# Matlab 学习笔记

(官方 get start)

## 1 Complex Numbers

Complex numbers have both real and imaginary parts, where the imaginary unit is the square root of −1.

```
sqrt(-1)
ans =
0.0000 + 1.0000i
```

To represent the imaginary part of complex numbers, use either i or j.

```
c = [3+4i, 4+3j; -i, 10j]
c =
3.0000 + 4.0000i   4.0000 + 3.0000i
0.0000 - 1.0000i   0.0000 +10.0000i
```

## 2 Character Strings

You can assign a string to a variable.

```
myText = 'Hello, world';
```

If the text includes a single quote, use two single quotes within the definition.

```
otherText = 'You''re right'
```

myText and otherText are arrays, like all MATLAB variables. Their class or data type is char, which is short for character.

You can concatenate strings with square brackets, just as you concatenate numeric arrays.

To convert numeric values to strings, use functions, such as `num2str` or `int2str`.

```
f = 71;
c = (f-32)/1.8;
tempText = ['Temperature is ',num2str(c),'C']
```

# 3 2-D and 3-D Plots

## plot, hold, legend

**3-D Plots .** Three-dimensional plots typically display a surface defined by a function in two variables, z = f(x,y) .

To evaluate z, first create a set of (x,y) points over the domain of the function using meshgrid.

```
[X,Y] = meshgrid(-2:.2:2);
Z = X .* exp(-X.^2 - Y.^2);
Then, create a surface plot.
surf(X,Y,Z)
```

Both the surf function and its companion mesh display surfaces in three dimensions. surf displays both the connecting lines and the faces of the surface in color. mesh produces wireframe surfaces that color only the lines connecting the defining points.

# 4 sum, transpose, and diag

MATLAB has two transpose operators. The apostrophe operator (for example, A') performs a complex conjugate transposition. It flips a matrix about its main diagonal, and also changes the sign of the imaginary component of any complex elements of the matrix. The dot-apostrophe operator (A.'),

fliplr, flips a matrix from left to right.

# 5 Variables

Although variable names can be of any length, MATLAB uses only the first N characters of the name, (where N is the number returned by the function namelengthmax), and ignores the rest.

# 6 Numbers

Numbers represented in the double format have a maximum precision of 52 bits. Any double requiring more bits than 52 loses some precision. For example, the following code shows two unequal values to be equal because they are both truncated:

```
x = 36028797018963968;
y = 36028797018963972;
```

```
x == y
ans =
1
```

Integers have available precisions of 8-bit, 16-bit, 32-bit, and 64-bit. Storing the same numbers as 64-bit integers preserves precision:

```
x = uint64(36028797018963968);
y = uint64(36028797018963972);
x == y
ans =
0
```

MATLAB software stores the real and imaginary parts of a complex number. It handles the magnitude of the parts in different ways depending on the context. For instance, the sort function sorts based on magnitude and resolves ties by phase angle.

```
sort([3+4i, 4+3i])
ans =
4.0000 + 3.0000i
3.0000 + 4.0000i
```

This is because of the phase angle:

```
angle(3+4i)
ans =
0.9273
angle(4+3i)
ans =
0.6435
```

The "equal to" relational operator == requires both the real and imaginary parts to be equal. The other binary relational operators > <, >=, and <= ignore the imaginary part of the number and consider the real part only.

# 7 Functions

For a list of the elementary mathematical functions, type

```
help elfun
```

For a list of more advanced mathematical and matrix functions, type

```
help specfun
```

```
help elmat
```

Several special functions provide values of useful constants.

```
pi          3.14159265...
i           Imaginary unit
j           Same as i
eps         Floating-point relative precision, ε = 2^-52
realmin     Smallest floating-point number, 2^-1022
realmax     Largest floating-point number, (2 - ε)^1023
Inf         Infinity
NaN         Not-a-number
```

`Infinity` is generated by dividing a nonzero value by zero, or by evaluating well defined mathematical expressions that overflow, that is, exceed realmax.

`Not-a-number` is generated by trying to evaluate expressions like 0/0 or Inf-Inf that do not have well defined mathematical values.

The function names are not reserved. It is possible to overwrite any of them with a new variable, such as

```
eps = 1.e-6
```

and then use that value in subsequent calculations. The original function can be restored with

```
clear eps
```

## 8 The format Function

If you want more control over the output format, use the `sprintf` and `fprintf` functions.

## 9 Deleting Rows and Columns

You can delete rows and columns from a matrix using just a pair of square brackets.

Then, to delete the second column of X, use

```
X(:,2) = []
```

If you delete a single element from a matrix, the result is not a matrix anymore. So, expressions like

```
X(1,2) = []
```

result in an error. However, using a single subscript deletes a single element, or sequence of elements, and reshapes the remaining elements into a row vector. So

```
X(2:2:10) = []
```

results in

```
X = 16 9 2 7 13 12 1
```

## 10 Logical Subscripting

The logical vectors created from logical and relational operations can be used to reference subarrays. Suppose X is an ordinary matrix and L is a matrix of the same size that is the result of some logical operation. Then X(L) specifies the elements of X where the elements of L are nonzero.

This kind of subscripting can be done in one step by specifying the logical operation as the subscripting expression. Suppose you have the following set of data:

```
x = [2.1 1.7 1.6 1.5 NaN 1.9 1.8 1.5 5.1 1.8 1.4 2.2 1.6 1.8];
```

The NaN is a marker for a missing observation, such as a failure to respond to an item on a questionnaire. To remove the missing data with logical indexing, use isfinite(x), which is true for all finite numerical values and false for NaN and Inf:

```
x = x(isfinite(x))
```

```
x =
```

```
2.1 1.7 1.6 1.5 1.9 1.8 1.5 5.1 1.8 1.4 2.2 1.6 1.8
```

Now there is one observation, 5.1, which seems to be very different from the others. It is an outlier. The following statement removes outliers, in this case those elements more than three standard deviations from the mean:

```
x = x(abs(x-mean(x)) <= 3*std(x))
```

```
x =
```

```
2.1 1.7 1.6 1.5 1.9 1.8 1.5 1.8 1.4 2.2 1.6 1.8
```

For another example, highlight the location of the prime numbers in Dürer’ s magic square by using logical indexing and scalar expansion to set the nonprimes to 0.

```
A(~isprime(A)) = 0
```

## 11 The find Function

The find function determines the indices of array elements that meet a given logical condition. In its simplest form, find returns a column vector of indices.

Transpose that vector to obtain a row vector of indices. For example, start again with Dürer's magic square.

```
k = find(isprime(A))'
```

picks out the locations, using one-dimensional indexing, of the primes in the magic square:

```
k = 2 5 9 10 11 13
```

Display those primes, as a row vector in the order determined by k, with

```
A(k)
```

```
ans = 5 3 2 11 7 13
```

## 12 Multidimensional Arrays

Multidimensional arrays in the MATLAB environment are arrays with more than two subscripts. One way of creating a multidimensional array is by calling zeros, ones, rand, or randn with more than two arguments. For example,

```
R = randn(3,4,5)
```

A three-dimensional array might represent three-dimensional physical data, say the temperature in a room, sampled on a rectangular grid. Or it might represent a sequence of matrices, A(k), or samples of a time-dependent matrix, A(t). In these latter cases, the (i, j)th element of the kth matrix, or the tkth matrix, is denoted by A(i,j,k).

## 13 Cell Arrays

Cell arrays in MATLAB are multidimensional arrays whose elements are copies of other arrays. A cell array of empty matrices can be created with the cell function. But, more often, cell arrays are created by enclosing a miscellaneous collection of things in curly braces, {}. The curly braces are also used with subscripts to access the contents of various cells. For example,

```
C = {A sum(A) prod(prod(A))}
```

produces a 1-by-3 cell array. The three cells contain the magic square, the row vector of column sums, and the product of all its elements.

Here are two important points to remember. First, to retrieve the contents of one of the cells, use subscripts in curly braces. For example, C{1} retrieves the magic square and C{3} is 16!. Second, cell arrays contain copies of other arrays, not pointers to those arrays. If you subsequently change A, nothing happens to C.

You can use three-dimensional arrays to store a sequence of matrices of the same size. Cell arrays can be used to store a sequence of matrices of different sizes. For example,

```
M = cell(8,1);
for n = 1:8
M{n} = magic(n);
end
```

## 14 Characters and Text

To manipulate a body of text containing lines of different lengths, you have two choices—a padded character array or a cell array of strings. When creating a character array, you must make each row of the array the same length. (Pad the ends of the shorter rows with spaces.) The `char` function does this padding for you. For example,

```
S = char('A','rolling','stone','gathers','momentum.')
```

produces a 5-by-9 character array:

```
S =
A
rolling
stone
gathers
momentum.
```

Alternatively, you can store the text in a cell array. For example,

```
C = {'A';'rolling';'stone';'gathers';'momentum.'}
```

You can convert a padded character array to a cell array of strings with

```
C = cellstr(S)
```

and reverse the process with

```
S = char(C)
```

## 15 Structures

Structures are multidimensional MATLAB arrays with elements accessed by

textual field designators. For example,

```
S.name = 'Ed Plum';
S.score = 83;
S.grade = 'B+'
```

Like everything else in the MATLAB environment, structures are arrays, so you can insert additional elements. In this case, each element of the array is a structure with several fields. The fields can be added one at a time,

```
S(2).name = 'Toni Miller';
S(2).score = 91;
S(2).grade = 'A-';
```

or an entire element can be added with a single statement:

```
S(3) = struct('name','Jerry Garcia',...
'score',70,'grade','C')

scores = [S.score]
scores =
83
91
70
avg_score = sum(scores)/length(scores)
avg_score =
81.3333
```

To create a character array from one of the text fields (name, for example), call the char function on the comma-separated list produced by S.name:

```
names = char(S.name)
```

Similarly, you can create a cell array from the name fields by enclosing the list-generating expression within curly braces:

```
names = {S.name}
```

To assign the fields of each element of a structure array to separate variables outside of the structure, specify each output to the left of the equals sign, enclosing them all within square brackets:

```
[N1 N2 N3] = S.name

N1 =

Ed Plum

N2 =

Toni Miller

N3 =

Jerry Garcia
```

## 16 Dynamic Field Names

The most common way to access the data in a structure is by specifying the name of the field that you want to reference. Another means of accessing structure data is to use dynamic field names. These names express the field as a variable expression that MATLAB evaluates at run time. The dot-parentheses syntax shown here makes expression a dynamic field name:

```
structName.(expression)
```

Index into this field using the standard MATLAB indexing syntax. For example, to evaluate expression into a field name and obtain the values of that field at columns 1 through 25 of row 7, use

structName.(expression)(7,1:25)

**Dynamic Field Names Example.** The avgscore function shown below computes an average test score, retrieving information from the testscores structure using dynamic field names:

```
function avg = avgscore(testscores, student, first, last)

for k = first:last

scores(k) = testscores.(student).week(k);

end

avg = sum(scores)/(last - first + 1);
```

You can run this function using different values for the dynamic field student.

First, initialize the structure that contains scores for a 25-week period:

```
testscores.Ann_Lane.week(1:25) = ...

[95 89 76 82 79 92 94 92 89 81 75 93 ...

85 84 83 86 85 90 82 82 84 79 96 88 98];

testscores.William_King.week(1:25) = ...

[87 80 91 84 99 87 93 87 97 87 82 89 ...

86 82 90 98 75 79 92 84 90 93 84 78 81];
```

Now run avgscore, supplying the students name fields for the testscores structure at run time using

dynamic field names:

```
avgscore(testscores, 'Ann_Lane', 7, 22)
ans =
85.2500
avgscore(testscores, 'William_King', 7, 22)
ans =
87.7500
```

## 17 Kronecker Tensor Product

The Kronecker product, kron(X,Y), of two matrices is the larger matrix formed from all possible products of the elements of X with those of Y. If X is m-by-n and Y is p-by-q, then kron(X,Y) is mp-by-nq. The elements are arranged in the following order:

```
[X(1,1)*Y X(1,2)*Y . . . X(1,n)*Y
            . . .
X(m,1)*Y X(m,2)*Y . . . X(m,n)*Y]
```

## 18 Vector and Matrix Norms

The p-norm of a vector x,

$$\|x\|_p = \left(\sum |x_i|^p\right)^{1/p},$$

is computed by norm(x,p). This is defined by any value of p > 1, but the most common values of p are 1, 2, and ∞. The default value is p = 2, which corresponds to Euclidean length:

```
v = [2 0 -1];
[norm(v,1) norm(v) norm(v,inf)]
ans =
3.0000      2.2361      2.0000
```

The p-norm of a matrix A,

$$\|A\|_p = \max_x \frac{\|Ax\|_p}{\|x\|_p},$$

can be computed for p = 1, 2, and ∞ by norm(A,p). Again, the default value is p = 2.

## 19 Systems of Linear Equations

If A is singular and Ax = b has a solution, you can find a particular solution that is not unique, by typing

```
P = pinv(A)*b
```

P is a pseudoinverse of A. If Ax = b does not have an exact solution, pinv(A) returns a least-squares solution.

**Exact Solutions**

For b =[5;2;12], the equation Ax = b has an exact solution, given by

```
pinv(A)*b
ans =
0.3850
-0.1103
0.7066
```

Verify that pinv(A)*b is an exact solution by typing

```
A*pinv(A)*b
ans =
5.0000
2.0000
12.0000
```

**Least-Squares Solutions**

However, if b = [3;6;0], Ax = b does not have an exact solution. In this case,

pinv(A)*b returns a least-squares solution. If you type

```
A*pinv(A)*b
ans =
-1.0000
4.0000
2.0000
```

you do not get back the original vector b.

You can determine whether Ax = b has an exact solution by finding the row reduced echelon form of the augmented matrix [A b]. To do so for this example, enter

```
rref([A b])
ans =
    1.0000         0    2.2857         0
         0    1.0000    1.5714         0
         0         0         0    1.0000
```

Since the bottom row contains all zeros except for the last entry, the equation does not have a solution. In this case, pinv(A) returns a least-squares solution.

## 20 Overdetermined Systems

This example shows how overdetermined syetems are often encountered in various kinds of curve fitting to experimental data.

A quantity, y, is measured at several different values of time, t, to produce the following observations. You can enter the data and view it in a table with the following statements.

```
t = [0 .3 .8 1.1 1.6 2.3]';
y = [.82 .72 .63 .60 .55 .50]';
B = table(t,y)
```

Try modeling the data with a decaying exponential function

$$y(t) = c_1 + c_2 * e^{-t}$$

The preceding equation says that the vector y should be approximated by a linear combination of two other vectors. One is a constant vector containing all ones and the other is the vector with components exp(-t). The unknown coefficients, and , can be computed by doing a least-squares fit, which minimizes the sum of the squares of the deviations of the data from the model. There are six equations in two unknowns, represented by a 6-by-2 matrix.

```
E = [ones(size(t)) exp(-t)]
```

Use the backslash operator to get the least-squares solution.

```
c = E\y
```

The following statements evaluate the model at regularly spaced increments in t, and then plot the result together with the original data:

```
T = (0:0.1:2.5)';
Y = [ones(size(T)) exp(-T)]*c;
plot(T,Y,'-',t,y,'o')
```

# 21 Underdetermined Systems

This example shows how the solution to underdetermined systems is not unique. Underdetermined linear systems involve more unknowns than equations. The matrix left division operation in MATLAB finds a basic solution, which has at most m nonzero components for an m-by-n coefficient matrix.

Here is a small, random example:

```
R = [6 8 7 3; 3 5 4 1]
```

```
rng(0);
```

```
b = randi(8,2,1)
```

The linear system Rp = b involves two equations in four unknowns. Since the coefficient matrix contains small integers, it is appropriate to use the format command to display the solution in rational format. The particular solution is obtained with

```
format rat
```

```
p = R\b
```

One of the nonzero components is p(2) because R(:,2) is the column of R with largest norm. The other nonzero component is p(4) because R(:,4) dominates after R(:,2) is eliminated.

The complete general solution to the underdetermined system can be characterized by adding p to an arbitrary linear combination of the null space vectors, which can be found using the null function with an option requesting a rational basis.

```
Z = null(R,'r')
```

It can be confirmed that R*Z is zero and that the residual R*x - b is small for any vector x, where

```
x = p + Z*q.
```

Since the columns of Z are the null space vectors, the product Z*q is a linear combination of those vectors:

$$Z * q = \begin{pmatrix} \bar{x}_1 & \bar{x}_2 \end{pmatrix} \begin{pmatrix} u \\ w \end{pmatrix} = u\bar{x}_1 + w\bar{x}_2 \ .$$

To illustrate, choose an arbitrary q and construct x.

```
q = [-2; 1];
```

```
x = p + Z*q;
```

Calculate the norm of the residual.

```
format short
```

```
norm(R*x - b)
```

## 22 Pseudoinverses

Rectangular matrices do not have inverses or determinants. At least one of the equations AX = I and XA = I does not have a solution. A partial replacement for the inverse is provided by the Moore-Penrose pseudoinverse, which is computed by the pinv function:

```
format short
C = fix(10*gallery('uniformdata',[3 2],0));
X = pinv(C)
```

Q = X*C is the 2-by-2 identity, but the matrix

P = C*X is not the 3-by-3 identity. However, P acts like an identity on a portion of the space in the sense that P is symmetric, P*C is equal to C, and X*P is equal to X.

## 23 Solving a Rank-Deficient System

If A is m-by-n with m > n and full rank n, each of the three statements

```
x = A\b
x = pinv(A)*b
x = inv(A'*A)*A'*b
```

theoretically computes the same least-squares solution x, although the backslash operator does it faster. However, if A does not have full rank, the solution to the least-squares problem is not unique. There are many vectors x that minimize norm(A*x -b) .

The solution computed by x = A\b is a basic solution; it has at most r nonzero components, where r is the rank of A. The solution computed by x = pinv(A)*b is the minimal norm solution because it minimizes norm(x).

An attempt to compute a solution with x = inv(A'*A)*A'*b fails because A'*A is singular.

## 24 Cholesky Factorization

The Cholesky factorization expresses a symmetric matrix as the product of a triangular matrix and its transpose

```
A = R'R,
```

where R is an upper triangular matrix.

Not all symmetric matrices can be factored in this way; the matrices that have such a factorization are said to be positive definite.

```
A = pascal(6)
```

```
R = chol(A)
```

The Cholesky factorization allows the linear system

```
Ax = b
```

to be replaced by

```
R'Rx = b.
```

Because the backslash operator recognizes triangular systems, this can be solved in the MATLAB environment quickly with

```
x = R\(R'\b)
```

If A is n-by-n, the computational complexity of chol(A) is O(n3), but the complexity of the subsequent backslash solutions is only O(n2).

## 25 LU Factorization

LU factorization, or Gaussian elimination, expresses any square matrix A as the product of a permutation of a lower triangular matrix and an upper triangular matrix

```
A = LU,
```

where L is a permutation of a lower triangular matrix with ones on its diagonal and U is an upper triangular matrix.

The permutations are necessary for both theoretical and computational reasons.

The matrix

0 1

1 0

cannot be expressed as the product of triangular matrices without interchanging its two rows. Although the matrix

ε 1

1 0

can be expressed as the product of triangular matrices, when ε is small, the elements in the factors are large and magnify errors, so even though the permutations are not strictly necessary, they are desirable. Partial pivoting ensures that the elements of L are bounded by one in magnitude and that the elements of U are not much larger than those of A.

```
[L,U] = lu(B)
```

The LU factorization of A allows the linear system

```
A*x = b
```

to be solved quickly with

```
x = U\(L\b)
```

Determinants and inverses are computed from the LU factorization using

```
det(A) = det(L)*det(U)
```

and

```
inv(A) = inv(U)*inv(L)
```

You can also compute the determinants using `det(A) = prod(diag(U))` since `det(L) = 1 or -1`, though the signs of the determinants might be reversed.

## 26 QR Factorization

An orthogonal matrix, or a matrix with orthonormal columns, is a real matrix whose columns all have unit length and are perpendicular to each other.

For complex matrices, the corresponding term is unitary. Orthogonal and unitary matrices are desirable for numerical computation because they preserve length, preserve angles, and do not magnify errors.

The orthogonal, or QR, factorization expresses any rectangular matrix as the product of an orthogonal or unitary matrix and an upper triangular matrix. A column permutation might also be involved:

```
A = QR
```

or

```
AP = QR,
```

where Q is orthogonal or unitary, R is upper triangular, and P is a permutation.

There are four variants of the QR factorization—full or economy size, and with or without column permutation. See `help qr`.

## 27 Inverse and Fractional Powers

If A is square and nonsingular, A^(-p) effectively multiplies inv(A) by itself p-1 times.

Fractional powers, like A^(2/3), are also permitted; the results depend upon the distribution of the eigenvalues of the matrix.

## 28 Exponentials

The function

```
sqrtm(A)
```

computes A^(1/2) by a more accurate algorithm. The m in `sqrtm` distinguishes this function from

sqrt(A), which, like A.^(1/2), does its job element-by-element.

The function

expm(A)

computes the matrix exponential.

## 29 Schur Decomposition

The MATLAB advanced matrix computations do not require eigenvalue

decompositions. They are based, instead, on the Schur decomposition

$A = USU^T$.

where U is an orthogonal matrix and S is a block upper triangular matrix with 1-by-1 and 2-by-2 blocks on the diagonal. The eigenvalues are revealed by the diagonal elements and blocks of S, while the columns of U provide a basis with much better numerical properties than a set of eigenvectors. The Schur decomposition of this defective example is

```
[U,S] = schur(A)
```

## 30 Singular Values

The eigenvalue decomposition is the appropriate tool for analyzing a matrix when it represents a mapping from a vector space into itself, as it does for an ordinary differential equation. However, the singular value decomposition is the appropriate tool for analyzing a mapping from one vector space into another vector space, possibly with a different dimension. Most systems of simultaneous linear equations fall into this second category.

In particular, the singular value decomposition of a real matrix is always real, but the eigenvalue decomposition of a real, nonsymmetric matrix might be complex.

```
[U,S,V] = svd(A)
```

## 31 Function Handles

You can create a handle to any MATLAB function, and then use that handle as a means of referencing the function. A function handle is typically passed in an argument list to other functions, which can then execute, or evaluate, the function using the handle. Construct a function handle in MATLAB using the at sign, @, before the function name. The following example creates a function handle for the sin function and assigns it to the variable fhandle:

```
fhandle = @sin;
```

You can call a function by means of its handle in the same way that you would call the function using its name. The syntax is

```
fhandle(arg1, arg2, ...);
```

The function plot_fhandle, shown below, receives a function handle and data, generates y-axis data using the function handle, and plots it:

```
function plot_fhandle(fhandle, data)
plot(data, fhandle(data))
```

When you call plot_fhandle with a handle to the sin function and the argument shown below, the resulting evaluation produces a sine wave plot:

```
plot_fhandle(@sin, -pi:0.01:pi)
```

## 32 Function Functions

A class of functions called "function functions" works with nonlinear functions of a scalar variable. That is, one function works on another function. The function functions include

- Zero finding
- Optimization
- Quadrature
- Ordinary differential equations

The first argument to `fminsearch` is a function handle to the function being minimized and the second argument is a rough guess at the location of the minimum:

```
p = fminsearch(@humps,.5)
```

Numerical analysts use the terms quadrature and integration to distinguish between numerical approximation of definite integrals and numerical integration of ordinary differential equations.

MATLAB quadrature routines are `quad` and `quadl`. The statement

```
Q = quadl(@humps,0,1)
```

computes the area under the curve in the graph.

Finally, the graph shows that the function is never zero on this interval. So, if you search for a zero with

```
z = fzero(@humps,.5)
```

you will find one outside the interval

```
z =
-0.1316
```

## 33 Multivariate Data

MATLAB uses column-oriented analysis for multivariate statistical data. Each column in a data set represents a variable and each row an observation. The (i,j)th element is the ith observation of the jth variable.

As an example, consider a data set with three variables:

• Heart rate

• Weight

• Hours of exercise per week

For five observations, the resulting array might look like

```
D = [ 72    134   3.2
      81    201   3.5
      69    156   7.1
      82    148   2.4
      75    170   1.2 ]
```

The first row contains the heart rate, weight, and exercise hours for patient 1, the second row contains the data for patient 2, and so on. Now you can apply many MATLAB data analysis functions to this data set. For example, to obtain the mean and standard deviation of each column, use

```
mu = mean(D), sigma = std(D)
```

For a list of the data analysis functions available in MATLAB, type

```
help datafun
```

If you have access to the Statistics ToolboxTM software, type

```
help stats
```

## 34 Data Analysis

Every data analysis has some standard components:

• Preprocessing — Consider outliers and missing values, and smooth data to identify possible models.

• Summarizing — Compute basic statistics to describe the overall location, scale, and shape of the data.

• Visualizing — Plot data to identify patterns and trends.

• Modeling — Give data trends fuller descriptions, suitable for predicting new values.

Data analysis moves among these components with two basic goals in mind:

1 Describe the patterns in the data with simple models that lead to accurate predictions.

2 Understand the relationships among variables that lead to the model.

# 35 Preprocessing Data

Begin by loading the data in count.dat:

```
load count.dat
```

**Missing Data .** The MATLAB NaN (Not a Number) value is normally used to represent missing data. NaN values allow variables with missing data to maintain their structure .

Check the data at the third intersection for NaN values using the `isnan` function:

```
c3 = count(:,3); % Data at intersection 3
c3NaNCount = sum(isnan(c3))
```

**Outliers .** Outliers are data values that are dramatically different from patterns in the rest of the data. Identifying outliers, and deciding what to do with them, depends on an understanding of the data and its source.

**Smoothing and Filtering.**

# 36 Visualizing Data

# 37 Figure Windows

To make an existing figure window the current figure, you can click the mouse while the pointer is in that window or you can type,

```
figure(n)
```

If you have set any figure properties in the previous plot, you might want to use the clf command with the reset option,

```
clf reset
```

before creating your new plot to restore the figure’ s properties to their defaults.

## 38 Controlling the Axes

The `axis` command provides a number of options for setting the scaling, orientation, and aspect ratio of graphs.

**Setting Axis Limits.**

The axis command enables you to specify your own limits:

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

or for three-dimensional graphs,

```
axis([xmin xmax ymin ymax zmin zmax])
```

Use the command

```
axis auto
```

to enable automatic limit selection again.

**Setting the Axis Aspect Ratio**

The axis command also enables you to specify a number of predefined modes.

For example,

```
axis square
```

makes the x-axis and y-axis the same length.

```
axis equal
```

makes the individual tick mark increments on the x-axes and y-axes the same length.

```
axis auto normal
```

returns the axis scaling to its default automatic mode.

**Setting Axis Visibility**

You can use the axis command to make the axis visible or invisible.

```
axis on
```

makes the axes visible. This is the default.

```
axis off
```

makes the axes invisible.

**Setting Grid Lines**

The grid command toggles grid lines on and off. The statement

```
grid on
```

turns the grid lines on, and

```
grid off
```

turns them back off again

## 39 Adding Axis Labels and Titles

The `xlabel` , `ylabel` , and `zlabel` commands add x-, y-, and z-axis labels. The `title` command adds a title at the top of the figure and the `text` function inserts text anywhere in the figure.

You can produce mathematical symbols using LaTeX notation in the text string.

## 40 Generating MATLAB Code to Recreate a Figure

You can generate MATLAB code that recreates a figure and the graph it contains by selecting Generate code from the figure File menu. This option is particularly useful if you have developed a graph using plotting tools and want to create a similar graph using the same or different data.

## 41 Creating Mesh and Surface Plots

`meshgrid`

**Colored Surface Plots**

```
surf(X,Y,Z)
colormap hsv
colorbar
```

**Making Surfaces Transparent**

```
surf(X,Y,Z)
colormap hsv
alpha(.4)
```

**Illuminating Surface Plots with Lights**

Lighting is the technique of illuminating an object with a directional light source. In certain cases, this

technique can make subtle differences in surface shape easier to see. Lighting can also be used to add realism to three-dimensional graphs.

```
surf(X,Y,Z,'FaceColor','red','EdgeColor','none')
camlight left; lighting phong
```

## 42 Plotting Image Data

```
load durer
image(X)
colormap(map)
axis image
```

## 43 Reading and Writing Images

You can read standard image files (TIFF, JPEG, BMP, and so on, using the imread function. The type of data returned by `imread` depends on the type of image you are reading.

You can write MATLAB data to a variety of standard image formats using the `imwrite` function.

## 44 Graphics Objects

When you call a plotting function, MATLAB creates the graph using various graphics objects, such as a figure window, axes, lines, text, and so on. Each object has a fixed set of properties, which you can use to control the behavior and appearance of your graph.

## 45 Object Handles

When MATLAB creates a graphics object, MATLAB assigns an identifier to the object. This identifier is called a handle. You can use this handle to access the object' s properties with the set and get functions. For example, the following statements create a graph and return a handle to a lineseries object in h :

```
x = 1:10;
y = x.^3;
h = plot(x,y);
```

You can use the handle h to set the properties of the lineseries object. For example, you can set its Color property:

```
set(h,'Color','red')
```

You can also specify the lineseries properties when you call the plotting function:

```
h = plot(x,y,'Color','red');
```

You can query the lineseries properties to see the current value:

```
get(h,'LineWidth')
```

The get function returns the answer (in units of points for LineWidth ):

```
ans =
0.5000
```

If you call get with only a handle, MATLAB returns a list of the object's properties:

```
get(h)
```

If you call set with only a handle, MATLAB returns a list of the object's properties with information about possible values:

```
set(h)
```

## 46 Functions for Working with Objects

**Finding All Objects of a Certain Type**

Because all objects have a Type property that identifies the type of object, you can find the handles of all occurrences of a particular type of object. For example,

```
h = findobj('Type','patch');
```

finds the handles of all patch objects.

You can specify multiple properties to narrow the search. For example,

```
h = findobj('Type','line','Color','r','LineStyle',':');
```

**Limiting the Scope of the Search**

You can specify the starting point in the object hierarchy by passing the handle of the starting figure or axes as the first argument. For example,

```
h = findobj(gca,'Type','text','String','\pi/2');
```

finds the string π/2 only within the current axes.

**Using findobj as an Argument**

```
set(findobj('Type','line','Color','red'),'LineStyle',':')
```

## 47 Primary and Subfunctions

Any function that is not anonymous must be defined within a file. Each such function file contains a required primary function that appears first, and any number of subfunctions that can follow the primary. Primary functions have a wider scope than subfunctions. That is, primary functions can be called from outside of the file that defines them (for example, from the MATLAB command line or from functions in other files) while subfunctions cannot. Subfunctions are visible only to the primary function and other subfunctions within their own file.

## 48 Private Functions

Private functions reside in subfolders with the special name private . They are visible only to functions in the parent folder. For example, assume the folder newmath is on the MATLAB search path. A subfolder of newmath called private can contain functions that only the functions in newmath can call.

## 49 Nested Functions

## 50 Global Variables

If you want more than one function to share a single copy of a variable, simply declare the variable as `global` in all the functions.Do the same thing at the command line if you want the base workspace to access the variable. The global declaration must occur before the variable is actually used in a function. Although it is not required, using capital letters for the names of global variables helps distinguish them from other variables. For example, create a new function in a file called falling.m :

```
function h = falling(t)
global GRAVITY
h = 1/2*GRAVITY*t.^2;
```

Then interactively enter the statements

```
global GRAVITY
GRAVITY = 32;
```

```
y = falling((0:.1:5)');
```