

## MATLAB Primer

Chen Yu  
Indiana University

---

---

---

---

---

---

---

---

## MATLAB as a calculator

- Built in functions  
e.g. `log(100)`; `log10(100)`; `log2(256)`;  
Toolbox: statistics, image processing, signal processing, Wavelet, control...  
`>> help`  
`>> help stats`  
`rand()`, `randperm()`;
- Built-in Variables  
`pi`, `sin(pi/6)`

---

---

---

---

---

---

---

---

## The Real Power of MATLAB

- e.g. solving linear equations in one step

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 1 \\ 3 & 5 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 7 \end{bmatrix}$$

```
>> A = [ 0 1 2; 1 2 1; 3 5 2]
>> b = [ 1; 3; 7]
>> x = A\b;
```

---

---

---

---

---

---

---

---

## Programming Environment

- Three windows: command window, workspace and command history
- Three ways to get help:
  - >> help sqrt
  - >> lookfor sqrt
  - >> doc sqrt

---

---

---

---

---

---

---

---

## All MATLAB variables are matrices

- “MATrix LABoratory”
- A vector is a matrix with one row or one column
- a scalar is a matrix with one row and one column
- a character string is a row of column vector of characters

---

---

---

---

---

---

---

---

## Assigning Values to Variables

- >> A = [ 3, 2; 3 1; 1 5]
- >> x = [ 5 ;6 ; 7]
- >> y = [ 1 2 3]
- >>A(4,4) = 100 Beyond the existing dimensions of the matrix?  
good or bad compared with other programming languages?

---

---

---

---

---

---

---

---

## Using functions to create matrices and vectors

- linspace  
`>> u = linspace(0,5,5)`  
How to create column vectors?
- Create matrices  
ones, zeros, eye, diag  
A closer look at `diag()`.

---

---

---

---

---

---

---

---

## Notation

- Subscript notation
- Colon notation  
`>> s = 1 : 4`  
`>> s = 1 :0.5: 4`
- Use colon notation to refer to subsets of columns and rows  
`>> A(1:2,1)`  
`>> A(:,1)`

---

---

---

---

---

---

---

---

## String Operations

- String are matrices with character elements.
- String constants are enclosed in single quotes.  
`>> myfirst= 'Chen';`  
`>> mylast = 'yu';`  
`>> myname = [ myfirst, ' ', mylast];`  
`length(myname)?`  
`myname(1:3)?`

---

---

---

---

---

---

---

---

## String Functions

- Compare two strings to see if they are exactly the same  
`strcmp('abc', 'abd');`
- Search for a substring with a longer string  
`findstr('abcdef', 'def');`  
`findstr('abcdefdfadfadfdef', 'def');`
- help string  
`strcat('chen', 'yu');`

---

---

---

---

---

---

---

---

## Built-in Functions

- Scalar function  
`sin cos tan exp log sqrt floor ceil abs`
- Vector functions  
`max min sum sort median mean`  
They also work for matrices.
- Matrix functions  
`eig inv size rank`

---

---

---

---

---

---

---

---

## Working space

- `who/whos/clear`
- `save`
- `Path`

---

---

---

---

---

---

---

---

## Vectorization

- Use vectorized indexing and logical functions
- Use vector operations instead of loops  
e.g.  $C = A ./3$ ;  
 $A.^2$  vs.  $A .^2$
- Pre-allocate memory for vectors and matrices

---

---

---

---

---

---

---

---

## Vectorized Indexing

- $x = [0.2, 0.55, 0.3, 3, 0.7, 1]$ ;  
 $\text{index} = \text{find}(x > 0.5)$   
 $x(\text{index})$
- $\text{find}(x == 1)$
- $x(\text{find}(\text{abs}(x - 1.5) > 0.5))$

---

---

---

---

---

---

---

---

## Why it is important?

Problem: construct an 50000 entry array  $x(i) = \frac{i}{\sin(i) + \cos(i)}$

Method 1:

```
clear v;  
t0 = cputime;  
imax = 50000;  
for i = 1 : imax  
    v(i) = i/(sin(i)+cos(i));  
end;  
t1 = cputime;  
fprintf('the running time:%6.3f sec', t1-t0);
```

---

---

---

---

---

---

---

---

## Why it is important?

### Method 2:

```
clear v;  
t0 = cputime;  
v = zeros(1,imax);  
imax = 50000;  
for i = 1 : imax  
    v(i) = i/(sin(i)+cos(i));  
end;  
t1 = cputime;  
fprintf('the running  
time:%6.3f sec', t1-t0);
```

### Method 3:

```
clear v;  
t0 = cputime;  
v = zeros(1,imax);  
i = 1 : imax;  
v = i./(sin(i) + cos(i));  
t1 = cputime;  
fprintf('the running time:%6.3f  
sec', t1-t0);
```

Comparing the times in these three methods

---

---

---

---

---

---

---

---

## Control Flow

- The control-flow statements of a programming language specify the order in which computations are performed.

---

---

---

---

---

---

---

---

## For Loop

- All elements in a vector or matrix have been processed

```
x = 1 : 5  
sumx = 0  
for k = 1 : length(x)  
    sumx = sumx + x(k);  
end;
```

```
for k = 1 : 2: n    % incremented by 2  
for k = n : -1 : 1 % count down  
for x = 0 : pi/15:pi % non-integer increments
```

---

---

---

---

---

---

---

---

## While Loop

- When an iteration is repeated until some termination criterion is met.  
it = 0; maxit = 10;  
while it < maxit  
    it = it + 1;  
end;
- Other ways to exit from a loop construct  
break  
return

---

---

---

---

---

---

---

---

## Condition Execution

```
if x > 0  
    disp('x is positive')  
elseif x < 0  
    disp('x is negative');  
else  
    disp('x is exactly zero');  
end;
```

---

---

---

---

---

---

---

---

## Switch Construct

```
switch color  
    case 'red'  
        disp(' color is red');  
    case 'blue'  
        disp(' color is blue');  
    case 'green'  
        disp(' color is green');  
    otherwise  
        disp(' color is not red, green or blue');  
end;
```

---

---

---

---

---

---

---

---

## Function

- Functions break large computing tasks into smaller ones, and enable people to build on what others have done instead of starting over from scratch.
- Appropriate functions hide details of operation from part of the program that don't need to know about them.

---

---

---

---

---

---

---

---

## Function

- Use local variables that exist only while the function is executing. Local variables are distinct from variables of the same names in the workspace.
- Function files are reusable.
- Specific tasks can be encapsulated into functions. This modular approach enables development of structured solutions to complex problems.

---

---

---

---

---

---

---

---

## Function

- Function [argouts] = funName(argins)  
Variable numbers of input/pout parameters
- Each function has internal variables, nargin and nargout can find how many input/output arguments.
- Allow a single function to perform multiple related tasks.
- All functions assume default values for some inputs, thereby simplifying the use of the function for some tasks.

---

---

---

---

---

---

---

---

## Function

```
function [a, b] = myfun(x,y)
% this function does
% input variables:
% x .....
% y .....
%output variables
% a .....
% b .....
```

---

---

---

---

---

---

---

---

## A Case Study

- Problem: Building a simulation program which will receive the same input like human subjects and do the same test.
- Main structure

```
[traindata] = load_training_data(inputfile);
[testdata] = load_testing_data(inputfile);

[trainresult] = do_train(traindata);
[simresult] = do_test(trainresult, testdata);

[humanresult] = load_human_data(inputfile);
cal_stats(simresult, humanresult);
```

---

---

---

---

---

---

---

---

## Programming Styles

- Programming style varies from programmer to programmer. It is good to stick to common coding practices.
- A consistent programming style gives your programs a visual familiarity that helps the reader quickly comprehend the intention of the code.
- A programming style consists of
  - visual appearance
  - conventions used for variable names
  - documentation with comment statements

---

---

---

---

---

---

---

---

## Programming Styles

- Indent switch and loop blocks
- Use meaningful variable names
- In-line comments %

---

---

---

---

---

---

---

---

## Programming Styles

- The best way is to assemble a whole program from well designed small functions. This will enhance readability and testing.
- A function interacts with other code through input and output arguments and global variables. The use of arguments is almost always clearer than the use of globals.
- Any block of code appearing in more than one m-file should be considered for packaging as function.

---

---

---

---

---

---

---

---

## Programming Styles

- Each function can just do one task and should occupy roughly one screen. If your function is much longer, then you may want to consider to splitting it into two or more separate functions.
- Every 5-10 lines of code should be accompanied by comments. However, commenting trivial operations such as incrementing of index variables is not necessary.

---

---

---

---

---

---

---

---