Introduction to programming in MATLAB

Dr. G.H.J. Lanel

Lecture 5

Outline

Outline

Arrays

- An array refers to a set of numbers or objects that will follow a specific pattern usually in rows and columns
- Each element of a array has an index
- Elements can be directly accessed using the index of the element

Arrays

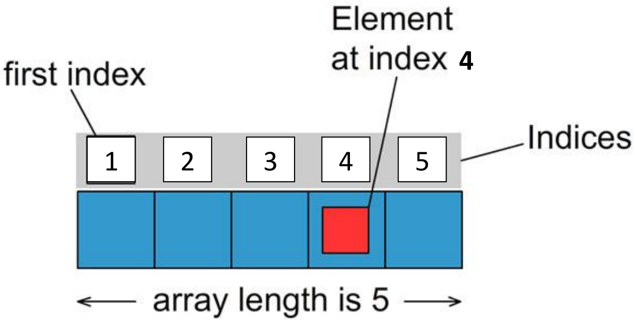
- An array refers to a set of numbers or objects that will follow a specific pattern usually in rows and columns
- Each element of a array has an index
- Elements can be directly accessed using the index of the element

Arrays

- An array refers to a set of numbers or objects that will follow a specific pattern usually in rows and columns
- Each element of a array has an index
- Elements can be directly accessed using the index of the element

Arrays

- An array refers to a set of numbers or objects that will follow a specific pattern usually in rows and columns
- Each element of a array has an index
- Elements can be directly accessed using the index of the element



Vectors and Matrices

- An array of dimension $1 \times n$ is called a **row vector**, whereas an array of dimension $m \times 1$ is called a **column vector.**
- A **matrix** is a two-dimensional array consisting of **m rows** and **n columns.**
- Elements of a matrix can be accessed using a pair of indices (i,j) where $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$

Vectors and Matrices

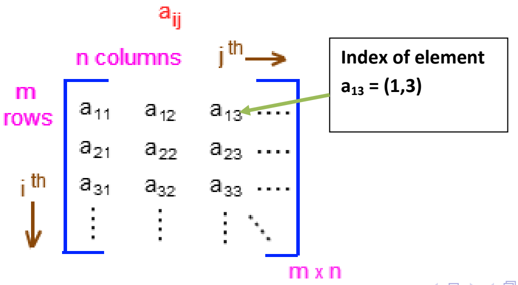
- An array of dimension $1 \times n$ is called a **row vector**, whereas an array of dimension $m \times 1$ is called a **column vector**.
- A **matrix** is a two-dimensional array consisting of **m rows** and **n columns**.
- Elements of a matrix can be accessed using a pair of indices (i,j) where $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$

Vectors and Matrices

- An array of dimension $1 \times n$ is called a **row vector**, whereas an array of dimension $m \times 1$ is called a **column vector.**
- A **matrix** is a two-dimensional array consisting of **m rows** and **n columns.**
- Elements of a matrix can be accessed using a pair of indices (i,j) where $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$

Vectors and Matrices

- An array of dimension $1 \times n$ is called a **row vector**, whereas an array of dimension $m \times 1$ is called a **column vector**.
- A **matrix** is a two-dimensional array consisting of **m rows** and **n columns**.
- Elements of a matrix can be accessed using a pair of indices (i,j) where $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$



Basic Operations on Arrays

- **Defining an array** : vectors or matrices can be defined as follows
  - » A = [5 7 2 1] or A = [1,2,3,4] % Defining a row vector
  - » B = [3;6;2;9] % Defining a column vector
  - » C = [7 5; 8 9] % Defining 2×2 dimensional matrix
- **Access elements in arrays** :
  - » A(3) % 3 rd element of the vector A
  - » B(2,1) % index (2,1) element of the matrix B
  - » B(1,:) % All elements of the 1st row in matrix B
  - » B(:,2) % All elements of the 2nd column in matrix B
- Rows of a matrix can also be entered as vectors using the notation for creating vectors with **constant spacing**, or the **linspace** command.
  - » D = [1:2:11 ; 0:5:25 ; linspace(10,60,6) ; 67 32 4 58 9 18]

Basic Operations on Arrays

- **Defining an array** : vectors or matrices can be defined as follows
  - » A = [5 7 2 1] or A = [1,2,3,4] % Defining a row vector
  - » B = [3;6;2;9] % Defining a column vector
  - » C = [7 5; 8 9] % Defining 2×2 dimensional matrix
- **Access elements in arrays** :
  - » A(3) % 3 rd element of the vector A
  - » B(2,1) % index (2,1) element of the matrix B
  - » B(1,:) % All elements of the 1st row in matrix B
  - » B(:,2) % All elements of the 2nd column in matrix B
- Rows of a matrix can also be entered as vectors using the notation for creating vectors with **constant spacing**, or the **linspace** command.
  - » D = [1:2:11 ; 0:5:25 ; linspace(10,60,6) ; 67 32 4 58 9 18]

Basic Operations on Arrays

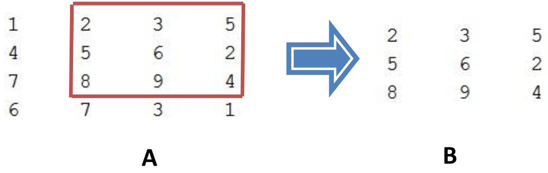
- **Defining an array** : vectors or matrices can be defined as follows
  - » A = [5 7 2 1] or A = [1,2,3,4] % Defining a row vector
  - » B = [3;6;2;9] % Defining a column vector
  - » C = [7 5; 8 9] % Defining 2×2 dimensional matrix
- **Access elements in arrays** :
  - » A(3) % 3 rd element of the vector A
  - » B(2,1) % index (2,1) element of the matrix B
  - » B(1,:) % All elements of the 1st row in matrix B
  - » B(:,2) % All elements of the 2nd column in matrix B
- Rows of a matrix can also be entered as vectors using the notation for creating vectors with **constant spacing**, or the **linspace** command.
  - » D = [1:2:11 ; 0:5:25 ; linspace(10,60,6) ; 67 32 4 58 9 18]

- **Deleting and inserting Elements** :
  - » B = [2 8 7 9 11 23 56 4 89 6];
  - » B(4) = 21; % insert 21 as 4th element
  - » B(3:6) = []; % remove elements from index 3 to 6
  - » B
- **Subset of an array** : subset of a vector or matrix can be obtained as follows
  - » A = [1 2 3 5; 4 5 6 2; 7 8 9 4;6 7 3 1]
  - » B = A(1:3,2:4) % subset of A

- **Deleting and inserting Elements** :
  - » B = [2 8 7 9 11 23 56 4 89 6];
  - » B(4) = 21; % insert 21 as 4th element
  - » B(3:6) = []; % remove elements from index 3 to 6
  - » B
- **Subset of an array** : subset of a vector or matrix can be obtained as follows
  - » A = [1 2 3 5; 4 5 6 2; 7 8 9 4;6 7 3 1]
  - » B = A(1:3,2:4) % subset of A

- **Deleting and inserting Elements** :
  - » B = [2 8 7 9 11 23 56 4 89 6];
  - » B(4) = 21; % insert 21 as 4th element
  - » B(3:6) = []; % remove elements from index 3 to 6
  - » B

- **Subset of an array** : subset of a vector or matrix can be obtained as follows
  - » A = [1 2 3 5; 4 5 6 2; 7 8 9 4;6 7 3 1]
  - » B = A(1:3,2:4) % subset of A



There are some useful elementary matrices in MATLAB

Elementary matrices

`eye(m,n)`	Returns an m-by-n matrix with 1 on the main diagonal
`eye(n)`	Returns an n-by-n square identity matrix
`zeros(m,n)`	Returns an m-by-n matrix of zeros
`ones(m,n)`	Returns an m-by-n matrix of ones
`diag(A)`	Extracts the diagonal of matrix A
`rand(m,n)`	Returns an m-by-n matrix of random numbers

Sometimes we have to perform arithmetic operations between the elements of two arrays of the same size in an element-by-element manner.

There are some useful elementary matrices in MATLAB

Elementary matrices

`eye(m,n)`	Returns an m-by-n matrix with 1 on the main diagonal
`eye(n)`	Returns an n-by-n square identity matrix
`zeros(m,n)`	Returns an m-by-n matrix of zeros
`ones(m,n)`	Returns an m-by-n matrix of ones
`diag(A)`	Extracts the diagonal of matrix A
`rand(m,n)`	Returns an m-by-n matrix of random numbers

Sometimes we have to perform arithmetic operations between the elements of two arrays of the same size in an element-by-element manner.

Summary of Array and Matrix operators

Character	Description
+ or -	Array **and** Matrix addition or subtraction of arrays
.*	Element-by-element multiplication of arrays
./	Element-by-element right division : a/b = `a(i,j)/b(i,j)`
.\	Element-by-element left division : a\b = `b(i,j)/a(i,j)`
.^	Element-by-element exponentiation
*	Matrix multiplication
/	Matrix right divide : `a/b = a*(b)`$^{-1}$
\	Matrix left divide (equation solve) : `a\b = (a)`$^{-1}$ `* b`
^	Matrix exponentiation

Outline

Functions

- Using functions to break down a large program to smaller and more manageable units is the heart of modular programming.
- In general, an m-file containing a Matlab function begins with the keyword **function** in the function header we specify the **name of the function** and the **input** and **output** parameters.

Functions

- Using functions to break down a large program to smaller and more manageable units is the heart of modular programming.
- In general, an m-file containing a Matlab function begins with the keyword **function** in the function header we specify the **name of the function** and the **input** and **output** parameters.

Functions

- Using functions to break down a large program to smaller and more manageable units is the heart of modular programming.
- In general, an m-file containing a Matlab function begins with the keyword **function** in the function header we specify the **name of the function** and the **input** and **output** parameters.

```
function [out1, out2] = funcName(in1, in2,in3)

out1 = in1+in2+in3;
out2 = out1/3;

end
```

Function Header

Function Body

- Functions can have multiple inputs and multiple outputs

Example of input and output arguments

`function C=FtoC(F)`	One input argument and one output argument
`function area=TrapArea(a,b,h)`	Three inputs and one output
`function [h,d]=motion(v,angle)`	Two inputs and two outputs

- function file must be saved by the function name
- Similarly as in Maple function can be called by function name

- Functions can have multiple inputs and multiple outputs

Example of input and output arguments

`function C=FtoC(F)`	One input argument and one output argument
`function area=TrapArea(a,b,h)`	Three inputs and one output
`function [h,d]=motion(v,angle)`	Two inputs and two outputs

- function file must be saved by the function name
- Similarly as in Maple function can be called by function name

- Functions can have multiple inputs and multiple outputs

Example of input and output arguments

`function C=FtoC(F)`	One input argument and one output argument
`function area=TrapArea(a,b,h)`	Three inputs and one output
`function [h,d]=motion(v,angle)`	Two inputs and two outputs

- function file must be saved by the function name
- Similarly as in Maple function can be called by function name

Sub Functions and Main Function

- Defining a main function and sub functions is important in divide and conquer approach
- Main function and sub functions can be implemented on separate M-files. But they should be saved in the same directory
- You can also implement main function and sub functions in the same M-file as follows

Sub Functions and Main Function

- Defining a main function and sub functions is important in divide and conquer approach
- Main function and sub functions can be implemented on separate M-files. But they should be saved in the same directory
- You can also implement main function and sub functions in the same M-file as follows

Sub Functions and Main Function

- Defining a main function and sub functions is important in divide and conquer approach
- Main function and sub functions can be implemented on separate M-files. But they should be saved in the same directory
- You can also implement main function and sub functions in the same M-file as follows

```
function [sm,avg] = addavg(x,y)   % Main Function

sm = addition(x,y);
avg = aver(x,y);
end

function a = aver(x,y)            % Sub Function 01

a = addition(x,y)/2;
end

function s = addition(x,y)        % Sub Function 02

s = x+y;
end
```

Local and Global variables

- The variables defined in a function are recognized only inside the function file.
- It is possible, however , to make a variable to be recognized in different function files. In other words to make the variables are **global**.
- Then they all share a single copy of that variable. Any change of value to that variable, in any function, is visible to all other functions

Local and Global variables

- The variables defined in a function are recognized only inside the function file.
- It is possible, however , to make a variable to be recognized in different function files. In other words to make the variables are **global**.
- Then they all share a single copy of that variable. Any change of value to that variable, in any function, is visible to all other functions

Local and Global variables

- The variables defined in a function are recognized only inside the function file.
- It is possible, however , to make a variable to be recognized in different function files. In other words to make the variables are **global**.
- Then they all share a single copy of that variable. Any change of value to that variable, in any function, is visible to all other functions

Inline Function

- Using inline function we can create a function without getting into edit window.
- Inline functions are created with the inline command in the following format.

  Name = inline('math expression typed as a string')

Examples

» FA = inline('$exp(x^2)/sqrt(x^2+5)$');

» FA

» FA(2)

» f = inline('$exp(x^2)/sqrt(x^2+y^2)$','x','y');

» f

» f(2,3)

Inline Function

- Using inline function we can create a function without getting into edit window.
- Inline functions are created with the inline command in the following format.

  Name = inline('math expression typed as a string')

Examples

» FA = inline('exp(x^2)/sqrt(x^2 + 5)');

» FA

» FA(2)

» f = inline('exp(x^2)/sqrt(x^2 + y^2)','x','y');

» f

» f(2,3)

Inline Function

- Using inline function we can create a function without getting into edit window.
- Inline functions are created with the inline command in the following format.

  Name = inline('math expression typed as a string')

Examples

» FA = inline($'exp(x^2)/sqrt(x^2+5)'$);

» FA

» FA(2)

» f = inline($'exp(x^2)/sqrt(x^2+y^2)','x','y'$);

» f

» f(2,3)

Inline Function

- Using inline function we can create a function without getting into edit window.
- Inline functions are created with the inline command in the following format.

  Name = inline('math expression typed as a string')

Examples

» FA = inline($'exp(x^2)/sqrt(x^2+5)'$);

» FA

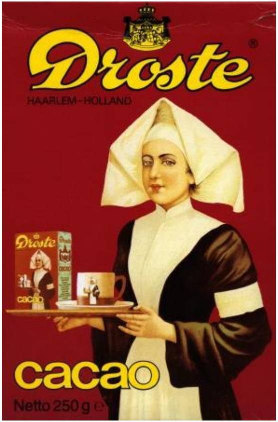
» FA(2)

» f = inline($'exp(x^2)/sqrt(x^2+y^2)','x','y'$);

» f

» f(2,3)

Recursion



Recursion is the process of repeating items in a self-similar way. The most common application of recursion is in mathematics and computer science, in which it refers to a method of defining functions in which the function being defined is applied within its own definition

Recursive Function

- An important class of functions are Recursive functions, function is said to be recursive if it calls itself in its own definition.
- Recursion is useful for computing the result of a function which can be expressed in terms of an integer (n) number of repetitive operations.
- For example, the sum of first n integers can be written as:

$$S(n) = 1 + 2 + 3 + ... + n \tag{1}$$

$$S(n) = S(n-1) + n \tag{2}$$

- The first equation shows a non-recursive way of calculating the sum of first (n) integers. This equation can be implemented using the familiar loops.
- The second equation defines a recursive formula for calculating the sum.

Recursive Function

- An important class of functions are Recursive functions, function is said to be recursive if it calls itself in its own definition.
- Recursion is useful for computing the result of a function which can be expressed in terms of an integer (n) number of repetitive operations.
- For example, the sum of first n integers can be written as:

$$S(n) = 1 + 2 + 3 + ... + n \tag{1}$$

$$S(n) = S(n-1) + n \tag{2}$$

- The first equation shows a non-recursive way of calculating the sum of first (n) integers. This equation can be implemented using the familiar loops.
- The second equation defines a recursive formula for calculating the sum.

Recursive Function

- An important class of functions are Recursive functions, function is said to be recursive if it calls itself in its own definition.
- Recursion is useful for computing the result of a function which can be expressed in terms of an integer (n) number of repetitive operations.
- For example, the sum of first n integers can be written as:

$$S(n) = 1 + 2 + 3 + ... + n \tag{1}$$

$$S(n) = S(n-1) + n \tag{2}$$

- The first equation shows a non-recursive way of calculating the sum of first (n) integers. This equation can be implemented using the familiar loops.
- The second equation defines a recursive formula for calculating the sum.

Recursive Function

- An important class of functions are Recursive functions, function is said to be recursive if it calls itself in its own definition.
- Recursion is useful for computing the result of a function which can be expressed in terms of an integer (n) number of repetitive operations.
- For example, the sum of first n integers can be written as:

$$S(n) = 1 + 2 + 3 + ... + n \tag{1}$$

$$S(n) = S(n-1) + n \tag{2}$$

- The first equation shows a non-recursive way of calculating the sum of first (n) integers. This equation can be implemented using the familiar loops.
- The second equation defines a recursive formula for calculating the sum.

Recursive Function

- An important class of functions are Recursive functions, function is said to be recursive if it calls itself in its own definition.
- Recursion is useful for computing the result of a function which can be expressed in terms of an integer (n) number of repetitive operations.
- For example, the sum of first n integers can be written as:

$$S(n) = 1 + 2 + 3 + ... + n \tag{1}$$

$$S(n) = S(n-1) + n \tag{2}$$

- The first equation shows a non-recursive way of calculating the sum of first (n) integers. This equation can be implemented using the familiar loops.
- The second equation defines a recursive formula for calculating the sum.

Example

Develop MATLAB function to calculate the sum of the first *n* integers using recursive formula

```
function [outsum] = sumrec(n)
if n<1
    error('Error : n must be positive\n');
elseif n==1
    outsum = 1;
else
    outsum = sumrec(n-1) + n; % recursive formula
end
```

Example

Develop MATLAB function to calculate the sum of the first *n* integers using recursive formula

```
function [outsum] = sumrec(n)
if n<1
    error('Error : n must be positive\n');
elseif n==1
    outsum = 1;
else
    outsum = sumrec(n-1) + n; % recursive formula
end
```

Example

Generating Fibonacci numbers : 0 1 1 2 3 5 8 13 21 ...
using recursive formula F(n) = F(n -1) + F(n - 2) ; F(0) = 0 and F(1) = 1

```
function [outfn] = fiborec(n)
if n<1
    error('Error : n must be positive\n');
elseif n==1
    outfn = 0;
elseif n==2
    outfn = [0 1];
else
    fnm1 = fiborec(n-1);
    outfn = fnm1(n-1) + fnm1(n-2);
    outfn = [fnm1 outfn];
end
```

Example

Generating Fibonacci numbers : 0 1 1 2 3 5 8 13 21 ...
using recursive formula F(n) = F(n -1) + F(n - 2) ; F(0) = 0 and F(1) = 1

```
function [outfn] = fiborec(n)
if n<1
    error('Error : n must be positive\n');
elseif n==1
    outfn = 0;
elseif n==2
    outfn = [0 1];
else
    fnm1 = fiborec(n-1);
    outfn = fnm1(n-1) + fnm1(n-2);
    outfn = [fnm1 outfn];
end
```

- Every recursive function must have a **terminating condition.** If the terminating condition is missing, then the recursive function would keep calling itself an infinite number of times.
- Recursive definitions are some times more important in programming than iterative definition since it is easier to write and debug complex problems.
- However if recursive algorithm is not much shorter than the non-recursive one, you should always go for the non-recursive(iterative) one.
- A well written iteration can be far more effective and efficient in such cases.

- Every recursive function must have a **terminating condition.** If the terminating condition is missing, then the recursive function would keep calling itself an infinite number of times.
- Recursive definitions are some times more important in programming than iterative definition since it is easier to write and debug complex problems.
- However if recursive algorithm is not much shorter than the non-recursive one, you should always go for the non-recursive(iterative) one.
- A well written iteration can be far more effective and efficient in such cases.

- Every recursive function must have a **terminating condition.** If the terminating condition is missing, then the recursive function would keep calling itself an infinite number of times.
- Recursive definitions are some times more important in programming than iterative definition since it is easier to write and debug complex problems.
- However if recursive algorithm is not much shorter than the non-recursive one, you should always go for the non-recursive(iterative) one.
- A well written iteration can be far more effective and efficient in such cases.

- Every recursive function must have a **terminating condition.** If the terminating condition is missing, then the recursive function would keep calling itself an infinite number of times.
- Recursive definitions are some times more important in programming than iterative definition since it is easier to write and debug complex problems.
- However if recursive algorithm is not much shorter than the non-recursive one, you should always go for the non-recursive(iterative) one.
- A well written iteration can be far more effective and efficient in such cases.

End!