הפקולטה למדעי המחשב
הטכניון - מכון טכנולוגי לישראל
234127 - מבוא למחשב בשפת מטלאב

נספח א: תחביר בסיסי

if expression
    statements
elseif expression
    statements
else
    statements
end

switch (value)
    case {v1, v2, ..., vn}
        statements
    case v3
        statements
    otherwise
        statements
end

for var = array
    statements
end

while expression
    statements
end
Matlab — פונקציות שימושיות ב -

**all**
For vectors, all(V) returns logical 1 (TRUE) if none of the elements of the vector are zero. Otherwise it returns logical 0 (FALSE). For matrices, ALL(X) operates on the columns of X, returning a row vector of logical 1’s and 0’s. For N-D arrays, ALL(X) operates on the first non-singleton dimension.

all(X,DIM) works down the dimension DIM. For example, ALL(X,1) works down the first dimension (the rows) of X.

**any**

B = any(A) tests whether any of the elements along various dimensions of an array is a non-zero number or is logical 1 (true). any ignores entries that are NaN (Not a Number).

If A is a vector, any(A) returns logical 1 (true) if any of the elements of A is a nonzero number or is logical 1 (true), and returns logical 0 (false) if all the elements are zero.

If A is a matrix, any(A) treats the columns of A as vectors, returning a row vector of logical 1’s and 0’s.

**ceil**

B = ceil(A) rounds the elements of A to the nearest integers greater than or equal to A. For complex A, the imaginary and real parts are rounded independently.

**char**

S = char(A) converts the array A that contains nonnegative integers representing character codes into a MATLAB character array (the first 127 codes are ASCII). The actual characters displayed depends on the character encoding scheme for a given font. The result for any elements of A outside the range from 0 to 65535 is not defined (and may vary from platform to platform). Use double to convert a character array into its numeric codes.

**double**

double(X) returns the double precision value for X.

If X is already a double precision array, double has no effect.

double is called for the expressions in FOR, IF, and WHILE loops if the expression isn't already double precision. double should be overloaded for all objects where it makes sense to convert it into a double precision value.
**find**

ind = find(X) locates all nonzero elements of array X, and returns the linear indices of those elements in vector ind. If X is a row vector, then ind is a row vector; otherwise, ind is a column vector. If X contains no nonzero elements or is an empty array, then ind is an empty array.

ind = find(X, k) or ind = find(X, k, 'first') returns at most the first k indices corresponding to the nonzero entries of X. k must be a positive integer, but it can be of any numeric data type.

ind = find(X, k, 'last') returns at most the last k indices corresponding to the nonzero entries of X.

[row, col] = find(X, ...) returns the row and column indices of the nonzero entries in the matrix X.

This syntax is especially useful when working with sparse matrices. If X is an N-dimensional array with N > 2, col contains linear indices for the columns.

**floor**

B = floor(A) rounds the elements of A to the nearest integers less than or equal to A. For complex A, the imaginary and real parts are rounded independently.

**input**

NUM = input(PROMPT) displays the PROMPT string on the screen, waits for input from the keyboard, evaluates any expressions in the input, and returns the value in NUM. To evaluate expressions, input accesses variables in the current workspace. If you press the return key without entering anything, input returns an empty matrix.

STR = input(PROMPT, 's') returns the entered text as a MATLAB string, without evaluating expressions.

To create a prompt that spans several lines, use '\n' to indicate each new line. To include a backslash ('\\') in the prompt, use '\\'.

**isempty**

TF = isempty(A) returns logical 1 (true) if A is an empty array and logical 0 (false) otherwise. An empty array has at least one dimension of size zero, for example, 0-by-0 or 0-by-5.

**length**

n = length(X) returns the size of the longest dimension of X. If X is a vector, this is the same as its length.

**log10**

log10(X) is the base 10 logarithm of the elements of X. Complex results are produced if X is not positive.
**max**

\[ C = \text{max}(A) \]
returns the largest elements along different dimensions of an array.
If \( A \) is a vector, \( \text{max}(A) \) returns the largest element in \( A \).
If \( A \) is a matrix, \( \text{max}(A) \) treats the columns of \( A \) as vectors, returning a row vector containing the maximum element from each column.
If \( A \) is a multidimensional array, \( \text{max}(A) \) treats the values along the first non-singleton dimension as vectors, returning the maximum value of each vector.

**min**

\[ C = \text{min}(A) \]
returns the smallest elements along different dimensions of an array.
If \( A \) is a vector, \( \text{min}(A) \) returns the smallest element in \( A \).
If \( A \) is a matrix, \( \text{min}(A) \) treats the columns of \( A \) as vectors, returning a row vector containing the minimum element from each column.
If \( A \) is a multidimensional array, \( \text{min}(A) \) operates along the first non-singleton dimension.
\[ C = \text{min}(A,[],\text{dim}) \]
returns the smallest elements along the dimension of \( A \) specified by scalar \( \text{dim} \).
For example, \( \text{min}(A,[],1) \) produces the minimum values along the first dimension (the rows) of \( A \).

**mod**

\[ M = \text{mod}(X,Y) \]
if \( Y \neq 0 \), returns \( X - n \cdot Y \) where \( n = \text{floor}(X/Y) \). If \( Y \) is not an integer and the quotient \( X/Y \) is within roundoff error of an integer, then \( n \) is that integer.
The inputs \( X \) and \( Y \) must be real arrays of the same size, or real scalars.
Example: \( X = \begin{bmatrix} 23 & 24 & 25 & 26 \end{bmatrix} \)
\[ M = \text{mod}(X, 3) \]
Then \( M \) gets the array \( \begin{bmatrix} 2 & 0 & 1 & 2 \end{bmatrix} \)

**ones**

\( Y = \text{ones}(n) \)
returns an \( n \)-by-\( n \) matrix of 1s. An error message appears if \( n \) is not a scalar.
\( Y = \text{ones}(m,n) \) or \( Y = \text{ones}([m n]) \)
returns an \( m \)-by-\( n \) matrix of ones.
\( Y = \text{ones}(m,n,p,...) \) or \( Y = \text{ones}([m n p ...]) \)
returns an \( m \)-by-\( n \)-by-\( p \)-by-... array of 1s.

**prod**

\[ S = \text{prod}(X) \]
is the product of the elements of the vector \( X \). If \( X \) is a matrix, \( S \) is a row vector with the product over each column. For \( N \)-D arrays, \( \text{prod}(X) \) operates on the first non-singleton dimension. If \( X \) is floating point, that is double or single, \( S \) is computed natively, that is in the same class as \( X \), and \( S \) has the same class \( X \). If \( X \) is not floating point, \( S \) is computed in double and \( S \) has class double.
\( \text{prod}(X,\text{DIM}) \)
works along the dimension \( \text{DIM} \).
\( S = \text{prod}(X,\text{'}double\text{'} ) \) and \( S = \text{prod}(X,\text{DIM,}\text{'}double\text{'} ) \) compute \( S \) in double and \( S \) has class double, even if \( X \) is single.
\( S = \text{prod}(X,\text{'}native\text{'} ) \) and \( S = \text{prod}(X,\text{DIM,}\text{'}native\text{'} ) \) compute \( S \) natively and \( S \) has the same class as \( X \).
Example: If \( X = \begin{bmatrix} 0 & 1 & 2; & 3 & 4 & 5 \end{bmatrix} \) then \( \text{prod}(X,1) \) is \( \begin{bmatrix} 0 & 4 & 10 \end{bmatrix} \) and \( \text{prod}(X,2) \) is \( \begin{bmatrix} 0; & 60 \end{bmatrix} \).
repmat

B = repmat(A,m,n) creates a large matrix B consisting of an m-by-n tiling of copies of A. The size of B is [size(A,1)*m, (size(A,2)*n].
The statement repmat(A,n) creates an n-by-n tiling.
B = repmat(A,[m n]) accomplishes the same result as repmat(A,m,n).
B = repmat(A,[m n p...]) produces a multidimensional array B composed of copies of A.
The size of B is [size(A,1)*m, size(A,2)*n, size(A,3)*p, ...].

reshape

B = reshape(X,M,N) returns the M-by-N matrix whose elements are taken columnwise from X. An error results if X does not have M*N elements.

round

round(X) rounds the elements of X to the nearest integers.

size

d = size(X) returns the sizes of each dimension of array X in a vector d with ndims(X) elements. If X is a scalar, which MATLAB regards as a 1-by-1 array, size(X) returns the vector [1 1].
[m,n] = size(X) returns the size of matrix X in separate variables m and n.
m = size(X,dim) returns the size of the dimension of X specified by scalar dim.
[d1,d2,d3,...,dn] = size(X), for n > 1, returns the sizes of the dimensions of the array X in the variables d1,d2,d3,...,dn, provided the number of output arguments n equals ndims(X).

sort

For vectors, sort(X) sorts the elements of X in ascending order.
For matrices, sort(X) sorts each column of X in ascending order.
For N-D arrays, sort(X) sorts the along the first non-singleton
dimension of X. When X is a cell array of strings, sort(X) sorts
the strings in ASCII dictionary order.

Y = sort(X,DIM,MODE)
has two optional parameters.
DIM selects a dimension along which to sort.
MODE selects the direction of the sort 'ascend' results in ascending order
'descend' results in descending order.
The result is in Y which has the same shape and type as X.

[Y,I] = sort(X,DIM,MODE) also returns an index matrix I.
If X is a vector, then Y = X(I).
If X is an m-by-n matrix and DIM=1, then for j = 1:n, Y(:,j) = X(I(:,j),j); end

When X is complex, the elements are sorted by ABS(X). Complex
matches are further sorted by ANGLE(X).
When more than one element has the same value, the order of the elements are preserved in the sorted result and the indexes of equal elements will be ascending in any index matrix.

Example: If \( X = \begin{bmatrix} 3 & 7 & 5; & 0 & 4 & 2 \end{bmatrix} \)
then \( \text{sort}(X,1) \) is \( \begin{bmatrix} 0 & 4 & 2; & 3 & 7 & 5 \end{bmatrix} \) and \( \text{sort}(X,2) \) is \( \begin{bmatrix} 3 & 5 & 7; & 0 & 2 & 4 \end{bmatrix} \).

**sqrt**

\( \text{sqrt}(X) \) is the square root of the elements of \( X \). Complex results are produced if \( X \) is not positive.

**sum**

\( B = \text{sum}(A) \) returns sums along different dimensions of an array.
If \( A \) is a vector, \( \text{sum}(A) \) returns the sum of the elements.
If \( A \) is a matrix, \( \text{sum}(A) \) treats the columns of \( A \) as vectors, returning a row vector of the sums of each column.
If \( A \) is a multidimensional array, \( \text{sum}(A) \) treats the values along the first non-singleton dimension as vectors, returning an array of row vectors.
\( B = \text{sum}(A,dim) \) sums along the dimension of \( A \) specified by scalar \( dim \). The \( dim \) input is an integer value from 1 to \( N \), where \( N \) is the number of dimensions in \( A \). Set \( dim \) to 1 to compute the sum of each column, 2 to sum rows, etc.

**zeros**

\( B = \text{zeros}(n) \) returns an \( n \)-by-\( n \) matrix of zeros. An error message appears if \( n \) is not a scalar.
\( B = \text{zeros}(m,n) \) or \( B = \text{zeros}([m \ n]) \) returns an \( m \)-by-\( n \) matrix of zeros.
\( B = \text{zeros}(m,n,p,...) \) or \( B = \text{zeros}([m \ n \ p \ ...]) \) returns an \( m \)-by-\( n \)-by-\( p \)-by-\( ... \) array of zeros.