =
[ 0, m, 0]
[ m + 1, 0, -1]
[ 0, 1, 1]
>> det(A)
ans =
- m^2 - m
>> solve('- m^2 - m=0','m')
ans =
-1
0
```

We see that the values of *m* that determine the rank of the coefficient matrix of the system are *m=-1* and *m=0* .

We now consider the extended matrix:

```
>> B=[0,m,0,m;m+1,0,-1,m;0,1,1,m]
B =
[ 0, m, 0, m]
[m + 1, 0, -1, m]
[ 0, 1, 1, m]
>> rank(subs(A,{m},{-1}))
ans =
2
>> rank(subs(B,{m},{-1}))
ans =
3
```

In the case *m=-1* , we see that the system has no solution (it is incompatible) because the rank of the matrix of the coefficients of the system is 2 and that of the extended matrix is 3.

Now, we analyse the case *m=0:*

When *m* is zero, the system is homogeneous, since the independent terms are all zero. To study it, we analyse the determinant of the matrix of the coefficients of the system.

```
>> det(subs(A,{m},{0}))
ans =
0
```

Since the determinant is null, the system has infinitely many solutions:

```
>> [x,y,z]=solve('x-z=0','y+z=0','x','y','z')
Warning: 2 equations in 3 variables. New variables might be introduced.
x =
    z1


y =
    -z1


z =
    z1
```

Then, the infinite solutions will be given when the parameter *z* varies in the real numbers. The subspace of solutions of dimension 1 is:

{z1,-z1,z1} ∀z1 ∈R

If *m* is different from 0 and different from -1, the system is determinate compatible (unique solution), since the ranks of the system matrix and the extended matrix coincide. The solution is calculated using the *solve* function.

```
    >> [x,y,z]=solve('m*y=m','(1+m)*x-z=m','y+z=m','x','y','z')
    x =
    (2*m - 1)/(m + 1)


y =
    1


z =
    m - 1
```

Study and solve the system:

$$2x + y + z + t = 1$$

$$x + 2y + z + t = 1$$

$$x + y + 2z + t = 1$$

$$x + y + z + 2t = 1$$

```
>> A=[2,1,1,1;1,2,1,1;1,1,2,1;1,1,1,2];
>> B=[2,1,1,1,1;1,2,1,1,1;1,1,2,1,1;1,1,1,2,1];
>> [rank(A), rank(B)] >> [rank(A), rank(B)].
ans =
4 4
>> b=[1,1,1,1]';
```

We see that the matrices *A* and *B* (expanded) have rank 4, which, moreover, coincides with the number of unknowns. Therefore, the system has a unique solution (it is determinate compatible with *the same number of equations and unknowns*). To calculate its solution, any of the multiple commands seen can be used.

```
>> x = nnls(A,b)
x =
0.2000
0.2000
0.2000
0.2000
>> x = bicg(A,b)
bicg converged at iteration 1 to a solution with relative residual 0
x =
0.2000
0.2000
0.2000
```

```
0.2000
>> x = bicgstab(A,b)
bicgstab converged at iteration 0.5 to a solution with relative residual 0
x =
0.2000
0.2000
0.2000
0.2000
>> x = pcg(A,b)
pcg converged at iteration 1 to a solution with relative residual 0
x =
0.2000
0.2000
0.2000
0.2000
>> gmres(A,b)
gmres converged at iteration 1 to a solution with relative residual 0
ans =
0.2000
0.2000
0.2000
0.2000
>> x = lsqr(A,b)
lsqr converged at iteration 2 to a solution with relative residual 0
x =
0.2000
0.2000
0.2000
0.2000
>> A
ans =
```

0.2000

0.2000

0.2000

0.2000

Chapter 5.

MATRICES, VECTOR SPACES AND LINEAR APPLICATIONS

In the chapter on vector and matrix variables, we have already studied the ways of representing vectors and matrices in MATLAB. However, we recall here the notation.

The matrix:

can be entered into Matlab in the following ways:

A=[a_{11},a_{12},...,a_{1n}; a_{21},a_{22},...,a_{2n}; ... ; a_{m1},a_{m2},...,a_{mn}]

A=[a_{11} a_{12} ... a_{1n}; a_{21} a_{22} ... a_{2n}; ... ; a_{m1} a_{m2} ... a_{mn}]

On the other hand, the vector *V=(v1,v2,...,vn)* is introduced as a particular case of a single row matrix (matrix of dimension *1xn*) or in the following form:

V=[v1, v2, ..., vn]

V=[v1 v2 ... vn]

5.1 OPERATIONS WITH MATRICES

M ATLAB incorporates commands that allow you to perform the most common operations with matrices, both numeric and symbolic. The following table shows the most important ones.

A+B	*Sum of matrices A and B* >> A=[-1 7 2;i -i 8; 3 1 4] A = -1.0000 7.0000 2.0000 0 + 1.0000i 0 - 1.0000i 8.0000 3.0000 1.0000 4.0000 >> B=[1 2 3; 1+i 1-2i 1-3i; -3 -5 -i]. B = 1.0000 2.0000 3.0000 1.0000 + 1.0000i 1.0000 - 2.0000i 1.0000 - 3.0000i -3.0000 -5.0000 0 - 1.0000i >> A+B ans = 0 9.0000 5.0000 1.0000 + 2.0000i 1.0000 - 3.0000i 9.0000 - 3.0000i 0 -4.0000 4.0000 - 1.0000i
A-B	*Difference of matrices A and B* >> A-B ans = -2.0000 5.0000 -1.0000 -1.0000 -1.0000 + 1.0000i 7.0000 + 3.0000i 6.0000 6.0000 4.0000 + 1.0000i
A*B	*Product of matrices A and B* >> A*B ans = 0 + 7.0000i -5.0000 -14.0000i 4.0000 -23.0000i -23.0000 -42.0000 + 1.0000i -3.0000 - 6.0000i -8.0000 + 1.0000i -13.0000 - 2.0000i 10.0000 - 7.0000i
c*A	*Product of a scalar by a matrix*

```
>> i*A
ans =
0 - 1.0000i 0 + 7.0000i 0 + 2.0000i
-1.0000 1.0000 0 + 8.0000i
0 + 3.0000i 0 + 1.0000i 0 + 4.0000i
```

expm(A)

e^{A} calculated through eigenvalues

```
>> expm(A)
ans =
1.0e+003 *
0.5710 + 0.0674i 0.7081 - 0.0337i 1.7019 - 0.0338i
0.5710 + 0.0674i 0.7081 - 0.0337i 1.7018 - 0.0337i
0.5710 + 0.0675i 0.7082 - 0.0337i 1.7018 - 0.0337i
```

expm1(A)

e^{A} calculated by means of Padé approximants

```
> expm1(A)
ans =
1.0e+003 *
-0.0006 1.0956 0.0064
-0.0005 + 0.0008i -0.0005 - 0.0008i 2.9800
0.0191 0.0017 0.0536
```

logm(A)

Neperian logarithm of matrix A

```
>> logm(A)
ans =
1.3756 + 0.0063i 2.4201 + 0.4906i -1.7163 - 0.4969i
-0.9903 + 0.1445i 1.7284 - 0.2954i 1.3413 + 0.1509i
0.6464 - 0.0412i -0.6375 - 0.1572i 2.0705 + 0.1984i
```

sqrtm(A)

Square root of the square matrix A

```
>> sqrtm(A)
ans =
1.1538 - 0.0556i 2.4904 + 0.3474i -0.8158 - 0.2918i
-0.6332 + 0.2331i 1.5192 - 0.3779i 1.9424 + 0.1449i
0.8187 - 0.0083i -0.2678 - 0.0893i 2.2775 + 0.0975i
```

funm(A, 'function')

Applies the function to the square matrix A. The functions that can be used are EXP, LOG,

COS, SIN, COSH, SINH, SQRTM and EXPM.

```
>> funm(A,'log')
ans =
1.3756 + 0.0063i 2.4201 + 0.4906i -1.7163 - 0.4969i
-0.9903 + 0.1445i 1.7284 - 0.2954i 1.3413 + 0.1509i
0.6464 - 0.0412i -0.6375 - 0.1572i 2.0705 + 0.1984i
>> funm(A,'sinh')
ans =
1.0e+002 *
2.8840 + 0.3729i 3.4704 - 0.1953i 8.5505 - 0.1776i
2.8821 + 0.3256i 3.5574 - 0.1265i 8.4652 - 0.1992i
2.8359 + 0.3291i 3.5556 - 0.1738i 8.5133 - 0.1553i
>> funm(A,'cos')
ans =
-22.0565 + 8.2759i 20.1371 -22.2895i 1.7740 +14.0136i
2.0152 +10.6436i -22.5464 + 2.7537i 20.3857 -13.3973i
6.7526 - 6.1638i 1.5253 + 5.1214i -8.4234 + 1.0424i
```

Transposed matrix of A

transpose(A) or A'

```
>> transpose(A)
ans =
-1.0000 0 + 1.0000i 3.0000
7.0000 0 - 1.0000i 1.0000
2.0000 8.0000 4.0000
```

Inverse matrix of square matrix A (A^{-1})

inv(A)

```
>> inv(A)
ans =
-0.0430 - 0.0266i -0.1465 - 0.0133i 0.3145 + 0.0400i
0.1373 - 0.0102i -0.0564 - 0.0051i 0.0441 + 0.0154i
-0.0020 + 0.0225i 0.1240 + 0.0113i 0.0031 - 0.0338i
>> A*INV(A)
??? Undefined function or method 'INV' for input arguments of type
double'.
>> A*inv(A)
ans =
1.0000 + 0.0000i 0 + 0.0000i -0.0000i -0.0000
0 1.0000 + 0.0000i 0.0000 + 0.0000i
0.0000 + 0.0000i 0.0000 + 0.0000i 1.0000
```

det(A)

Determinant of the square matrix A

```
>> det(A)
ans =
1.7600e+002 -1.6000e+001i
```

rank(A)

Rank of matrix A

```
>> rank(A)
ans =
3
```

trace(A)

Sum of the diagonal elements of A

```
>> trace(A)
ans =
3.0000 - 1.0000i
```

svd(A)

Vector V of singular values of A, which are the square roots of the eigenvalues of the symmetric matrix A'A

```
>> svd(A)
ans =
9.7269
6.6228
2.7434
```

[U,S,V]=svd(A)

*Give the diagonal matrix S of singular values of A (ordered from largest to smallest) and the matrices U and V such that A=U*S*V'.*

```
>> [U,S,V]=svd(A)
U =
-0.1891 - 0.4193i -0.6966 - 0.5354i -0.0595 + 0.1139i
-0.5350 - 0.5583i 0.3725 + 0.1937i -0.4752 - 0.0029i
-0.2989 - 0.3181i 0.2210 - 0.0543i 0.8621 - 0.1204i
S =
9.7269 0 0
0 6.6228 0
0 0 2.7434
V =
-0.1301 0.2346 0.9634
```

	-0.1094 - 0.3895i -0.7321 - 0.5179i 0.1635 + 0.0735i -0.6018 - 0.6762i 0.3731 + 0.0396i -0.1721 - 0.1010i
rcond(A)	*Reciprocal of the matrix condition A* >> rcond(A) ans = 0.1796
norm(A)	*Norm of A or 2-norm (largest singular value of matrix A)* >> norm(A) ans = 9.7269
norm(A,1)	*1-norm of A (largest sum of the columns of A)* >> norm(A,1) ans = 14
norm(A,inf)	*Infinite norm of A (largest sum of the rows of A)* >> norm(A,inf) ans = 10
norm(A,'fro')	*F-norm of A, defined by sqrt(sum(sum(diag(A'A))))* >> norm(A,'fro') ans = 12.0830

cond(A) or cond(A,2)
cond(A,1)
cond(A,inf)
cond(A,fro)
	Gives the condition of the matrix A (quotient between the largest and the smallest singular value of A). Condition according to the 2-norm
	Condition of A according to the 1-norm

Condition of A according to the infinite norm

Condition of A according to the F-norm (Frobenius norm)

```
>> cond(A)
ans =
3.5456
>> cond(A,1)
ans =
5.5676
>> cond(A,inf)
ans =
5.1481
>> cond(A,'fro')
ans =
4.9266
```

Z=null(A)

It gives an orthonormal basis of the kernel of A (Z'Z=I). The

number of columns of Z is the null of A

```
>> null([1 2 3;4 5 6;7 8 9])
ans =
-0.4082
0.8165
-0.4082
```

Z=null(A, 'r')

Gives a rational basis for the kernel of A

```
>> null([1 2 3;4 5 6;7 8 9],'r')
ans =
1
-2
1
```

Q=orth(A)

(Q'Q=I).

Gives an orthonormal basis for the rank of A. The columns of Q generate the same space *as the columns of A, and the number of columns of Q is the rank of A.*

```
>> orth([1 2 3;4 5 6;7 8 9])
ans =
-0.2148 0.8872
-0.5206 0.2496
-0.8263 -0.3879
```

subspace(A,B)

Give the angle between the subspaces specified by the columns of A and B. If A and B are vectors, give the angle formed by both of them

```
>> subspace(A,B)
ans =
1.1728e-015
```

rref(A)

Gives the reduced Gauss-Jordan stepwise row-wise matrix of A. The number of non-zero rows of rref(A) is the rank of A.

```
>> rref([1 2 3;4 5 6;7 8 9])
ans =
1 0 -1
0 1 2
0 0 0
```

A^p

Matrix A raised to scalar power p

```
>> A^3
ans =
1.0e+002 *
1.8600 - 0.1200i 1.0400 + 0.5400i 2.2200 - 0.4200i
0.6400 - 0.1000i 2.1400 - 0.2000i 2.3400 + 0.3000i
0.8200 + 0.2400i 0.9200 - 0.1800i 3.3800 - 0.0600i
```

p^A

Scalar p raised to the matrix A

```
>> 3^A
ans =
1.0e+003 *
```

```
1.2568 + 0.1484i 1.5585 - 0.0742i 3.7457 - 0.0742i
1.2568 + 0.1484i 1.5586 - 0.0742i 3.7456 - 0.0742i
1.2568 + 0.1485i 1.5586 - 0.0742i 3.7456 - 0.0742i
```

diag(V,k)

Construct a diagonal square matrix of order n+|k| with the n elements of the vector V on the kth diagonal. If k=0 the diagonal is the principal, if k>0 the diagonal is above the principal k units. If k<0 the diagonal is below the principal k units. Furthermore diag(V,0)=diag(V)

```
    >> diag([1 2 3 3 4 5 6])
    ans =



1 0 0 0 0 0
    0 2 0 0 0 0
    0 0 3 0 0 0
    0 0 0 4 0 0
    0 0 0 0 5 0
    0 0 0 0 0 6
    >> diag([1 2 3 4 5 6],1)
    ans =
    0 1 0 0 0 0 0
    0 0 2 0 0 0 0
    0 0 0 3 0 0 0
    0 0 0 0 4 0 0
    0 0 0 0 0 5 0
    0 0 0 0 0 0 6
    0 0 0 0 0 0 0
    >> diag([1 2 3 4 5 6],-2)
    ans =
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    1 0 0 0 0 0 0 0
    0 2 0 0 0 0 0 0
```

```
0 0 3 0 0 0 0 0
0 0 0 4 0 0 0 0
0 0 0 0 5 0 0 0
0 0 0 0 0 6 0 0
```

triu(A,k)

Construct an upper triangular matrix with the elements of A that are above its kth diagonal. If k=0 the diagonal is the principal, if k>0 the diagonal is above the principal k units. If k<0 the diagonal is below the principal k units. Moreover triu(A,0)=triu(V)

```
>> triu([1 2 3 ;4 5 6; 7 8 9],1)
ans =
0 2 3
0 0 6
0 0 0
>> triu([1 2 3 ;4 5 6; 7 8 9],1)
ans =
0 2 3
0 0 6
0 0 0
>> triu([1 2 3 ;4 5 6; 7 8 9],-1)
ans =
1 2 3
4 5 6
0 8 9
```

tril(A,k)

Construct a lower triangular matrix with the elements of A that are below its kth diagonal. If k=0 the diagonal is the principal, if k>0 the diagonal is above the principal k units. If k<0 the diagonal is below the principal k units. Furthermore tril(A,0)=tril(V)

```
>> tril([1 2 3 ;4 5 6; 7 8 9])
ans =
1 0 0
```

```
4 5 0
7 8 9
>> tril([1 2 3 ;4 5 6; 7 8 9],1)
ans =
1 2 0
4 5 6
7 8 9
>> tril([1 2 3 ;4 5 6; 7 8 9],-1)
ans =
0 0 0
4 0 0
7 8 0
```

A=sym('[f1;f2,...,fm]') *Define the symbolic matrix mxn of rows f1 to fn where fi=[ai1,ai2,...,ain]. The operations and functions for symbolic matrices are similar to those already seen for numerical matrices.*

```
>> A=sym('[a,a+1,a+2;a,a-1,a-2;1,2,3]')
A =
[ a, a + 1, a + 2].
[a, a - 1, a - 2]
[ 1, 2, 3]
>> B=sym('[b,b+1,b+2;b,b-1,b-2;1,2,3]')
B =
[ b, b + 1, b + 2].
[ b, b - 1, b - 2].
[ 1, 2, 3]
>> A+B
ans =
[ a + b, a + b + 2, a + b + 4].
[ a + b, a + b - 2, a + b - 4].
[ 2, 4, 6]
>> det(A)
ans =
0
>> rank(A)
ans =
2
>> trace(A)
```

```
ans =
2*a + 2
>> triu(A,2)
ans =
[ 0, 0, a + 2]
[ 0, 0, 0]
[ 0, 0, 0]
```

Scalar product of vectors A and B

dot(A,B)

```
>> a = [1 2 3]; b = [4 5 6];
c = dot(a,b)
c =
32
```

Vector product of vectors A and B

cross(A,B)

```
>> a = [1 2 3]; b = [4 5 6];
c = cross(a,b)
c =
-3 6 -3
```

5.2 DECOMPOSITION OF MATRICES. EIGENVALUES AND EIGENVECTORS

M ATLAB allows you to work with special care in the field of matrix decomposition. It implements commands for most of the known decompositions and for working with eigenvalues and eigenvectors. The syntax of the most common commands in this field is presented in the following table.

eig(A)

Find the eigenvalues of the square matrix A

```
>> A=rand(4)
A =
0.8147 0.6324 0.9575 0.9572
0.9058 0.0975 0.9649 0.4854
0.1270 0.2785 0.1576 0.8003
0.9134 0.5469 0.9706 0.1419
>> B=sym('[1, a, a^2;1, a, -a;0, 0, 1]')
B =
[ 1, a, a^2]
[ 1, a, -a]
[ 0, 0, 1]
>> eig(A)
ans =
2.4021
-0.0346
-0.7158
-0.4400
>> eig(B)
ans =
0
1
a + 1
```

[V,D] = eig(A)

Find the diagonal matrix D of eigenvalues of A and a matrix V whose columns are the *corresponding eigenvectors, where A*V=V*D*

```
   >> [V,D]=eig(A)
   V =
   -0.6621 -0.7149 0.1745 0.1821
   -0.4819 -0.0292 -0.6291 -0.9288
   -0.2766 0.6972 0.5995 0.3178
   -0.5029 -0.0438 -0.4630 0.0570



D =
   2.4021 0 0 0
   0 -0.0346 0 0
   0 0 -0.7158 0
   0 0 0 -0.4400
   >> [V1,D1]=eig(B)
   V1 =
   [ -a, a^2, 1]
   [ 1, -a, 1]
   [ 0, 1, 0]



D 1 =
   [ 0, 0, 0]
   [ 0, 1, 0]
   [ 0, 0, a + 1]
```

eig(A,B) *Returns a vector with the generalised eigenvalues of the square matrices A and B. The generalised eigenvalues of A and B are the roots of the polynomial in det(λλ*C - A).*

```
   >> C=randn(4)



C =
   -0.1241 0.6715 0.4889 0.2939
```

```
1.4897 -1.2075 1.0347 -0.7873
1.4090 0.7172 0.7269 0.8884
1.4172 1.6302 -0.3034 -1.1471
>> eig(A,C)
ans =
0.6654 + 0.6533i
0.6654 - 0.6533i
0.0259
0.1997
```

[V,D] = eig(A,B)

*Find the diagonal matrix D of generalised eigenvalues of A and B and a matrix V whose columns are the corresponding eigenvectors, satisfying A*V=B*V*D*

```
>> [V2,D2]=eig(A,C)
V2 =
0.2665 - 0.0340i 0.2665 + 0.0340i 1.0000 0.4002
-0.3908 + 0.0213i -0.3908 - 0.0213i 0.0953 -1.0000
-0.8463 - 0.1537i -0.8463 + 0.1537i -0.9489 -0.1449
0.6062 - 0.2931i 0.6062 + 0.2931i 0.0211 0.3193



D 2 =
0.6654 + 0.6533i 0 0 0
0 0.6654 - 0.6533i 0 0
0 0 0.0259 0
0 0 0 0.1997
```

[AA, BB, Q, Z, V] [AA, BB, Q, Z, V]. = qz(A,B)

*Calculate the upper triangular matrices AA and BB and the matrices Q and Z such that Q*A*Z = AA and Q*B*Z = BB, and give the matrix V of generalised eigenvectors of A and B. The generalised eigenvalues are the elements of the diagonal of AA and BB, so that one has the following equality:*

$$A*V*diag(BB) = B*V*diag(AA)$$

```
>> [AA, CC, Q, Z, V]= qz(A,C)
AA =
0.6430 - 0.6312i 0.9129 - 1.0466i 1.2477 - 0.7149i -0.1102 + 0.2576i
0 0.9818 + 0.9639i -0.1338 + 0.8139i 0.2874 + 0.0652i
0 0 0.0482 0.0822
0 0 0 0.4374
```

```
C C =
0.9663 0.1979 + 0.6630i 1.5008 - 0.7681i -0.2960 + 0.0079i
0 1.4756 -0.0309 + 1.0436i -0.1974 - 0.3071i
0 0 1.8638 0.1951
0 0 0 2.1900
```

```
Q =
0.4727 - 0.0698i 0.3875 - 0.1637i 0.1251 + 0.3290i 0.4287 - 0.5358i
0.2342 + 0.4500i 0.0410 + 0.2938i 0.5545 + 0.3640i -0.4572 + 0.0751i
-0.5656 -0.0713 0.6541 0.4972
0.4410 -0.8544 0.0909 0.2594
Z =
-0.1939 - 0.1152i 0.0800 + 0.0321i 0.8930 -0.3798
0.2939 + 0.1469i -0.0424 - 0.0841i 0.4423 0.8292
0.7032 + 0.1644i 0.4323 - 0.4410i 0.0188 -0.3110
-0.3690 - 0.4283i 0.7481 - 0.2067i -0.0812 0.2673
V =
-0.2235 - 0.1328i 0.1328 + 0.2235i 1.0000 + 0.0000i -0.4002 - 0.0000i
0.3388 + 0.1694i -0.1694 - 0.3388i 0.0953 - 0.0000i 1.0000 + 0.0000i
0.8106 + 0.1894i -0.1894 - 0.8106i -0.9489 - 0.0000i 0.1449 + 0.0000i
-0.4253 - 0.4937i 0.4937 + 0.4253i 0.0211 - 0.0000i -0.3193 + 0.0000i
```

[T,B] = balance(A) _Find a transformation matrix T such that B = T\A*T has eigenvalues approximating those of A. Matrix B is called the balanced matrix of matrix A._

```
>> [T,B2]=balance(A)
T =
1.0000 0 0 0
0 1.0000 0 0
0 0 0.5000 0
0 0 0 1.0000
B2 =
0.8147 0.6324 0.4788 0.9572
0.9058 0.0975 0.4824 0.4854
0.2540 0.5570 0.1576 1.6006
0.9134 0.5469 0.4853 0.1419
```

balance(A)

Calculates the balanced matrix B of the matrix A. Its essential use is to approximate the eigenvalues of A when they are difficult to compute. It follows that eig(A)=eig(balance(A))

```
>> balance(A)
ans =
0.8147 0.6324 0.4788 0.9572
0.9058 0.0975 0.4824 0.4854
0.2540 0.5570 0.1576 1.6006
0.9134 0.5469 0.4853 0.1419
```

[V,D] = cdf2rdf(V,D)

Transforms the complex output [V,D] of the eig command to real form. Each complex eigenvalue on the diagonal of D gives rise to a 2x2 submatrix in the real form of the matrix D

```
>> [V,D]=cdf2rdf(A,C)
V =
0.8147 0.6324 0.9575 0.9572
0.9058 0.0975 0.9649 0.4854
0.1270 0.2785 0.1576 0.8003
0.9134 0.5469 0.9706 0.1419
D =
-0.1241 0.6715 0.4889 0.2939
1.4897 -1.2075 1.0347 -0.7873
1.4090 0.7172 0.7269 0.8884
1.4172 1.6302 -0.3034 -1.1471
```

[U,T] = schur(A)

*Find a matrix T and a unitary matrix U such that A = U*T*U' and U'*U = eye(U). If A is complex, T is an upper triangular matrix with the eigenvalues of A on the diagonal. If A is real, the* *matrix T has the eigenvalues of A on the diagonal, and the corresponding complex eigenvalues correspond to the 2x2 diagonal blocks of the matrix T*

```
>> [U,T] = schur(A)
U =
-0.6621 -0.5327 0.5206 -0.0825
-0.4819 0.1301 -0.5815 -0.6424
-0.2766 0.8273 0.4848 -0.0637
-0.5029 0.1217 -0.3947 0.7593
T =
2.4021 -0.8133 -0.6225 -0.1304
0 -0.0346 -0.1940 0.2110
0 0 -0.7158 0.3496
0 0 0 -0.4400
```

schur(A)

Returns the matrix T from the previous section only

```
>> schur(A)
ans =
2.4021 -0.8133 -0.6225 -0.1304
0 -0.0346 -0.1940 0.2110
0 0 -0.7158 0.3496
0 0 0 -0.4400
```

[U,T] = rsf2csf(U,T)

Converts the output [U,T] of the schur command to complex.

```
>> [U1,T1] = rsf2csf(A,C)
U1 =
0.0394 + 0.2165i -0.8561 + 0.0945i -0.9360 1.1092
0.5415 + 0.2407i -0.7067 - 0.0479i -0.8523 0.6459
-0.1702 + 0.0337i -0.2203 + 0.0689i -0.1028 0.8155
0.1937 + 0.2427i -0.8861 + 0.0600i -1.0560 0.3086
T1 =
-0.8576 + 0.4577i -1.7102 + 0.4507i 0.1154 + 0.1647i 0.9728 - 0.0995i
0 -0.8576 - 0.4577i 0.7230 + 0.1166i 0.0766 + 0.2499i
1.4090 0 0.9535 -0.9437
```

```
1.4172 1.6302 0 -0.9901
```

[P,H] = hess(A)

*Returns the unitary matrix P and the Hessenberg matrix H such that A = P*H*P' and P'*P =eye(size(P))*

```
>> [P,H] = hess(A)
P =
1.0000 0 0 0
0 -0.7007 0.1180 -0.7036
0 -0.0982 -0.9928 -0.0687
0 -0.7066 0.0210 0.7073
```

```
H =
0.8147 -1.2135 -0.8559 0.1663
-1.2926 0.8399 1.2997 0.0668
0 0.6987 -0.0233 -0.3394
0 0 0.0323 -0.4196
```

[L,U] = lu(A)

*Decomposes the matrix A into the product A=L*U (LU decomposition of A), where U is an upper triangular matrix and L is a lower pseudo-triangular matrix (triangularisable by permutation).*

```
>> [L,U] = lu(A)
L =
0.8920 -0.3250 1.0000 0
0.9917 1.0000 0 0
0.1390 -0.4552 0.2567 1.0000
1.0000 0 0 0
```

```
U =
0.9134 0.5469 0.9706 0.1419
0 -0.4448 0.0024 0.3447
```

```
    0 0 0.0925 0.9426
    0 0 0 0.6955
```

[L,U,P] = lu(A)

*Gives a lower triangular matrix L, an upper triangular matrix U and a permutation matrix P such that P*A=L*U*

```
    >> [L,U,P] = lu(A)
    L =
    1.0000 0 0 0
    0.9917 1.0000 0 0
    0.8920 -0.3250 1.0000 0
    0.1390 -0.4552 0.2567 1.0000


U =
    0.9134 0.5469 0.9706 0.1419
    0 -0.4448 0.0024 0.3447
    0 0 0.0925 0.9426
    0 0 0 0.6955


P =
    0 0 0 1
    0 1 0 0
    1 0 0 0
    0 0 1 0
```

R = chol(A)

*Returns the upper triangular matrix R such that R'*R=A (Cholesky decomposition of A), in case A is positive definite. If A is not positive definite, returns an error.*

[Q,R] = qr(A)

*Returns the upper triangular matrix R of the same dimension as A, and the orthogonal matrix Q such that A=Q*R (QR decomposition of A).*

This decomposition can be applied to non-square matrices

```
   >> [Q,R] = qr(A)
   Q =
   0.5332 0.4892 0.6519 -0.2267
   0.5928 -0.7162 0.1668 0.3284
   0.0831 0.4507 -0.0991 0.8833
   0.5978 0.2112 -0.7331 -0.2462



R =
   1.5279 0.7451 1.6759 0.9494
   0 0.4805 0.0534 0.5113
   0 0 0.0580 0.5216
   0 0 0 0.6143
```

[Q,R,E] = qr(A)

*Returns the upper triangular matrix R of the same dimension as A, the permutation matrix E and the orthogonal matrix Q such that A*E=Q*R*

```
>> [Q,R,E] = qr(A)
Q =
0.5707 0.4307 0.2564 -0.6504
0.5751 -0.0867 -0.8029 0.1308
0.0939 0.7694 0.0861 0.6259
0.5785 -0.4636 0.5313 0.4100
R =
1.6777 0.9827 0.7595 1.5263
0 0.9201 0.2246 -0.0534
0 0 0.3983 -0.0222
0 0 0 0.0425
E =
0 0 0 1
0 0 1 0
1 0 0 0
0 1 0 0
```

jordan(A)

Find the canonical Jordan matrix J of the

[V,J]=jordan(A)

matrix A (J has the eigenvalues of A on the diagonal).

*Find the canonical Jordan matrix J of the matrix A and the step matrix V whose columns are the eigenvectors of A with V^{-1*} A*V=J.*

```
>> jordan(A)
ans =
-0.0346 + 0.0000i 0 0 0
0 2.4021 - 0.0000i 0 0
0 0 -0.4400 + 0.0000i 0
0 0 0 -0.7158 - 0.0000i
>> [V,J]=jordan(A)
V =
16.3153 + 0.0000i 1.3164 + 0.0000i 3.1933 + 0.0000i -0.3768 - 0.0000i
0.6653 + 0.0000i 0.9583 + 0.0000i -16.2888 - 0.0000i 1.3588 + 0.0000i
-15.9101 - 0.0000i 0.5499 - 0.0000i 5.5734 - 0.0000i -1.2947 - 0.0000i
1.0000 1.0000 1.0000 1.0000

J =
-0.0346 + 0.0000i 0 0 0
0 2.4021 - 0.0000i 0 0
0 0 -0.4400 + 0.0000i 0
0 0 0 -0.7158 - 0.0000i
```

condeig(A)

Vector with the condition numbers for the eigenvalues of A

```
>> condeig(A)
ans =
1.0733
1.1338
1.7360
1.6539
```

[V,D,s]=condeig(A)

Equivalent to [V,D]=eig(A) and s=condeig(A)

```
>> [V,D,s]=condeig(A)
V =
-0.6621 -0.7149 0.1745 0.1821
-0.4819 -0.0292 -0.6291 -0.9288
-0.2766 0.6972 0.5995 0.3178
```

```
-0.5029 -0.0438 -0.4630 0.0570
D =
2.4021 0 0 0
0 -0.0346 0 0
0 0 -0.7158 0
0 0 0 -0.4400
s =
1.0733
1.1338
1.7360
1.6539
```

[U,V,X,C,S]= gsvd(A,B)

*Give the unitary matrices U and V, the square matrix X and the non-negative diagonal matrices C and S such that A = U*C*X', B = V*S*X' and C'*C + S'*S = I. A is (m,p), B is (n,p), U is (M,M), V is (n,n), X is (p,q) and q=min(m+n,p).*

[U,V,X,C,S]= gsvd(A,B,0)

Result above when m o n ≥p. U and V have at most p columns and C and S have at most p rows.

sigma = gsvd(A,B)

*Returns the vector of generalised singular values sqrt(diag(C'*C)./diag(S'*S))*

X = pinv(A)

*Returns the matrix X (pseudoinverse of A), of the same dimension as A' such that A*X*A=A and X*A*X=X, where A*X and X*A are Hermitian matrices.*

```
>> X = pinv(A)
X =
-15.2997 3.0761 14.7235 9.6445
-0.2088 -1.8442 1.0366 1.8711
14.5694 -1.9337 -14.6497 -9.0413
-0.3690 0.5345 1.4378 -0.4008
```

hess(A)

Returns the Hessenberg matrix H

```
>> hess(A)
```

```
ans =
0.8147 -1.2135 -0.8559 0.1663
-1.2926 0.8399 1.2997 0.0668
0 0.6987 -0.0233 -0.3394
0 0 0.0323 -0.4196
```

poly(A)

Returns the characteristic polynomial of the matrix A

```
>> poly(A)
ans =
1.0000 -1.2118 -2.5045 -0.8416 -0.0261
```

poly(V)

Returns a vector whose components are the coefficients of the polynomial whose roots are the elements of the vector V

```
>> poly([1 2 3 4 5 6])
ans =
1 -21 175 -735 1624 -1764 720
```

vander(C)

Returns the Vandermonde matrix A such that its j-th column is the vector C. Furthermore A(:,j) = C $^{n-j}$

```
>> vander([1 -1 0 6 6 3])
ans =
1 1 1 1 1
1 -1 1 -1 1
0 0 0 0 1
1296 216 36 6 1
81 27 9 3 1
```

eigs(A)

Vector with the six largest eigenvalues of A

eigs(A,B)

*Solve the equation A*V == B*V*D with symmetrical B*

eigs(A,k)

Vector with the k largest eigenvalues of A

eigs(A,B,k)

*Returns the k largest solutions of A*V == B*V*D*

eigs(A,k, σ)

e igs(A,B,k,σ)

Vector with the k largest eigenvalues of A based on σ, where σit can be worth 'lm' (largest magnitude), 'sm' (smallest magnitude), 'lr' (largest real part), 'sr' (smallest real part), 'li' (largest imaginary part) and 'si' (smallest imaginary part).

[V,D] = eigs(A,...)

*Gives the k largest solutions of A*V == B*V*D according to σ*

Gives the diagonal matrix D of eigenvalues of A and the matrix V whose columns are the respective eigenvectors

```
>> [V,D] = eigs(A)
V =
-0.6621 0.1745 0.1821 -0.7149
-0.4819 -0.6291 -0.9288 -0.0292
-0.2766 0.5995 0.3178 0.6972
-0.5029 -0.4630 0.0570 -0.0438



D =
2.4021 0 0 0
0 -0.7158 0 0
0 0 -0.4400 0
0 0 0 -0.0346
```

svds(A) *Gives the 5 largest singular values of A*

svds(A,k) *Gives the k largest singular values of A*

svds(A,k,0) *Gives the k largest singular values of A using eigs*

[U,S,V] = svds(A,...)

Gives the matrices U(m,k) with orthonormal columns, S(k,k) diagonal and V(n,k) with *orthonormal columns*

```
   >> [U,S,V] = svds(A)
   U =
   -0.6380 0.3590 0.2155 0.6462
   -0.5202 -0.2451 -0.8111 -0.1069
   -0.2287 0.7358 0.0083 -0.6373
   -0.5197 -0.5193 0.5436 -0.4059



S =
   2.6201 0 0 0
   0 0.8590 0 0
   0 0 0.3796 0
   0 0 0 0.0306



V =
   -0.5705 -0.3613 -0.1621 -0.7196
   -0.3061 0.1444 0.9401 -0.0416
   -0.6310 -0.3268 -0.1247 0.6924
   -0.4274 0.8613 -0.2729 -0.0321
```

H=hadamard(n)

*Matrix with values 1 or -1 such that H'*H = n*eye(n)*

```
   >> H=hadamard(4)
   H =
   1 1 1 1
   1 -1 1 -1
   1 1 -1 -1
   1 -1 -1 1
```

hankel(V)

Matrix whose first column is the vector V and whose elements are zero below the first antidiagonal. The matrix Hankel(C,R) has as its *first column the vector C and as its last row the vector R*

```
>> hankel([1 2 3 4 5])
ans =
1 2 3 4 5
2 3 4 5 0
3 4 5 0 0
4 5 0 0 0
5 0 0 0 0
```

hilb(n)

Hilbert matrix of order n such that Aij=1/(i+j-1)

```
>> hilb(4)
ans =
1.0000 0.5000 0.3333 0.2500
0.5000 0.3333 0.2500 0.2000
0.3333 0.2500 0.2000 0.1667
0.2500 0.2000 0.1667 0.1429
```

invhilb(n)

Inverse of the Hilbert matrix of order n

```
>> invhilb(4)
ans =
16 -120 240 -140
-120 1200 -2700 1680
240 -2700 6480 -4200
-140 1680 -4200 2800
```

magic(n)

Magic matrix of order n. Its elements are the integers from 1 to n ^{2} with equal sums of rows and columns.

```
>> magic(4)
ans =
16 2 3 13
5 11 10 8
9 7 6 12
4 14 15 1
```

pascal(n)

Pascal matrix of order n (symmetric, positive *definite and based on Pascal's triangle)*

```
>> pascal(4)
ans =
1 1 1 1
1 2 3 4
1 3 6 10
1 4 10 20
```

rosser

8x8 matrix with one double eigenvalue, one null, three almost equal, one very small eigenvalue and two opposites

```
>> rosser
ans =
611 196 -192 407 -8 -52 -49 29
196 899 113 -192 -71 -43 -8 -44
-192 113 899 196 61 49 8 52
407 -192 196 611 8 44 59 -23
-8 -71 61 8 411 -599 208 208
-52 -43 49 44 -599 411 208 208
-49 -8 8 59 208 208 99 -911
29 -44 52 -23 208 208 -911 99
```

toeplitz(C,R)

Toeplitz matrix (non-symmetric with vector C as first column and vector R as first row)

```
>> toeplitz([1 2 3 4 ],[1 6 7 8])
ans =
1 6 7 8
2 1 6 7
3 2 1 6
4 3 2 1
```

wilkinson(n)

Wilkinson matrix (tridiagonally symmetric with close but not equal eigenvalue pairs)

```
>> wilkinson(5)
ans =
2 1 0 0 0
1 1 1 0 0
0 1 0 1 0
```

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

compan(P)

Matrix of the polynomial of coefficients P

```
>> compan([1 2 3 4 5])
ans =
-2 -3 -4 -5
1 0 0 0
0 1 0 0
0 0 1 0
```

5.3 VECTOR SPACES, LINEAR APPLICATIONS AND QUADRATIC FORMS

T he matrix commands presented above allow you to work with vector spaces, linear applications and quadratic forms. It will be possible to work with linear dependence and independence, bases, changes of basis and vector geometry in two and three dimensions. These statements will be illustrated by means of examples.

As a first example, we check whether the vectors {{1,2,-3,4}, {3,-1,2,1}, {1,-5,8,-7}, {2,3,1,-1}} are linearly independent.

```
>> A=[1,2,-3,4;3,-1,2,1;1,-5,8,-7;2,3,1,-1]
A =
1 2 -3 4
3 -1 2 1
1 -5 8 -7
2 3 1 -1
>> det(A)
ans =
0
```

Since the determinant of the matrix of their coordinates is zero, the vectors are linearly independent.

As a second example, let us check whether the set of vectors of R ^{4} {{1,2,2,1},{3,4,4,3},{1,0,0,1}} are linearly independent.

```
>> B=[1,2,2,1;3,4,4,3;1,0,0,1]
B =
1 2 2 1
3 4 4 3
1 0 0 1
```

```
>> rank(B)
ans =
2
```

We have three vectors in R^4, they would be independent if the rank of the matrix of their coordinates is 3. But we have seen that this rank is 2, which indicates that the vectors are linearly dependent.

As a third example we are going to obtain the dimension and a basis of the linear variety generated by the vectors {{2,3,4,-1,1},{3,4,7,-2,-1},{1,3,-1,1,8},{0,5,5,-1,4}}.

To find the dimension of the linear variety we calculate the rank of the matrix formed by the vectors that generate it. This rank will be the dimension of the linear variety.

```
>> A=[2,3,4,-1,1;3,4,7,-2,-1;1,3,-1,1,8;0,5,5,-1,4]
A =
2 3 4 -1 1
3 4 7 -2 -1
1 3 -1 1 8
0 5 5 -1 4
>> rank(A)
ans =
3
```

We have then that the dimension of the linear variety is 3 and a basis will be formed by the vectors containing the components of any non-zero minor of order 3 of the matrix A.

```
>> det([3 4 7; 1 3 -1;0 5 5 5])
ans =
75
```

Then a basis of the linear variety will be formed by the vectors {{3,4,7,-2,-1}, {1,3,-1,1,8},{0,5,5,5,-1,4}}

As a fourth example, we will check whether the vectors {{2,3,-1},{0,0,1},{2,1,0}} form a basis in R ^{3} and if so, we will obtain the components of the vector {3,5,1} in the aforementioned basis.

The given vectors are 3 vectors of a 3-dimensional space, so the condition for them to be a basis is that their determinant is zero.

```
>> det([2,3,-1;0,0,1;2,1,0])

ans =

4
```

The vectors form a basis. To find the components of the vector {3,5,1} in this basis, the following operations are carried out:

```
>> inv([2,0,2;3,0,1;-1,1,0])*[3,5,1]'

ans =

1.7500

2.7500

-0.2500
```

As a fifth example we consider the bases of R ^{3} defined as B={{1,0,0},{-1,1,0},{0,1,-1}} and B1={{{1,0,-1},{2,1,0},{-1,1,1}} and find the change of basis matrix from B to B1 by additionally calculating the components of the vector {2,1,3} in basis B in basis B1.

The operation to be carried out would be as follows:

```
>> B=[1,0,0;-1,1,0;0,1,-1]

B =

1 0 0

-1 1 0

0 1 -1

>> B1=[1,0,-1;2,1,0;-1,1,1]

B1 =
1 0 -1
   2 1 0
   -1 1 1
   >> A=inv(B1')*B'
   A =
   -0.5000 1.5000 2.5000
   0.5000 -0.5000 -0.5000
   -0.5000 1.5000 1.5000
```

To find the components of the vector {2,1,3} in base B in base B1 from the change of base matrix A, we perform the following operation:

```
   >> inv(B1')*B'*[2,1,3]'
   ans =
   8
   -1
   5
```

As a sixth example we find the mixed product of the vectors {{1,1,2},{0,1,0},{0,1,1}} and calculate the area of the triangle whose vertices have as coordinates the points (0,0), (5,1) and (3,7).

```
   >> dot([1,1,2],cross([0,1,0],[0,1,1]))
   ans =
   1
   >> (1/2)*det([0,0,1;5,1,1;3,7,1])
   ans =
   16
```

As a seventh example we consider a linear application of R ^{5} in R ^{3} whose matrix in the canonical bases is the following:

Imagen en blanco y negro Descripción generada automáticamente cc
confianza media

The aim is to find a basis for its kernel and the dimension of this kernel, as well as the image of the vectors {4,2,0,0,0,-6} and {1,2,-1,-2,3} by means of this application. Find also a basis for the image of *f* and its dimension.

```
>> A=[0,-3,-1,-3,-1;-3,3,-3,-3,-1;2,2,-1,1,2]
A =
0 -3 -1 -3 -1
-3 3 -3 -3 -1
2 2 -1 1 2
>> null(A)
ans =
-0.5397 -0.1251
-0.2787 -0.2942
-0.0266 -0.6948
0.0230 0.6021
0.7936 -0.2292
```

The two vectors in the above output form the basis of the kernel and therefore the dimension of the kernel is 2.

To find the image of any vector X by means of the linear application f we assume that f(X) = A*X. We have:

```
>> A*[4 2 0 0 -6]'
ans =
0
0
0
>> A*[1 2 -1 -2 3]'
ans =
-2
9
11
```

The dimension of the image of *f* coincides with the rank of its matrix.

```
>> rank(A)
ans =
3
```

The dimension of the image of *f* is 3 and a basis will consist of three columns contained in the matrix A that are linearly independent.

```
>> det([0 -3 -2;-3 3 2; -1 -3 -1])
ans =
-9
```

Therefore, the vectors {{0 -3 -2}; {-3 3 2}; {-1 -3 -1}} form a basis of the image of *f* .

As an eighth example, we consider the linear application *f* between two vector subspaces U and V of three-dimensional real space, such that f(*a,b,c*) = (*a+b,b+c,a+c*), where (*a,b,c*) is any point of U. We need to find the matrix associated with the applications *f* , f^5 , and e^f .

To find the matrix of *f* , we must consider the vectors transformed by *f* from those of the canonical basis:

```
>> f
f =
[ a + b, b + c, a + c].
>> subs(f,{a,b,c},{1,0,0})
ans =
1 0 1
>> subs(f,{a,b,c},{0,1,0})
ans =
1 1 0
>> subs(f,{a,b,c},{0,0,1})
ans =
0 1 1
```

The matrix associated to the linear application *f* will have as columns the above vectors transformed from the canonical basis of R^3 . It will be then:

Imagen en blanco y negro Descripción generada automáticamente con confianza baja

T he matrix associated to f^5 shall be A^5 and the matrix associated to e^f shall be e^A.

```
>> A=([1 0 1;1 1 0;0 1 1])'
A =
1 1 0
0 1 1
1 0 1
>> A^5
ans =
11 10 11
11 11 10
10 11 11
>> expm(A)
ans =
3.1751 2.8321 1.3819
1.3819 3.1751 2.8321
2.8321 1.3819 3.1751
```

As a ninth example we classify the bilinear form f:UxV→R and the quadratic form g:U→R defined as follows:

Texto Descripción generada automáticamente con confianza media

```
>> A=[1,-2,0;0,0,4;-1,0,-3]
A =
1 -2 0
0 0 4
-1 0 -3
>> det(A)
ans =
8
```

Since the determinant of the matrix of *f* is non-zero, the bilinear form is regular non-degenerate.

```
>> B=[1,-1,3;-1,1,-3/2;3,-3/2,4]
B =
1.0000 -1.0000 3.0000
-1.0000 1.0000 -1.5000
3.0000 -1.5000 4.0000
```

To classify the quadratic form we calculate its diagonal determinants.

```
>> det(B)
ans =
-2.2500
>> det([1,-1;-1,1])
ans =
0
```

It turns out that the quadratic form is negative semidefinite.

But classification can also be done through the eigenvalues of the matrix of the quadratic form.

A quadratic form is positive definite if, and only if, all its eigenvalues are strictly positive. A quadratic form is defined negative if, and only if, all its eigenvalues are strictly negative.

A quadratic form is positive semidefinite if, and only if, all its eigenvalues are non-negative. A quadratic form is negative semidefinite if, and only if, all its eigenvalues are non-positive.

A quadratic form is indefinite if there are positive and negative eigenvalues.

```
>> eig(B)
ans =
-0.8569
0.4071
6.4498
```

There are positive and negative eigenvalues, so the quadratic form is undefined.

Exercise 1. Consider the following matrix:

Texto Descripción generada automáticamente

Find its transpose, its inverse, its determinant, its rank, its trace, its singular values, its condition, its norm, M ^{3} , e ^{M} , log(M) and sqrt(M):

```
" M=[1/3,1/4,1/5; 1/4,1/5,1/6;1/5,1/6,1/7]
M =
0.3333 0.2500 0.2000
0.2500 0.2000 0.1667
0.2000 0.1667 0.1429
"transposed=M'
transposed =
0.3333 0.2500 0.2000
0.2500 0.2000 0.1667
0.2000 0.1667 0.1429
"inverse=inv(M)
inverse =
1.0e+003 *
0.3000 -0.9000 0.6300
-0.9000 2.8800 -2.1000
0.6300 -2.1000 1.5750
```

To check that the inverse is correctly calculated, we multiply it by M and the result must be the identity matrix of order 3:

```
"M*inv(M)
ans =
1.0000 0.0000 0.0000
0.0000 1.0000 0.0000
0.0000 0.0000 1.0000
"determinant=det(M)
```

```
determinant =
2.6455e-006
"rank=rank(M)
rank =
3
"trace=trace(M)
trace =
0.6762
"vsingulars=svd(M)
vsingulars =
0.6571
0.0189
0.0002
"condition=cond(M)
condition =
3.0886e+003
```

To calculate the norm, we find the norm, the 1-norm, the infinite norm and the F-norm:

```
"norm(M)
ans =
0.6571
"norm(M,1)
ans =
0.7833
"norm(M,inf)
ans =
0.7833
"norm(M,'fro')
ans =
0.6573
" M^3
ans =
0.1403 0.1096 0.0901
```

```
0.1096 0.0856 0.0704
0.0901 0.0704 0.0578
"logm(M)
ans =
-2.4766 2.2200 0.5021
2.2200 -5.6421 2.8954
0.5021 2.8954 -4.7240
"sqrtm(M)
ans =
0.4631 0.2832 0.1966
0.2832 0.2654 0.2221
0.1966 0.2221 0.2342
```

To calculate e^{M}, the variants using eigenvalues, Padé approximants, Taylor developments and matrix condition M will be used:

```
"expm(M)
ans =
1.4679 0.3550 0.2863
0.3550 1.2821 0.2342
0.2863 0.2342 1.1984
"expm1(M)
ans =
1.4679 0.3550 0.2863
0.3550 1.2821 0.2342
0.2863 0.2342 1.1984
"expm2(M)
ans =
1.4679 0.3550 0.2863
0.3550 1.2821 0.2342
0.2863 0.2342 1.1984
"expm3(M)
ans =
1.4679 0.3550 0.2863
```

```
0.3550 1.2821 0.2342
0.2863 0.2342 1.1984
```

As we can see, the exponential matrix coincides for all the methods used.

Consider the following matrix:

Texto Descripción generada automáticamente con confianza media

Find its transpose, its inverse, its determinant, its rank, its trace, its singular values, M ^{3} , log(M) and sqrt(M):

```
>> M=sym('[1/3,1/4,1/5; 1/4,1/5,1/6;1/5,1/6,1/7]')
M =
[ 1/3, 1/4, 1/5]
[ 1/4, 1/5, 1/6]
[ 1/5, 1/6, 1/7]
>> transposed=M'
transposed =
[ 1/3, 1/4, 1/5]
[ 1/4, 1/5, 1/6]
[ 1/5, 1/6, 1/7]
>> inverse=inv(M)
inverse =
[ 300, -900, 630]
[ -900, 2880, -2100]
[ 630, -2100, 1575]
>> determinant=det(M)
determinant =
1/378000
>> rank=rank(M)
rank =
3
>> trace=trace(M)
trace =
71/105
>> vsingulars=svd(M)
vsingulars =
(12703/88200 - (1045602865/35129803161616 + (102103^(1/2)*i)/49787136)^(1/3)/2 - 1030177/(99574272*
(1045602865/35129803161616 + (102103^(1/2)*i)/49787136)^(1/3))) - (3^(1/2)*(1030177/(49787136*(1045602865/351298031616
+ (102103^(1/2)*i)/49787136)^(1/3)) - ((102103^(1/2)*i)/49787136 + 1045602865/351298031616)^(1/3))*i)/2)^(1/2)

(12703/88200 - (1045602865/351298031616 + (102103^(1/2)*i)/49787136)^(1/3)/2 - 1030177/(99574272*
(1045602865/3512129803161616 + (102103^(1/2)*i)/49787136)^(1/3)) + (3^(1/2)*(1030177/(49787136*
(1045602865/35129803161616 + (102103^(1/2)*i)/49787136)^(1/3)) - ((102103^(1/2)*i)/49787136 +
1045602865/351298031616)^(1/3))*i)/2)^(1/2)

(1030177/(49787136*(1045602865/35129803161616 + (102103^(1/2)*i)/49787136)^(1/3)) + (1045602865/35129803161616 +
(102103^(1/2)*i)/49787136)^(1/3) + 12703/88200)^(1/2)
```

```
>> M^3

ans =

[ 10603/75600, 1227/11200, 26477/294000]

[ 1227/11200, 10783/126000, 74461/1058400]

[ 26477/294000, 74461/1058400, 8927/154350]

>> log(M)

ans =

[ -log(3), -log(4), -log(5)].

[ -log(4), -log(5), -log(6)].

[ -log(5), -log(6), -log(7)].

>> sqrt(M)

ans =

[ 3^(1/2)/3, 1/2, 5^(1/2)/5]

[ 1/2, 5^(1/2)/5, 6^(1/2)/6]

[ 5^(1/2)/5, 6^(1/2)/6, 7^(1/2)/7]
```

Exercise 3. Consider the following symbolic matrix:

Imagen que contiene Texto Descripción generada automáticamente

Calculate A', A $^{-1}$, determinant(A), trace(A), rank(A), inv(A) and A ^{2} .

```
>> A=sym('[a,b,c; 3*c,a-3*c,b; 3*b,-3*b+3*c,a-3*c]')

A =

[ a, b, c]

[ 3*c, a - 3*c, b]

[ 3*b, 3*c - 3*b, a - 3*c ]

>> transpose(A)

ans =

[ a, 3*c, 3*b ]

[ b, a - 3*c, 3*c - 3*b ]
[ c, b, a - 3*c]

>> pretty(det(A))

3 2 2 2 3 2 3

a - 6 a c + 3 a b - 9 a b c + 9 a c + 9 a c + 3 b + 9 b c + 9 c

>> pretty(trace(A))

3 a - 6 c

>> rank(A)

ans =

3

>> simplify(inv(A))

ans =

[ (a^2 - 6*a*c + 3*b^2 - 3*b*c + 9*c^2)/(c^2*(9*a + 9*b) - c*(6*a^2 + 9*b*a) + 3*a*b^2 + a^3 + 3*b^3 + 9*c^3),  -(a*b - 3*c^2)/(c^2*(9*a + 9*b) - c*(6*a^2 + 9*b*a) + 3*a*b^2 + a^3 + 3*b^3 + 9*c^3), (b^2 + 3*c^2 - a*c)/(c^2*(9*a + 9*b) - c*(6*a^2 + 9*b*a) + 3*a*b^2 + a^3 + 3*b^3 + 9*c^3))]

[ (3*b^2 + 9*c^2 - 3*a*c)/(c^2*(9*a + 9*b) - c*(6*a^2 + 9*b*a) + 3*a*b^2 + a^3 + 3*b^3 + 9*c^3),  -(c*(3*a + 3*b) - a^2)/(9*b*c^2 - 6*a^2*c + a*(3*b^2 - 9*b*c + 9*c^2) + a^3 + 3*b^3 + 9*c^3), -(a*b - 3*c^2)/(c^2*(9*a + 9*b) - c*(6*a^2 + 9*b*a) + 3*a*b^2 + a^3 + 3*b^3 + 9*c^3)]

[ -(3*a*b - 9*c^2)/(c^2*(9*a + 9*b) - c*(6*a^2 + 9*b*a) + 3*a*b^2 + a^3 + 3*b^3 + 9*c^3), (3*b^2 + 3*a*b - 3*a*c)/(c^2*(9*a + 9*b) - c*(6*a^2 + 9*b*a) + 3*a*b^2 + a^3 + 3*b^3 + 9*c^3), -(c*(3*a + 3*b) - a^2)/(9*b*c^2 - 6*a^2*c + a*(3*b^2 - 9*b*c + 9*c^2) + a^3 + 3*b^3 + 9*c^3)]

>> pretty(simplify(A^2))

+- -+

| 2 2 2 2 |

| a + 6 b c, 3 c - 6 b c + 2 a b, b - 3 c + 2 a c |

| |

| 2 2 2 2 2 |

| 3 b - 9 c + 6 a c, 6 b c - 3 b + (a - 3 c) , 2 b (a - 3 c) + 3 c |

| |

| 2 2 2 2 |

| 9 c - 18 b c + 6 a b, 3 b - 2 (a - 3 c) (3 b - 3 c), 6 b c - 3 b + (a - 3 c) |

+- -+
```

Exercise 4. Given the following matrices A and B:

image

a) *Calculate M1=A* ^{2} *+B* ^{2} *, M2=A* ^{2} *-B* ^{2} *, A* ^{n} *, B* ^{n} *, e* ^{A} *, e* ^{B}

b) *Find inverses, determinants, traces and norms of matrices A and B.*

```
>> A=sym('[cosh(a),sinh(a);sinh(a),cosh(a)]')

A =

[ cosh(a), sinh(a)].

[ sinh(a), cosh(a)].

>> B=sym('[sinh(a),cosh(a);cosh(a),sinh(a)]')

B =
[ sinh(a), cosh(a)].
[ cosh(a), sinh(a)].
>> M1=A^2+B^2
M1 =
[ 2*cosh(a)^2 + 2*sinh(a)^2, 4*cosh(a)*sinh(a)].
[ 4*cosh(a)*sinh(a), 2*cosh(a)^2 + 2*sinh(a)^2].
>> pretty(simplify(M1))
+- -+
| 2 |
| 4 sinh(a) + 2, 2 sinh(2 a) |
| |
| 2 |
| 2 sinh(2 a), 4 sinh(a) + 2 |
+- -+
>> M2=A^2-B^2
M2 =
[ 0, 0]
[ 0, 0]
```

To calculate A^n and B^n, we find their successive powers to try to see the law of formation:

```
> [simplify(A^2),simplify(A^3),simplify(A^4)].
ans =
[ cosh(2*a), sinh(2*a), cosh(3*a), sinh(3*a), cosh(4*a), sinh(4*a)].
[sinh(2*a), cosh(2*a), sinh(3*a), cosh(3*a), sinh(4*a), cosh(4*a)].
>> [simplify(B^2),simplify(B^3),simplify(B^4)].
ans =
[ cosh(2*a), sinh(2*a), sinh(3*a), cosh(3*a), cosh(4*a), sinh(4*a)].
[sinh(2*a), cosh(2*a), cosh(3*a), sinh(3*a), sinh(4*a), cosh(4*a)].
```

The law of formation of the nth powers is obvious, so we can now write:

```
image >> simplify(inv(A))
ans =
[ cosh(a), -sinh(a)].
[ -sinh(a), cosh(a)].
>> simplify(inv(B))
ans =
[ -sinh(a), cosh(a)].
[ cosh(a), -sinh(a)].
>> simplify(det(A))
ans =
1
>> simplify(det(B))
ans =
-1
>> simplify(trace(A))
ans =
2*cosh(a)
>> simplify(trace(B))
ans =
2*sinh(a)
>> simplify(exp(A))
ans =
[ exp(cosh(a)), exp(sinh(a))].
[ exp(sinh(a)), exp(cosh(a))].
>> simplify(exp(B))
ans =
[ exp(sinh(a)), exp(cosh(a))].
[ exp(cosh(a)), exp(sinh(a))].
```

C onsider a random normal square normal random matrix A of order 3 and calculate the diagonal matrix D with the eigenvalues of A and the matrix V *whose columns are its eigenvectors (if the output is complex, transform it to real).*

Find the balanced matrix of A and the real and complex forms of its Schur decomposition.

Find the coefficients of the characteristic polynomial of the matrix A.

*Calculate the upper triangular matrix R of the same dimension as the matrix A, the permutation matrix E and the orthogonal matrix Q such that A*E=Q*R and check the result.*

*Consider the Hessenberg matrix B of A and calculate the diagonal matrix D of generalised eigenvalues of A and B, and a matrix V whose columns are the corresponding eigenvectors, A*V=B*V*D. Also, calculate the vector of generalised singular values of A and B.*

```
>> A=randn(3)
A =
-0.4326 0.2877 1.1892
-1.6656 -1.1465 -0.0376
0.1253 1.1909 0.3273
>> [V,D] = eig(A)
V =
0.2827 0.4094 - 0.3992i 0.4094 + 0.3992i
0.8191 -0.0950 + 0.5569i -0.0950 - 0.5569i
-0.4991 0.5948 0.5948
D =
-1.6984 0 0
0 0.2233 + 1.0309i 0
0 0 0.2233 - 1.0309i
>> [V,D] = cdf2rdf(V,D)
V =
0.2827 0.4094 -0.3992
0.8191 -0.0950 0.5569
```

```
-0.4991 0.5948 0
D =
-1.6984 0 0
0 0.2233 1.0309
0 -1.0309 0.2233
>> [T,B] = balance(A)
T =
1 0 0
0 1 0
0 0 1
B =
-0.4326 0.2877 1.1892
-1.6656 -1.1465 -0.0376
0.1253 1.1909 0.3273
>> [U,T] = schur(A)
U =
0.2827 0.2924 0.9136
0.8191 -0.5691 -0.0713
-0.4991 -0.7685 0.4004
T =
-1.6984 0.2644 -1.2548
0 0.2233 0.7223
0 -1.4713 0.2233
>> [U,T] = rsf2csf(U,T)
U =
0.2827 -0.7482 + 0.1678i 0.2395 - 0.5242i
0.8191 0.0584 - 0.3266i -0.4661 + 0.0409i
-0.4991 -0.3279 - 0.4410i -0.6294 - 0.2298i
T =
-1.6984 1.0277 + 0.1517i 0.2165 + 0.7201i
0 0.2233 + 1.0309i 0.7490 - 0.0000i
0 0 0.2233 - 1.0309i
```

```
>> poly(A)
ans =
1.0000 1.2517 0.3540 1.8895
```

Then calculate the upper triangular matrix *R* of the same dimension as matrix *A* in the previous example, the permutation matrix *E* and the orthogonal matrix *Q* such that *A*E=Q*R* and check the result.

```
>> [Q,R,E] = qr(A)
Q =
-0.2507 0.4556 -0.8542
-0.9653 -0.0514 0.2559
0.0726 0.8887 0.4527
R =
1.7254 1.1211 -0.2380
0 1.2484 0.8346
0 0 -0.8772
E =
1 0 0
0 1 0
0 0 1
>> A*E
ans =
-0.4326 0.2877 1.1892
-1.6656 -1.1465 -0.0376
0.1253 1.1909 0.3273
>> Q*R
ans =
-0.4326 0.2877 1.1892
-1.6656 -1.1465 -0.0376
0.1253 1.1909 0.3273
```

It is observed that *A*E=Q*R* is satisfied.

We now consider the Hessenberg matrix B of A and calculate the diagonal matrix D of generalised eigenvalues of A and B , and a matrix V whose columns are the corresponding eigenvectors, $A*V=B*V*D$. We also calculate the vector of generalised singular values of A and B .

```
>> B=hess(A)
B =
-0.4326 -0.1976 1.2074
1.6703 -1.2245 0.1544
0 -1.0741 0.4053
>> [V,D] = eig(A,B)
V =
0.0567 1.0000 1.0000
-0.0376 -0.4998 0.5297
-1.0000 0.4172 0.3785
D =
1.0000 0 0
0 -0.4722 0
0 0 -2.1176
>> A*V
ans =
-1.2245 -0.0803 0.1699
-0.0137 -1.1082 -2.2872
-0.3649 -0.3334 0.8801
>> B*V*D
ans =
-1.2245 -0.0803 0.1699
-0.0137 -1.1082 -2.2872
-0.3649 -0.3334 0.8801
>> sigma = gsvd(A,B)
sigma =
0.2874
```

```
1.0000

3.4799
```

Consider the following 3x3 square matrix:

Un dibujo de una persona Descripción generada automáticamente co confianza baja

Perform the Schur, LU, QR, Cholesky, Hessenberg and singular value decompositions and check that the results are correct. Find also the pseudoinverse matrix of A.

First, we find the Schur decomposition, checking that the result is correct.

```
>> A=[1,5,-2;-7,3,1;2,2,-2];
>> [U,T] = schur(A)
U =
-0.0530 -0.8892 -0.4544
-0.9910 -0.0093 0.1337
0.1231 -0.4573 0.8807
T =
2.4475 -5.7952 -4.6361
5.7628 0.3689 2.4332
0 0 -0.8163
```

Now we check that *U*T*U'=A* and that *U*U'=eye* (3):

```
>> [U*T*U', U*U']
ans =
1.0000 5.0000 -2.0000 1.0000 0.0000 0.0000
-7.0000 3.0000 1.0000 0.0000 1.0000 0.0000
2.0000 2.0000 -2.0000 0.0000 0.0000 1.0000
```

We now find the LU, QR, Cholesky, Hesenberg and singular value decompositions, checking the results for each case:

```
" [L,U,P] = lu(A)
L =
1.0000 0 0
-0.1429 1.0000 0 Lower triangular matrix
-0.2857 0.5263 1.0000
```

```
U =
-7.0000 3.0000 1.0000
0 5.4286 -1.8571 Upper triangular matrix
0 0 -0.7368
P =
0 1 0
1 0 0
0 0 1
>> [P*A, L*U].
ans =
-7 3 1  -7 3 1
1 5 -2  1 5 -2 We have that P*A=L*U
2 2 -2  2 2 -2
>> [Q,R,E] = qr(A)
Q =
-0.1361 -0.8785 -0.4579
0.9526 -0.2430 0.1831
-0.2722 -0.4112 0.8700
R =
-7.3485 1.6330 1.7691
0 -5.9442 2.3366 Upper triangular matrix
0 0 -0.6410
E =
1 0 0
0 1 0
0 0 1
>> [A*E,Q*R] [A*E,Q*R] >> [A*E,Q*R]
ans =
1.0000 5.0000 -2.0000 1.0000 5.0000 -2.0000
-7.0000 3.0000 1.0000 -7.0000 3.0000 1.0000
2.0000 2.0000 -2.0000 2.0000 2.0000 -2.0000
```

Therefore, A*E=Q*R.

```
>> R = chol(A)
??? Error using ==> chol
Matrix must be positive definite.
```

You get an error message because the matrix is not positive definite.

```
>> [P,H] = hess(A)
P =
1.0000 0 0
0 -0.9615 0.2747
0 0.2747 0.9615
H =
1.0000 -5.3571 -0.5494
7.2801 1.8302 -2.0943
0 -3.0943 -0.8302
>> [P*H*P', P'*P].
ans =
1.0000 5.0000 -2.0000 1.0000 0 0
-7.0000 3.0000 1.0000 0 1.0000 0
2.0000 2.0000 -2.0000 0 0 1.0000
```

Then $PHP'=A$ and $P'P=I$.

```
>> [U,S,V]=svd(A)
U =
-0.1034 -0.8623 0.4957
-0.9808 0.0056 -0.1949
0.1653 -0.5064 -0.8463
S =
7.8306 0 0
0 6.2735 0 Diagonal matrix
0 0 0.5700
V =
0.9058 -0.3051 0.2940
-0.3996 -0.8460 0.3530
```

```
-0.1411 0.4372 0.8882
>> U*S*V'
ans =
1.0000 5.0000 -2.0000
-7.0000 3.0000 1.0000 We see that USV'=A
2.0000 2.0000 -2.0000
```

Now we calculate the pseudoinverse matrix of *A* :

```
>> X = pinv(A)
X =
0.2857 -0.2143 -0.3929
0.4286 -0.0714 -0.4643
0.7143 -0.2857 -1.3571
>> [A*X*A, X*A*X] >> [A*X*A, X*A*X] >> [A*X*A, X*A*X]
ans =
1.0000 5.0000 -2.0000 0.2857 -0.2143 -0.3929
-7.0000 3.0000 1.0000 0.4286 -0.0714 -0.4643
2.0000 2.0000 -2.0000 0.7143 -0.2857 -1.3571
```

Then, it is satisfied that $AXA=A$ and $XAX=X$.

E xercise 7. Given the following matrix:

Texto Descripción generada automáticamente

Calculate its eigenvalues, its characteristic polynomial, its canonical Jordan form and its singular values.

We begin by defining the matrix *A* as a symbolic matrix:

```
"A=sym('[1 0 0; 0 cos(a) -sin(a); 0 sin(a) cos(a)]')
A =
[ 1, 0, 0]
[ 0,cos(a),-sin(a)].
[ 0,sin(a), cos(a)].
```

```
"eigensys(A)
ans =
[ 1]
[cos(a)+1/2*(-4*sin(a)^2)^(1/2)]
[cos(a)-1/2*(-4*sin(a)^2)^(1/2)]
"pretty(simple(poly(A)))
3   2   2
x - 2 x cos(a) + x - x + 2 x cos(a) - 1
" jordan(A)
ans =
[1, 0, 0]
[0, cos(a)+1/2*(-4*sin(a)^2)^(1/2), 0].
[0, 0, cos(a)-1/2*(-4*sin(a)^2)^(1/2)].
"simple(svd(A))
ans =
[ 1]
[ 1/2*(4*cos(a-conj(a))+2*(-2+2*cos(2*a-2*conj(a)))^(1/2))^(1/2)]
[ 1/2*(4*cos(a-conj(a))-2*(-2+2*cos(2*a-2*conj(a)))^(1/2))^(1/2)]
```

Diagonalise the symmetric matrix whose rows are the vectors:

(3,-1,0), (-1,2,-1), (0,-1,3)

Find the step matrix V, check the result and see that the eigenvalues of the initial matrix are the elements of the diagonal of the matrix similar to the given one.

We calculate the diagonal matrix *J* similar to *A* , which will have the eigenvalues of *A* on the diagonal and the step matrix V. To do this, we use the command *[V,J]=jordan(A)* :

```
>> A=[3,-1,0; -1,2,-1; 0,-1,3]
A =
3 -1 0
-1 2 -1
```

```
    0 -1 3
    >> [V,J]=jordan(A)
    V =
    1 -1 1
    2 0 -1
    1 1 1



J =
    1 0 0
    0 3 0
    0 0 4
```

Now we check that the diagonal matrix *J* similar to *A* has the eigenvalues of *A* on the diagonal:

```
    >> eig(A)
    ans =
    1.0000
    3.0000
    4.0000
```

The matrices *A* and *J* turn out to be similar, since there exists the step matrix *V* that satisfies the relation $V^{-1} *A*V = J$:

```
    >> inv(V)*A*V
    ans =
    1.0000 0 -0.0000
    0 3.0000 0
    0 0 4.0000
```

E xercise 9. Find a diagonal matrix similar to each of the following matrices:

Imagen de la pantalla de un celular con letras Descripción generada automáticamente con confianza media

Find the step matrices and check the results. Find the characteristic polynomial for the three matrices.

```
>> A=sym('[0,-r,q;r,0,-p;-q,p,0]');
>> [V,J]=jordan(A)
V =
[ (q*(- p^2 - q^2 - q^2 - r^2)^(1/2))/(p^2 + q^2) - (p*r)/(p^2 + q^2), - (q*(- p^2 - q^2 - r^2)^(1/2))/(p^2 + q^2) - (p*r)/(p^2 + q^2), p/r], p/r].
[ - (p*(- p^2 - q^2 - q^2 - r^2)^(1/2))/(p^2 + q^2) - (q*r)/(p^2 + q^2), (p*(- p^2 - q^2 - r^2)^(1/2))/(p^2 + q^2) - (q*r)/(p^2 + q^2), q/r], q/r].
[ 1, 1, 1]

J =
[ -(- p^2 - q^2 - r^2)^(1/2), 0, 0].
[ 0, (- p^2 - q^2 - r^2)^(1/2), 0]
[ 0, 0, 0]
```

We already have the diagonal matrix *J* similar to the matrix *A* and the step matrix *V* . Now, we analyse matrix *B* :

```
" B=sym('[0,1,-sin(a);-1,0,cos(a);-sin(a),cos(a),0]')
"J=simple(jordan(B))
J =
[0, 0, 0]
[0, 0, 0]
[0, 0, 0]
```

It is observed that the matrix *B* has as its only eigenvalue zero and is of multiplicity 3. Moreover, the kernel of *B - 0*eye(3) =B* has dimension less than three, since the determinant of *B* is null. Specifically, it has dimension one (seen by calculating a basis with the command *nullspace(B)*) . Since the multiplicity and the dimension of the kernel differ, it is concluded that the matrix *B* is not diagonalisable:

```
    "null(B)
    ans =
    [cos(a)].
    [sin(a)] [sin(a)
    [ 1]
```

We have calculated a basis of the kernel of *B* , which consists of a single vector, so the dimension of the kernel of *B* is 1:

```
    "det (B)
    ans =
    0
```

We now analyse the matrix *C* :

```
    >> C=sym('[cos(a),-sin(a);sin(a),cos(a)]');
    >> [V,J]=jordan(C)
    V =
    [ 1/2, 1/2]
    [ i/2, -i/2]



J =
    [cos(a) - sin(a)*i, 0]
    [ 0, cos(a) + sin(a)*i]
```

We can try to simplify the expression of Jordan *J.'s* matrix.

W e already have the diagonal matrix *J* similar to the matrix *C* and the step matrix *V* . We now calculate the characteristic polynomials of the three matrices:

```
    >> pretty(poly(A))
    3 2 2 2
```

```
x + (p + q + r ) x
>> pretty(simple(sym(poly(B))))
3
X
>> pretty(simple(sym(poly(C))))
2
x - 2 cos(a) x + 1
```

Find the eigenvalues of the Wilkinson matrix of order 8, the magic matrix of order 8 and the Rosser matrix.

```
>> [eig(wilkinson(8)), eig(rosser), eig(magic(8))].
ans =
1.0e+003 *
-0.0010 -1.0200 0.2600
0.0002 0.0000 0.0518
0.0011 0.0001 -0.0518
0.0017 1.0000 0.0000
0.0026 1.0000 -0.0000 + 0.0000i
0.0028 1.0199 -0.0000 - 0.0000i
0.0042 1.0200 -0.0000 + 0.0000i
0.0043 1.0200 -0.0000 - 0.0000i
```

It is observed that the Wilkinson matrix has close, but not equal, pairs of eigenvalues. The Rosser matrix has one double eigenvalue, one null eigenvalue, three nearly equal eigenvalues, one very small eigenvalue and two opposite eigenvalues.

Consider the linear application f between two vector subspaces U (contained in R^3) and V (contained in R^4), such that for any point (a,b,c) of U one has:

Imagen que contiene Texto Descripción generada automáticamente

Find the kernel and the dimensions of the kernel and the image of f.

```
>> A=([1,0,0;0,0,0;0,0,1;0,0,0]);
```

The kernel is the set of vectors of *U* with null image in *V* :

```
>> null(A)

ans =

0

1

0
```

So the kernel will be the set of vectors {0, *b* ,0} with *b* varying in *U* . Moreover, the kernel is obviously of dimension 1, since we have seen that a basis is the vector {0,1,0}.

```
>> rank(A)

ans =

2
```

The dimension of the image of *f* will be 2, since the sum of the dimensions of the kernel and the image has to be the dimension of *U* (which is 3, since it is a subspace of R^3). On the other hand, the dimension of the image of *f* must coincide with the rank of the matrix of the linear application, which is 2. The two column vectors containing the elements of the non-zero minor defining the rank of the matrix will form a basis of the image of *f* .

```
>> det([1,0;0,1])

ans =

1
```

Then a basis of the image of *f* will be {{1,0,0,0,0},{0,0,1,0}}.

Given the quadratic form g:U→R defined as follows:

Diagrama Descripción generada automáticamente

classify it and find its reduced equation, its rank and its signature.

To classify the quadratic form we calculate its diagonal determinants.

```
>> G=[1,0,0;0,2,2;0,2,2]
G =
1 0 0
0 2 2
0 2 2
>> det(G)
ans =
0
>> det([1,0;0,2])
ans =
2
```

The quadratic form is degenerate, positive semidefinite.

To find the reduced equation we diagonalise its matrix and find the expression using the diagonal matrix.

```
>> J=jordan(G)
J =
0 0 0
0 1 0
0 0 4
```

The reduced equation of the quadratic form will be:

Imagen que contiene objeto, reloj Descripción generada automáticamente >> rank(J)

```
ans =
2
```

The rank of the quadratic form is 2, since the rank of its matrix is 2. The signature is also 2, since the number of positive terms on the diagonal of the diagonalised matrix is 2.