Chapter 5: User-Defined Functions (Cont.)

Yunong Zhang (张雨浓)

Email: zhynong@mail.sysu.edu.cn

Optional Arguments

```
plot(x,y);

plot(x,y1,x,y2);

plot(x,y,'r--',x,y,'bo');

plot(x,y,‘ro','LineWidth',3.0,'Mar
kerSize',8,'MarkerEdgeColor','b','
MarkerFaceColor','g');
```

```
a=max(x);
[m n]=max(x);
```

Optional Arguments (Cont.)

- Eight special functions are used to get information about their optional arguments and to report errors in those arguments

Optional Arguments (Cont.)

- nargin

  return the number of actual input arguments that were used to call the function

- nargout

  return the number of actual output arguments (results) that were generated by calling the function

Remember as n-arg-in and n-arg-out!!

Optional Arguments (Cont.)

- nargchk

  return a standard error message when a function is called with too few or too many (input) arguments

message=nargchk(min_args,max_args, num_args)

min_args: The minimum number of arguments

max_args: The maximum number of arguments

num_args: The actual number of arguments

Optional Arguments (Cont.)

- An **empty** string is returned if the number of arguments is within acceptable limits
- A **standard error message** is produced if the number of arguments is outside the acceptable limits

Optional Arguments (Cont.)

```
>> message=nargchk(0,3,2)
message = []
>> message=nargchk(0,3,4)
message = Too many input arguments.
>> message=nargchk(1,3,0)
message = Not enough input arguments.
```

Optional Arguments (Cont.)

- error

  display an error message and abort the user-defined function which caused the error.

  error('message')

  where 'message' is the error-message string.

  Doing **nothing** if the message string is **empty**; otherwise, **halting** the function and **displaying** the error message in the Command Window.

Optional Arguments (Cont.)

Command Window

```
>> error( 'message' )
??? error( 'message' )
          |
Error: Missing variable or function.

>> error('message')
??? message

>> |
```

Workspace

Stack: Base

Name	Size	Bytes	Class
abc message	1x27	54	char array

Optional Arguments (Cont.)

- warning

  Display a warning message but continue the function execution

  warning(‘message’)

  where ‘message’ is the warning-message string.

  Doing **nothing** if the message string is **empty**; otherwise, **Displaying** the warning message in the Command Window, and **listing** the function name and line number where the warning came from.

Optional Arguments (Cont.)

- inputname

  return the **actual** name of the variable that corresponds to a particular argument number

  name=inputname(argno)

  where argno denotes the number (actually, index) of the argument.

  The argument name is returned if the argument is a variable. An empty string is returned if the argument is an expression.

Optional Arguments (Cont.)

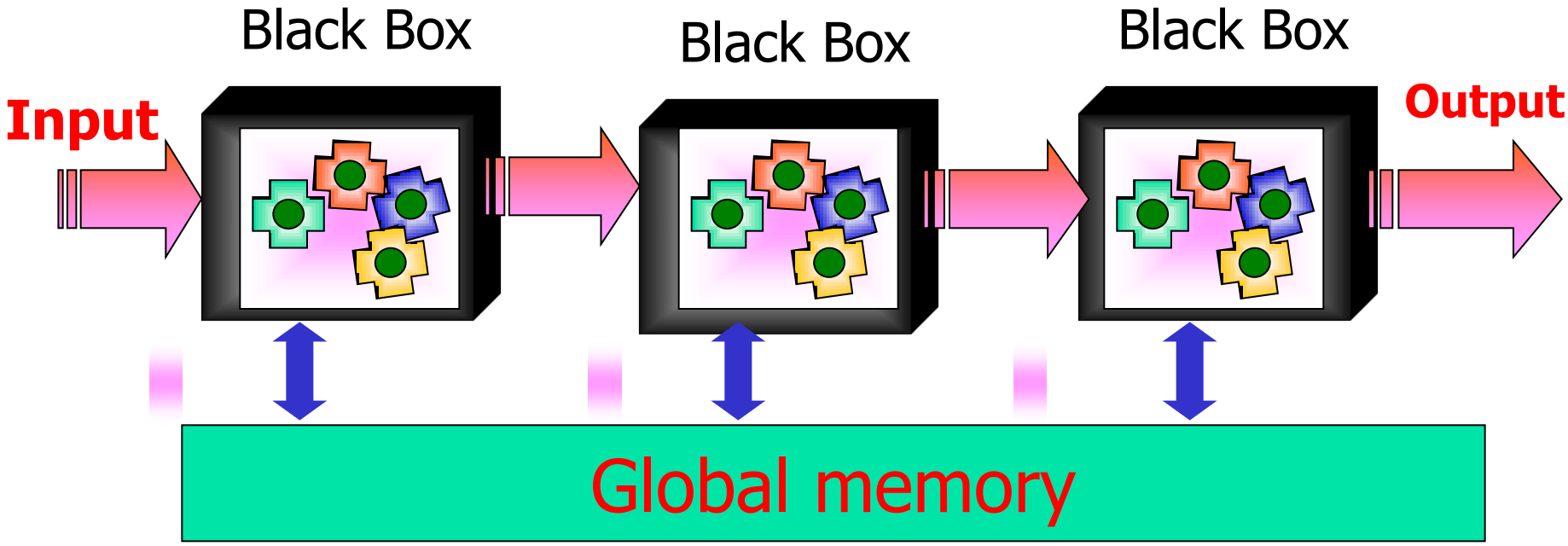
```
function dis=distance(x1,y1,x2,y2)
%Calculate the distance between two points
%where A(x1,y1) and B(x2,y2)

dis=sqrt((x2-x1)^2+(y2-y1)^2);

name=inputname(2);
disp(['The name of the second actual argument is: ',name]);
```

```
result=distance(ax,ay,bx,by);
```

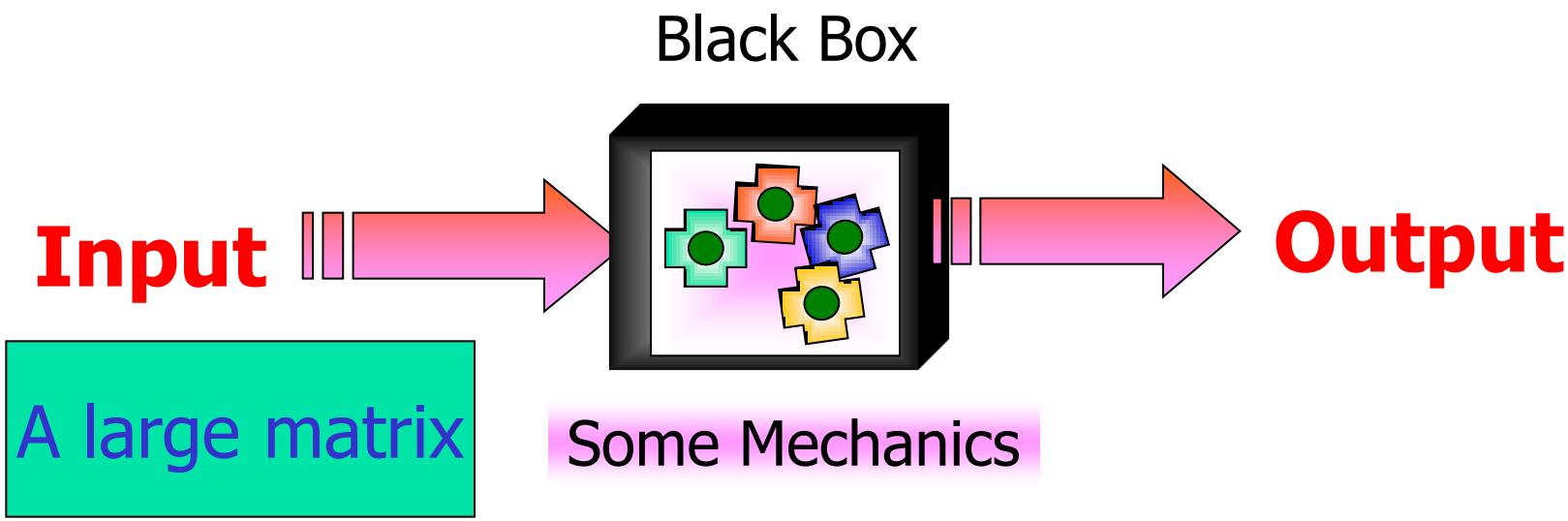
The name of the second actual argument is: ay

Sharing data using global memory

- Each function has its own workspace
- local function



Sharing data using global memory (Cont.)



Less efficient / Inefficient Way!

The time cost is high to exchange data when the function is called.

Sharing data using global memory (Cont.)

- Global variable

  global var1, var2, var3 ....

- Global variables **must** be declared **before** they are used for the first time in a function
- A warning message is displayed in the Command Window if declaring a variable to be global **after** it has already been created

Sharing data using global memory (cont.)

p210.m

```
a=1;
global a;
```

```
>> Warning: The value of local variables may have been
changed to match the globals.  Future versions of MATLAB
will require that you declare a variable to be global before
you use that variable.
> In C:\MATLAB6p5\work\matlab_programming\p210.m
at line 2
```

Example: Random Number Generator

- To generate a noise-like data
- Scientific research and engineering applications, such as, testing of some devices, design of secure communications, and so on.

欢迎就此阅读各文献、编程、测试和展示自己结果

Example: Random Number Generator (cont.)

- A simple random number generator algorithm:

$$n_{i+1} = \mathrm{mod}(8121\, n_i + 28411, 134456)$$

n_i: a non-negative integer

mod: modulus function

n_{i+1}: [0, 134455]

$$n_0, n_1, n_2, n_3, \ldots, n_i, \ldots$$

Any given number appears with an equal probability:
Uniform distribution

Example: Random Number Generator (cont.)

- Generate a random number in the range [0,1) based on the equation

$$ran_i = n_i / 134456$$

1. *State the problem*

Write a function that can generate and return an array containing numbers with a uniform probability distribution in the range [0,1).

Example: Random Number Generator (cont.)

Note:

The function can have one or two input-arguments (n, m) specifying the size of the array to return.

Generate a square array of size n*n if there is only one argument.

Generate an array of size n*m if there are two arguments.

Example: Random Number Generator (cont.)

- *2. Define the inputs and outputs*

  Two functions:

  seed: Initialize the random sequence n_0

  Input: an integer to serve as the starting point of the sequence

  Output: No output

  random0: Generate one or more random numbers

  Input: n, m

  Output: array of random values in the range [0,1)

?

Example: Random Number Generator (cont.)

3. *Describe the algorithm*

Generate a seed for function *seed*

Generate a random array with dimension n*n or n*m

Output the generated random array

Example: Random Number Generator (cont.)

- *4. Turn the algorithm into Matlab statements*

```
function seed(original_seed)
    global ISEED;
    msg=nargchk(1,1,nargin);
    error(msg);
    original_seed=round(original_seed);
    ISEED=abs(original_seed);
```

改变环境或全局变量！

Example: Random Number Generator (cont.)

- **function ran_array=random0(n,m)**

```
msg=nargchk(1,2,nargin);
error(msg);
if nargin<2
  m=n;
end
```

Any mistake here?

Example: Random Number Generator (cont.)

```
ran_array=zeros(n,m);
for i=1:n
    for j=1:m
        ISEED=mod(8121*ISEED+28411,134456);
        ran_array(i,j)=ISEED/134456;
    end
end
```

Example: Random Number Generator (cont.)

- *5. Test the resulting Matlab program:*

```
>>seed(100)
>>random0(4)

>>seed(5)
>>random0(3,7)

>>seed(10,20)  ?
```

```
>> zynseed(1000)
>> random0(4)
??? Undefined function or variable 'ISEED'.

Error in ==> C:\MATLAB6p1\work\random0.m
On line 10  ==>            ISEED=mod(8121*ISEED+28411,134456);
```

C:\MATLAB6p1\work\random0.m

File Edit View Text Debug Breakpoints Web Window Help

```
function ran_array=random0(n,m)
global ISEED;
msg=nargchk(1,2,nargin);
error(msg);
```

zynseed.m random0.m

Ready

```
>> random0(4)

ans =

    0.8491    0.5356    0.7806    0.7445
    0.6285    0.1030    0.7366    0.4244
    0.8165    0.7466    0.6610    0.0464
    0.8591    0.9534    0.5099    0.4647

>>
```

Research on it!!

Preserving data between calls to a function

- All local variables within a function will disappear when this function finishes executing
- It is sometimes useful to preserve some local information within a function between calls to the function

- persistent var1 var2 var3 ...

Preserving data between calls to a function (cont.)

- p220.m

```
a=0;
persistent_test(a);

a=1;
persistent_test(a);
```

Preserving data between calls to a function (cont.)

```
function persistent_test(a)
if a==0
    b=0;
else
    b=b+1;
end
b
```

Preserving data between calls to a function (cont.)

- >>p220 **?**

Preserving data between calls to a function (cont.)

```
b =

     0

??? Undefined function or variable 'b'.

Error in ==> C:\MATLAB6p5\work\persistent_test.m
On line 6  ==>     b=b+1;

Error in ==> C:\MATLAB6p5\work\p220.m
On line 5  ==> persistent_test(a);
```

Preserving data between calls to a function (cont.)

```
function persistent_test(a)
persistent b;
if a==0
    b=0;
else
    b=b+1;
end
b
```

Preserving data between calls to a function (cont.)

- >>p220

```
b =

     0

b =

     1
```

Additional Tests

```
>> clear; clc;
>> persistent_test(1)
b = []
>> persistent_test(2)
b = []
>> persistent_test(3)
b = []
>> persistent_test(0)
b = 0
>> persistent_test(1)
b = 1
>> persistent_test(33)
b = 2
>> persistent_test(33)
b = 3
>>
```

Sincere Thanks!

- Using this group of PPTs, please read

- [1] Yunong Zhang, Weimu Ma, Xiao-Dong Li, Hong-Zhou Tan, Ke Chen, MATLAB Simulink modeling and simulation of LVI-based primal-dual neural network for solving linear and quadratic programs, Neurocomputing 72 (2009) 1679-1687

- [2] Yunong Zhang, Chenfu Yi, Weimu Ma, Simulation and verification of Zhang neural network for online time-varying matrix inversion, Simulation Modelling Practice and Theory 17 (2009) 1603-1617