Chapter 7: Sparse Arrays, Cell Arrays, and Structures

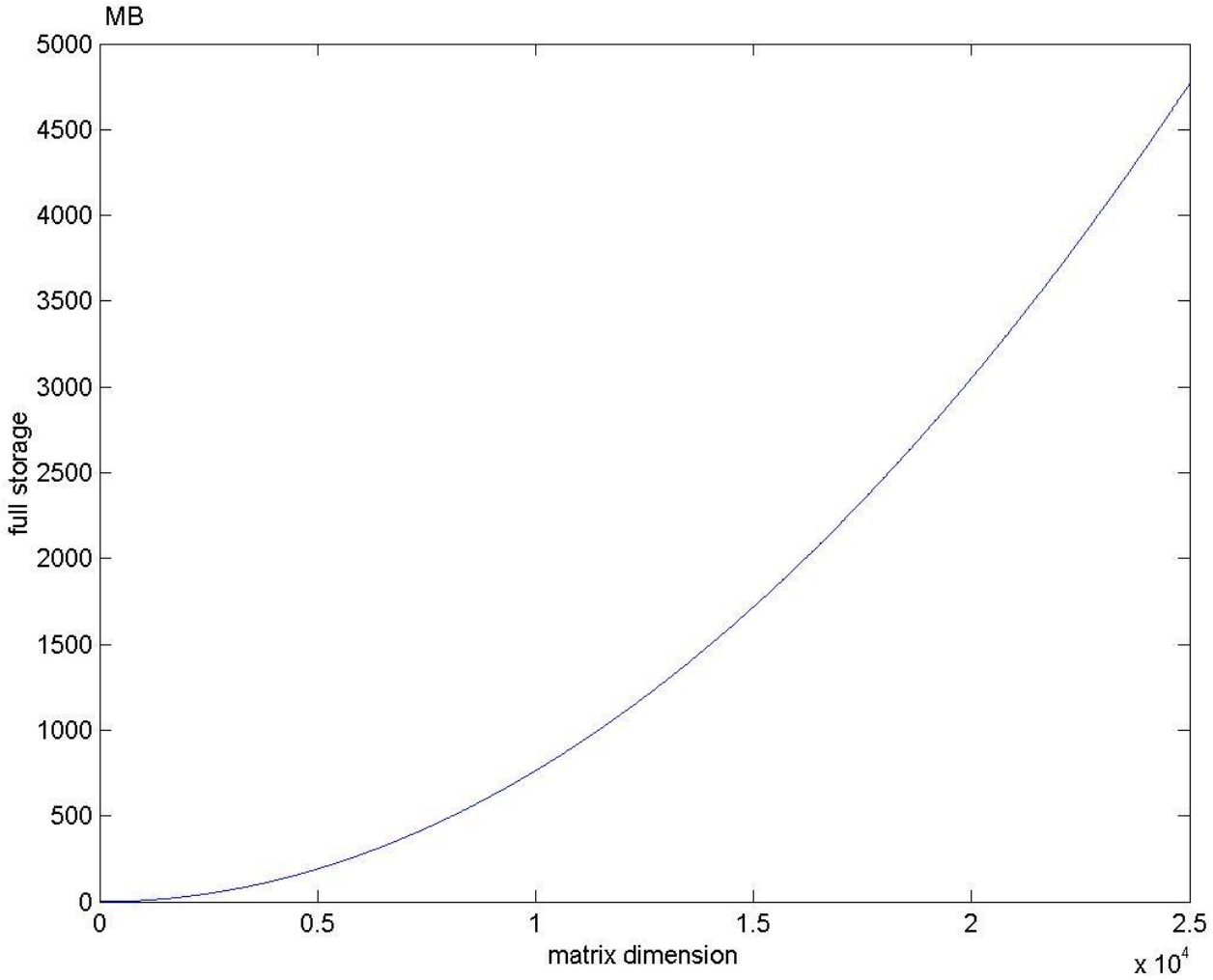
Yunong Zhang (张雨浓)

Email: zhynong@mail.sysu.edu.cn

Why introduce Sparse Arrays?

```
>> x=1:10:25000;
>> y=((x.^2*8)/1024)/1024;% mb
>> plot(x,y)
>> text(0,5000,'MB')
>> ylabel('full storage')
>> xlabel('matrix dimension')
>> y(end)          ➔  ans = 4764.9
>> y(end)/1024     ➔  ans = 4.6533
```

Full matrix storage



memory:
25000*25000
*8 bytes
=4765MB
=4.65GB

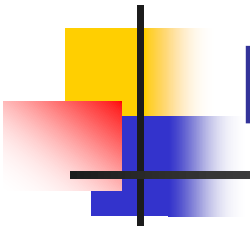
Matrix / Array Storage

In standard matrix implementations,

- the **computational complexity is O(N^3) operations,**
- the **storage complexity is O(N^2) operations**.

Table 1: Memory requirement of a matrix in double precision versus its dimension N

Dimension N	1000	3000	5000	7000	...	9000	11000	13000	15000	17000
Storage (MB)	7.7	68.7	190.8	373.9	...	618.0	923.2	1289.4	1716.6	2204.9



Matrix / Array Storage (Cont.)

1) What happens if we don't have enough memory storage?

2) **Virtual memory**?

3) What could we do?

A report

Sparse Arrays

- [0 1 0 0 0 0 0 0 0 0;
  1 0 0 0 1 0 0 0 0 0;
  0 0 0 1 0 0 0 0 0 0;
  0 0 1 0 0 0 0 0 1 0;
  0 1 0 0 0 1 0 0 0 0;
  0 0 0 0 1 0 0 0 0 0;
  0 0 0 0 0 0 0 1 0 1;
  0 0 0 0 0 0 1 0 0 0;
  0 0 0 1 0 0 0 0 0 0;
  0 0 0 0 0 0 1 0 0 0]

10*10=100 elements

value 1: 14 elements
value 0: 86 elements

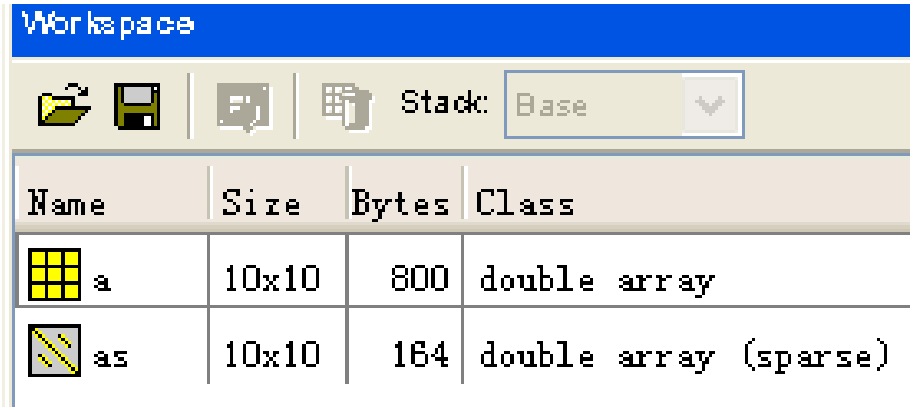
Sparse data type:
three values for each nonzero element are saved:
value of the element;
the row and column **numbers** where the element is located

Sparse Arrays (Cont.)

```
>> a=eye(10)
a =

     1     0     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0     0
     0     0     1     0     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     0     0
     0     0     0     0     1     0     0     0     0     0
     0     0     0     0     0     1     0     0     0     0
     0     0     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     0     0     1
```

Sparse Arrays (Cont.)

```
>> as=sparse(a)

as =

   (1,1)        1
   (2,2)        1
   (3,3)        1
   (4,4)        1
   (5,5)        1
   (6,6)        1
   (7,7)        1
   (8,8)        1
   (9,9)        1
  (10,10)       1
```



压缩了5倍：原空间的20%就够了

Generating Sparse Matrices

```
>> a=speye(5)

a =

   (1,1)        1
   (2,2)        1
   (3,3)        1
   (4,4)        1
   (5,5)        1
```

```
>> b=full(a)

b =

     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
```

Working with Sparse Matrices

```
>> a=speye(5)

a =

   (1,1)        1
   (2,2)        1
   (3,3)        1
   (4,4)        1
   (5,5)        1
```

```
>> b=full(a)

b =

     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
```

Working with Sparse Matrices (Cont.)

```
>> a(2,1)=2

a =

   (1,1)        1
   (2,1)        2
   (2,2)        1
   (3,3)        1
   (4,4)        1
   (5,5)        1
```

```
>> c=a+2*b

c =

     3     0     0     0     0
     2     3     0     0     0
     0     0     3     0     0
     0     0     0     3     0
     0     0     0     0     3
```

full matrix

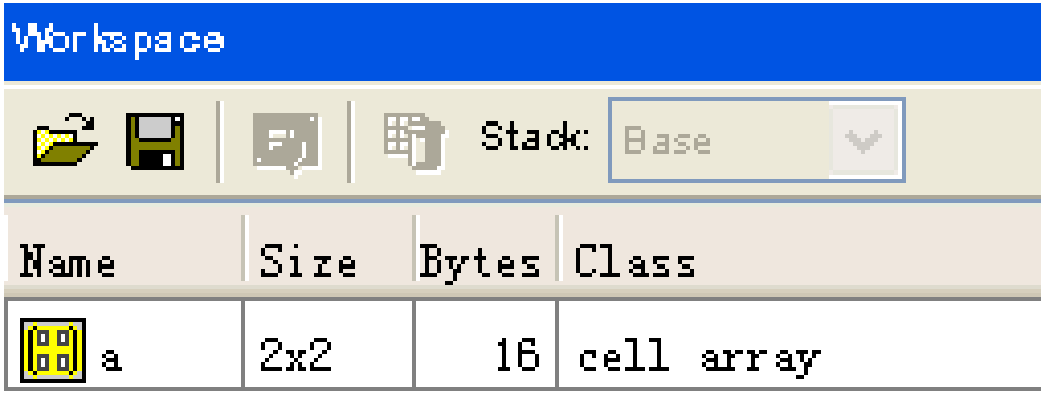
Cell Arrays

- Each element of a cell array is a pointer to another data structure, and those data structures can be of different types.

cell 1,1 'project name: Computer networks'	cell 1,2 'PI: Lisa'
cell 2,1 ['J.F.Mends'; 'A. L. Albert' ...]	cell 2,2 [2000; 3000; ...]

Creating Cell Arrays

a=cell(2,2)




```
>> a=cell(2,2)

a =

    []    []
    []    []
```

Creating Cell Arrays (Cont.)

```
>> a{1,1}=[1 2 3; 4 5 6; 7 8 9];
>> a{1,2}='Matlab Programming';
>> a{2,1}=[3+4i 5; 4+6i 7];
>> a{2,2}=[];
```

a{m,n}: set/display the values of cell a at location (m,n)

a(m,n): set/display the data structure of cell a at location (m,n)

Parentheses ( ), Brackets [ ], Braces { }

>> help .

Operators and special characters.

Arithmetic operators.

plus	- Plus	+
uplus	- Unary plus	+
minus	- Minus	-
uminus	- Unary minus	-
mtimes	- Matrix multiply	*
times	- Array multiply	.*
mpower	- Matrix power	^
power	- Array power	.^
mldivide	- Backslash or left matrix divide	\
mrdivide	- Slash or right matrix divide	/
ldivide	- Left array divide	.\
rdivide	- Right array divide	./
kron	- Kronecker tensor product	kron

Relational operators.

eq	- Equal	==
ne	- Not equal	~=
lt	- Less than	<
gt	- Greater than	>
le	- Less than or equal	<=
ge	- Greater than or equal	>=

Logical operators.

and	- Logical AND	&
or	- Logical OR	|
not	- Logical NOT	~
xor	- Logical EXCLUSIVE OR	
any	- True if any element of vector is nonzero	
all	- True if all elements of vector are nonzero	

Special characters.

colon	- Colon	:

Parentheses ( )

Brackets [ ]

Braces { }

punct	- Function handle creation	@
punct	- Decimal point	.
punct	- Structure field access	.
punct	- Parent directory	..
punct	- Continuation	...
punct	- Separator	,
punct	- Semicolon	;
punct	- Comment	%
punct	- Invoke operating system command	!
punct	- Assignment	=
punct	- Quote	'
transpose	- Transpose	.'
ctranspose	- Complex conjugate transpose	'
horzcat	- Horizontal concatenation	[,]
vertcat	- Vertical concatenation	[;]
subsasgn	- Subscripted assignment	( ),{ },.
subsref	- Subscripted reference	( ),{ },.

Bitwise operators.

bitand	- Bit-wise AND.
bitcmp	- Complement bits.
bitor	- Bit-wise OR.
bitmax	- Maximum floating point integer.
bitxor	- Bit-wise XOR.
bitset	- Set bit.
bitget	- Get bit.
bitshift	- Bit-wise shift.

Set operators.

union	- Set union.
unique	- Set unique.
intersect	- Set intersection.
setdiff	- Set difference.
setxor	- Set exclusive-or.
ismember	- True for set member.

Viewing the Cell Arrays

```
>> a{1,1}

ans =

     1     2     3
     4     5     6
     7     8     9
```

```
>> a(1,1)

ans =

  [3x3 double]
```

Viewing the Cell Arrays(Cont.)

```
>> a

a =

    [3x3 double]    'Matlab Programming'
    [2x2 double]                []
```

Displays the data structures in each element of a cell array in a condensed form that limits each data structure to a single line.

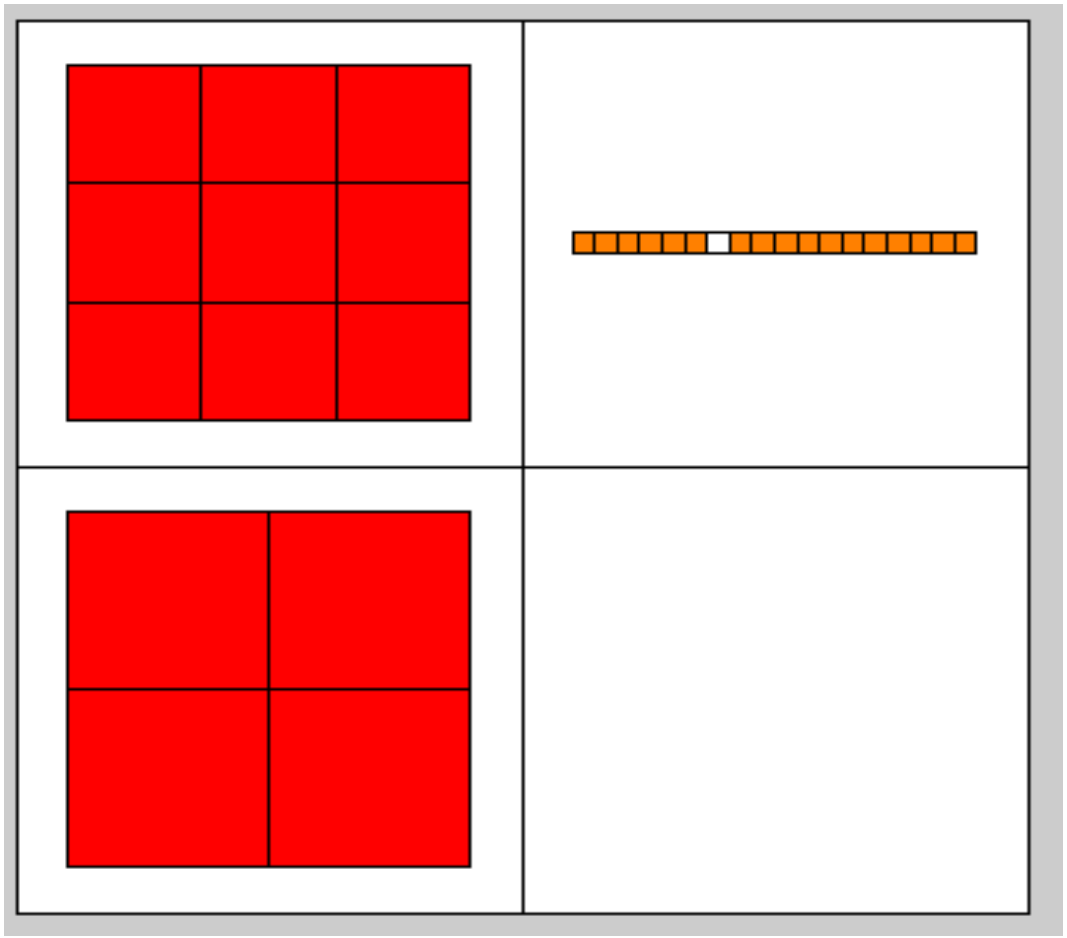

输入: condensed juice
索引
condense
condensed
condensed aromatic
condensed balance sheet
condensed buttermilk
condensed film
condensed fish soluble
condensed fluid
condensed gas
condensed income statemer
condensed instruction deck
condensed juice
condensed milk
condensed monolayer
condensed nucleus
简明英汉词典
condensed juice
浓缩(果)汁
condensed phase
condensed phosphate
condensed profile
condensed section
condensed spark
condensed specifications
condensed steam
condensed tannin extracts
condensed type
condensed water outlet
condensed water removal
condensed whey
condenser
condenser armature
condenser auxiliary
condenser bank
condenser block
condenser bolometer
condenser box
condenser bracket
condenser capacity
简明英汉词典
condenser capacity
(1) 电容器电容量
(2) 冷凝器容量
condenser coil
condenser component
condenser coupling
condenser current
condenser current compens
condenser damping
condenser discharge
condenser divider
condenser duty
condenser excitation
condenser expansion joint
condenser flange
condenser foil
condenser gasket

Viewing the Cell Arrays(Cont.)

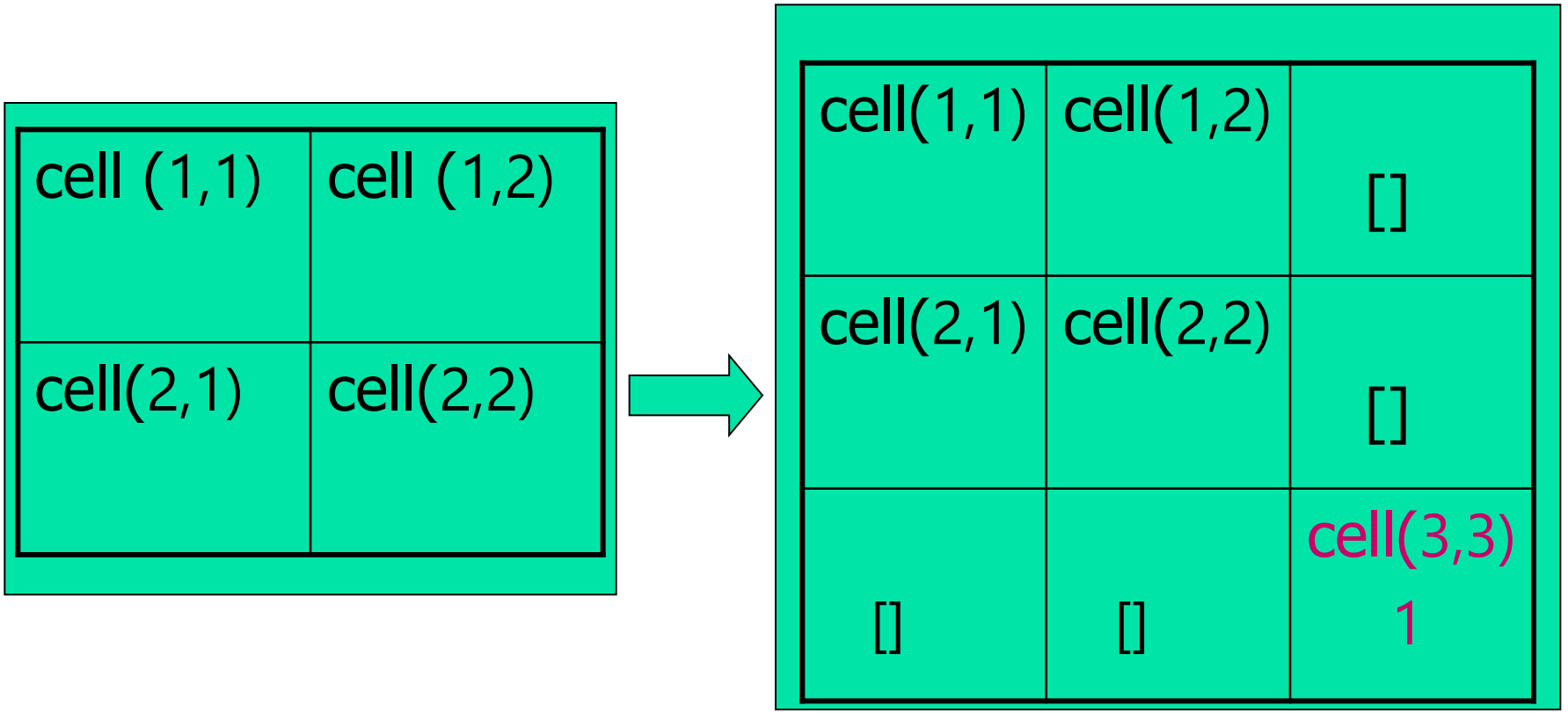
```
>> celldisp(a)
a{1,1} =

    1    2    3
    4    5    6
    7    8    9

 a{2,1} =

                3 +   4i                    5
                4 +   6i                    7

a{1,2} =

Matlab Programming

 a{2,2} =

    []
```

Viewing the Cell Arrays(Cont.)

```
>> cellplot(a)
```



Extending Cell Arrays



cell(3,3)=1

Extending Cell Arrays (Cont.)

```
a=cell(3,3);
for i=1:3
    for j=1:3
        a{i,j}=[i,j];
    end
end
```

(a)

```
for i=1:3
    for j=1:3
        a{i,j}=[i,j];
    end
end
```

(b)

Extending Cell Arrays (Cont.)

Recall the learned knowledge!

CASE 1:

```
square=zeros(1,100);
for i=1:100
    square(i)=i^2;
end
```

CASE 2:

```
for i=1:100
    square(i)=i^2;
end
```

In CASE 2, the vector square has different size at different time. At each time, Matlab has to

1) create a new vector;
2) copy the contents of the old array to the new longer array;
3) add the new value to the array;
4) delete the old array.

Using Data in Cell Arrays

cell(1,1) [1 2 3; 4 5 6; 7 8 9]	cell(1,2) 'Matlab Programming'
cell(2,1) [3 + 4i 5 4 + 6i 7]	cell(2,2) []

```
>> a{1,1}
ans =

     1     2     3
     4     5     6
     7     8     9

>> a{1,1}(2,3)

ans =

     6
```

Significance of Cell Arrays Application example:

```
x=0:pi/100:2*pi;
y=sin(2*x);

plot(x,y);

plot(x,y,'-ro');

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

Significance of Cell Arrays (Cont.)

- A variable number of input arguments
- Arguments may have different data types

```
plot(x,y);
cell a <== x  y
plot(x,y,'-ro');
cell a <== x  y  'String'
plot(x,y,'-ro','LineWidth',3.0,'MarkerSize',
8,'MarkerEdgeColor','b','MarkerFaceColor','g');
cell a <== x y 'string 1'  'string2'  value 1  ....
```

Significance of Cell Arrays (Cont.)

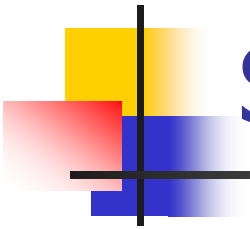
- **varargin**

  This argument returns a cell array with any number of actual arguments.

- **varargout**

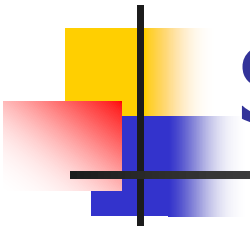
cf. nargin
nargout
nargchk
error
warning
inputname

```
>> help varargin
VARARGIN Variable length input argument list.
   Allows any number of arguments to a function.  The variable
   VARARGIN is a cell array containing the optional arguments to the
   function.  VARARGIN must be declared as the last input argument
   and collects all the inputs from that point onwards. In the
   declaration, VARARGIN must be lowercase (i.e., varargin).
For example, the function,
                function myplot(x,varargin)
                plot(x,varargin{:})
   collects all the inputs starting with the second input into the
   variable "varargin".  MYPLOT uses the comma-separated list syntax
   varargin{:} to pass the optional parameters to plot.  The call,
        myplot(sin(0:.1:1),'color',[.5 .7 .3],'linestyle',':')
   results in varargin being a 1-by-4 cell array containing the
   values 'color', [.5 .7 .3], 'linestyle', and ':'.
```



Significance of Cell Arrays (Cont.)

```
plot(x,y);
plot(x,y,'-ro');

function plotline(varargin)
```



Significance of Cell Arrays (Cont.)

```
function plotline(varargin)

msg=nargchk(2,inf,nargin);
error(msg);

j=0;
string='';
```

Significance of Cell Arrays (Cont.)

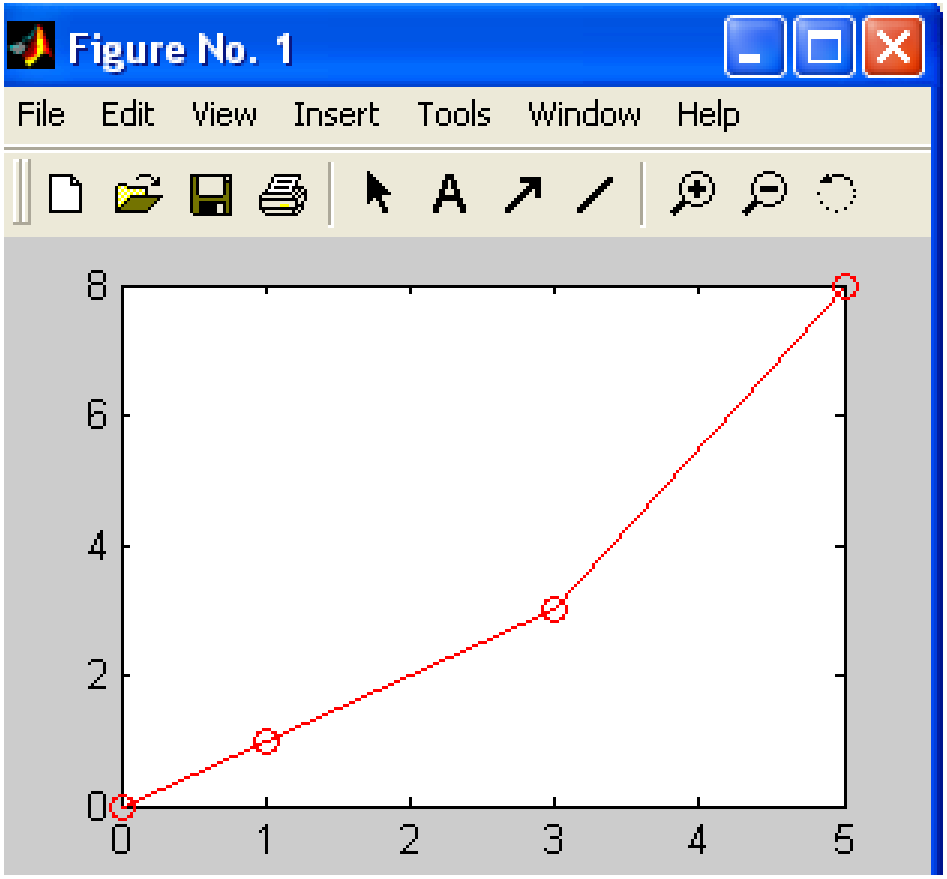
```
for i=1:nargin
    if ischar(varargin{i})
        string=varargin{i};
    else
        j=j+1;
        x(j)=varargin{i}(1);
        y(j)=varargin{i}(2);
    end
end
```

Significance of Cell Arrays (Cont.)

```
if isempty(string)
    plot(x,y);
else
    plot(x,y,string);
end
```

Significance of Cell Arrays (Cont.)

```
>> plotline([0 0], [1 1], [3 3], [5 8], '-ro');
```



About varargout

>> help varargout VARARGOUT Variable length output argument list.

Allows any number of output arguments from a function. The variable VARARGOUT is a cell array containing the optional output arguments from the function. VARARGOUT must be declared as the last output argument and must contain all the outputs after that point onwards. In the declaration, VARARGOUT must be lowercase (i.e., varargout).

VARARGOUT is not initialized when the function is invoked. You must create it before your function returns. Use NARGOUT to determine the number of outputs to produce.

For example, the function,

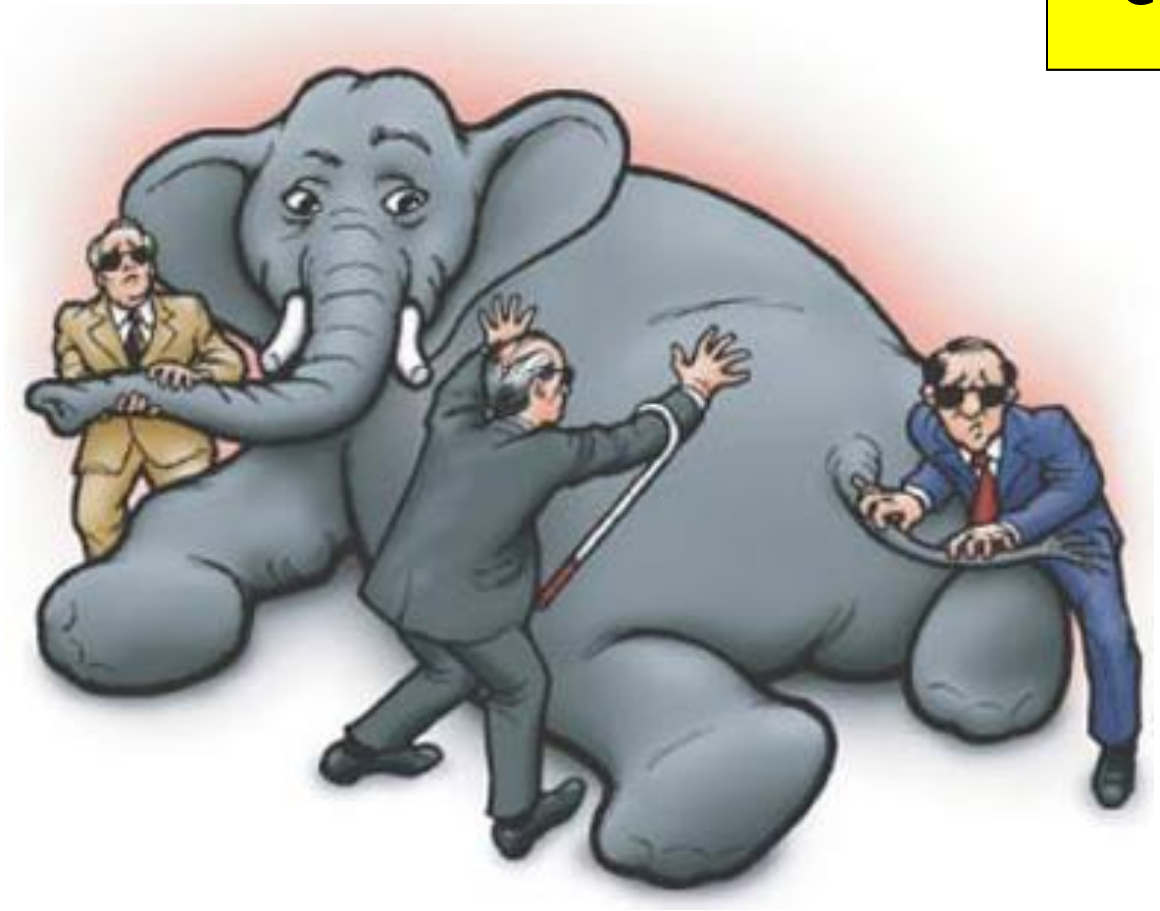
```
function [s,varargout] = mysize(x)
nout = max(nargout,1)-1;
s = size(x);
for i=1:nout, varargout(i) = {s(i)}; % varargout{i} = s(i), in some new version
end
```

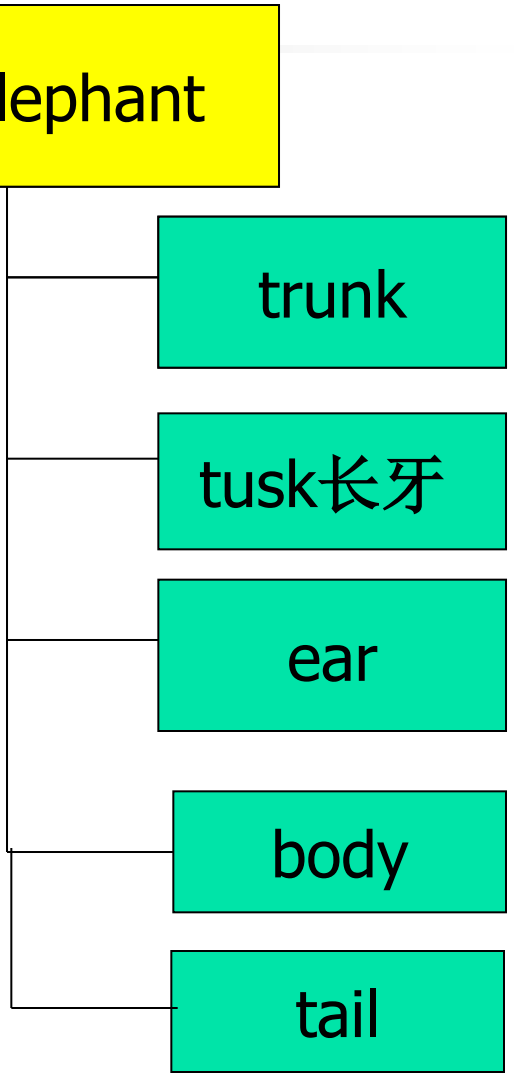
returns the size vector and optionally individual sizes. So,

```
[s,rows,cols] = mysize(rand(4,5));
```

returns s = [4 5], rows = 4, cols = 5.

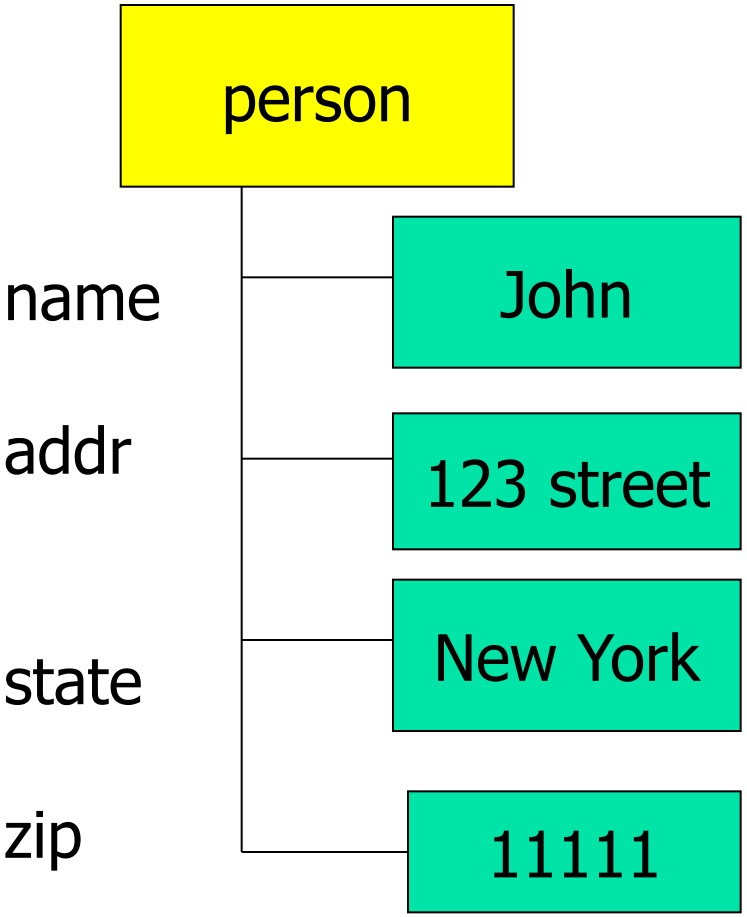
Structure Arrays





读老子感言：道不可口，可口者乃其似也。

Structure Arrays (Cont.)




```
person.name='John';
person.addr='123 street';
person.state='New York';
person.zip=11111;
```

A **structure** is a data type in which each individual property is given a name

field: an individual property of a structure

cf. the situation in C

Creating Structures

var=struct(fields)

a=struct('name','addr','state','zip');

a(1).name='John';
a(1).addr='123 street';
a(1).state='New York';
a(1).zip=111111;

a(2).name='...';
a(2).addr='...';
a(2).state='...';
a(2).zip=...;

a(3).name=...
...

Be careful: Different Versions!!

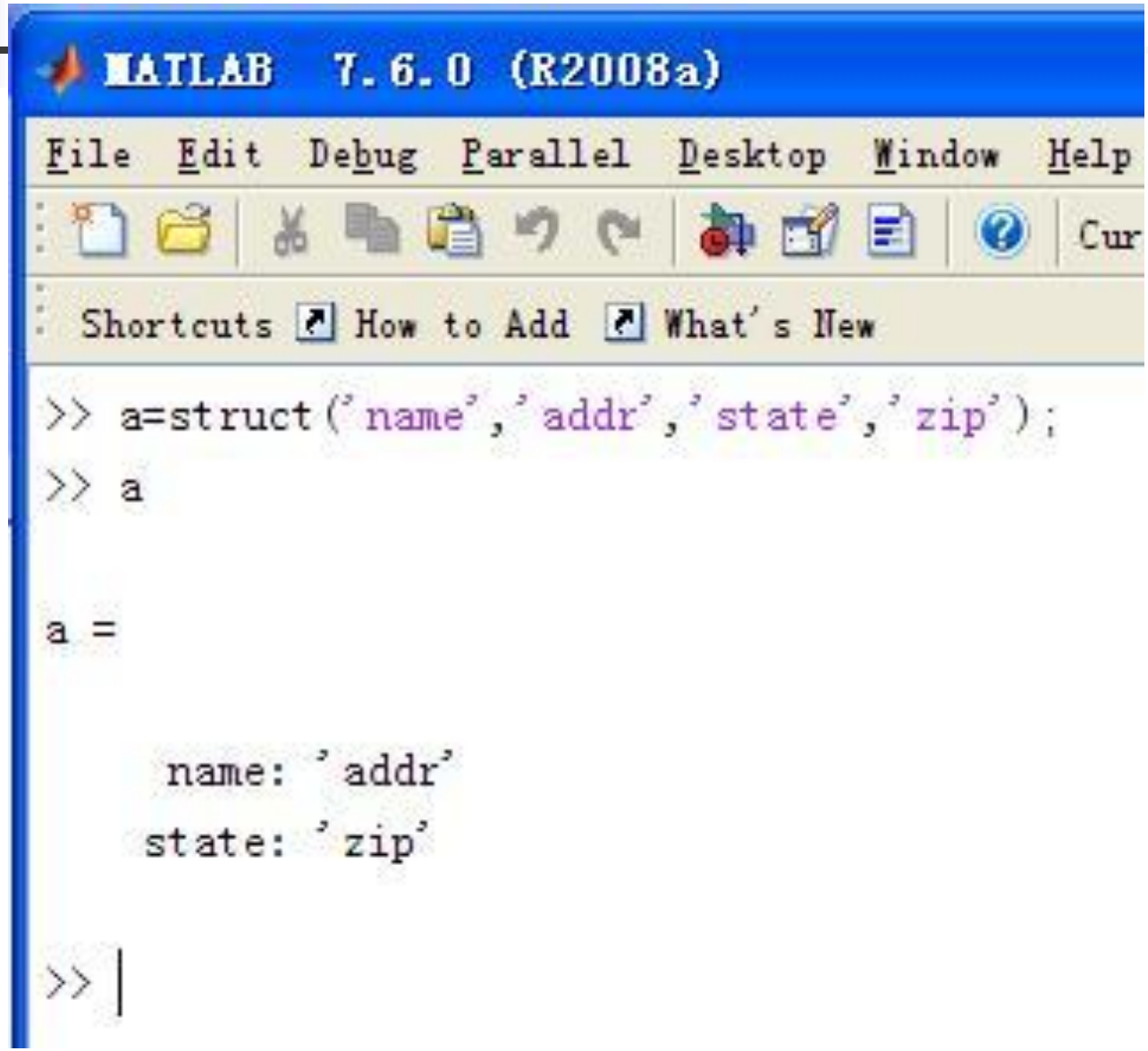
```
>> help struct      STRUCT Create or convert to structure array.
    S = STRUCT('field1',VALUES1,'field2',VALUES2,...) creates a
structure array with the specified fields and values.  The value
arrays VALUES1, VALUES2, etc. must be cell arrays of the same size,
scalar cells or single values. Corresponding elements of the value
arrays are placed into corresponding structure array elements.
    The size of the resulting structure is the same size as the
value cell arrays or 1-by-1 if none of the values is a cell.
    STRUCT(OBJ) converts the object OBJ into its equivalent
structure. The class information is lost.
    STRUCT([]) creates an empty structure.
    To create fields that contain cell arrays, place the cell
arrays within a VALUE cell array. For instance,
      s = struct('strings',{{'hello','yes'}},'lengths',[5 3])
    creates the 1-by-1 structure
       s = strings: {'hello'  'yes'}
           lengths: [5 3]
     Example
       s = struct('type',{'big','little'},'color','red','x',{3 4})
```

Be careful: Different Versions!!

```
>> a=struct('name','addr','state','zip')
a = name: 'addr'
    state: 'zip'
>> a(1)
ans = name: 'addr'
      state: 'zip'
>> a(2)
??? Index exceeds matrix dimensions.
>> a(2)=a(1);
>> a
a = 1x2 struct array with fields:
    name
    state
>> a(2)
ans = name: 'addr'
      state: 'zip'
```

证实→勿轻言否定→证伪

MATLAB 7.6 version：



Adding Fields to Structures

- By adding a new field to any element in a structure array, this field is automatically added to all of the elements in the array

```
a(1).grade=[2 3];
```

```
a(1).name='John';
a(1).addr='123 street';
a(1).state='New York';
a(1).zip=111111;
a(1).grade=[2 3];
```

Removing Fields from Structures

- new_struc=rmfield(old_array,'field')

  b=rmfield(a,'zip');

getfield and setfield functions

- var=getfield(array,{array_idx},'field',{field_idx})
- var=array(array_index).field(field_index)

```
a(1).name='John';
a(1).addr='123 street';
a(1).state='New York';
a(1).zip=111111;
a(1).grade=[2 3];
```

```
b=getfield(a,{1},'grade',{2});

b=a(1).grade(2);
```

getfield and setfield functions (Cont.)

- f=setfield(array,{array_idx},'field',{field_idx},val)
- array(array_index).field(field_index)=value

```
a(1).name='John';
a(1).addr='123 street';
a(1).state='New York';
a(1).zip=111111;
a(1).grade=[2 3];
```

f=setfield(a,{1},'grade',{1},4);

```
a(1).name='John';
a(1).addr='123 street';
a(1).state='New York';
a(1).zip=111111;
a(1).grade=[4 3];
```

→面向对象程序设计、类

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