Matlab 4: Matrix

Cheng-Hsin Hsu

National Tsing Hua University

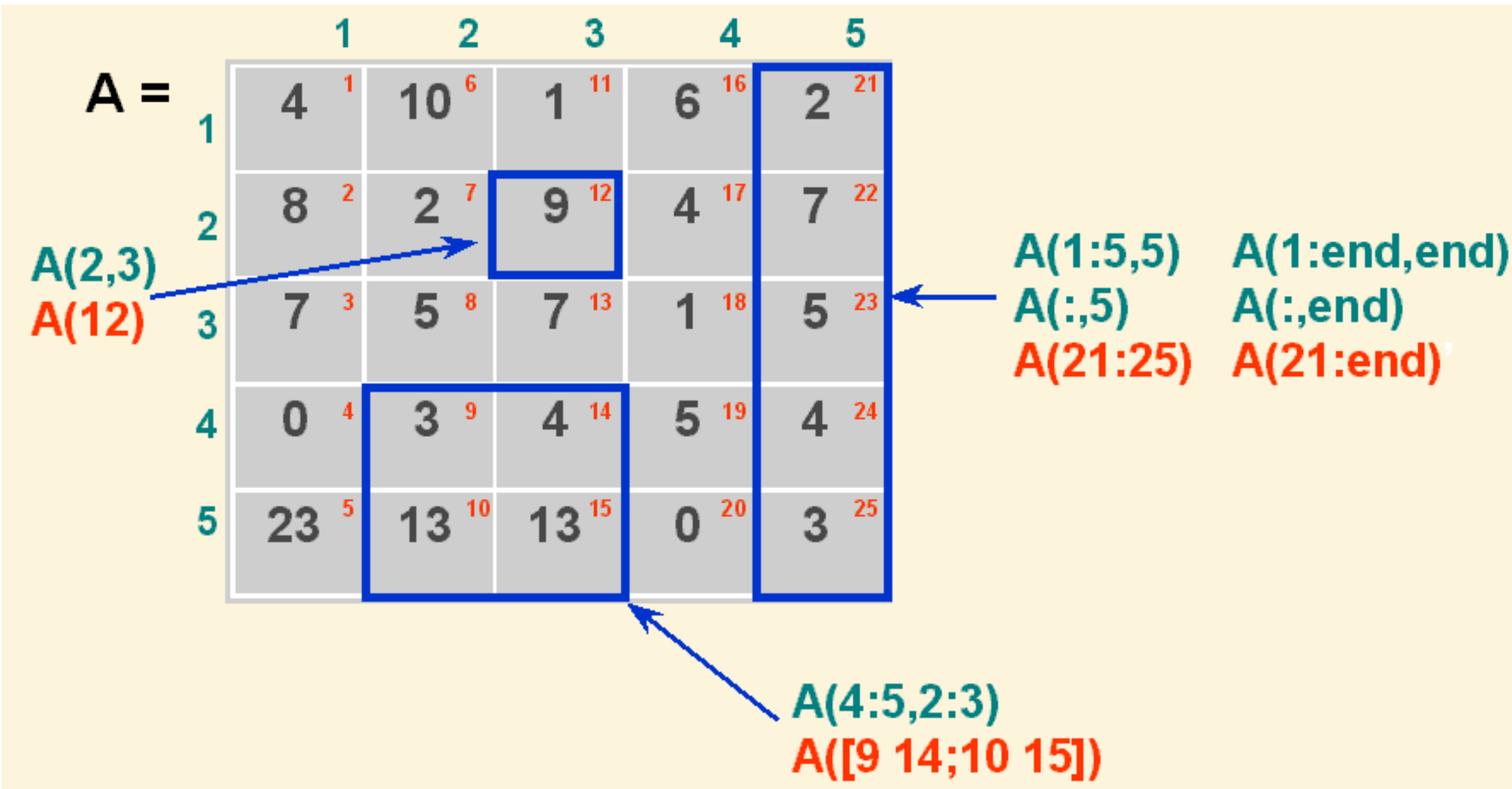
Department of Computer Science

Slides are based on the materials from Prof. Roger Jang

Matrix Indexes and Subscripts

- In Matlab, $A(i, j)$ denotes the i-th row, j-th column element of matrix A
  - i and j are the indexes or subscripts of the element
- Matlab adopts column-wise indexes
  - We can use 1- or 2-dimensional indexes to access the same elements
  - $A(i, j)$ and $A(i+(j-1)*m)$ are the same, where m is the number of rows

1- and 2-dimensional Indexes



Indexing Part of A Matrix

- A(4:5,2:3) refers to rows 4 and 5, columns 2 and 3
  - A([9 14; 10 15]) points to the same 2x2 sub-matrix
- : indicates the whole row or column
  - A(:,5) refers to the 5th column of A
- end indicates the last element of a dimension
  - A(:,end) is the last (right-most) column of A
- Use [] to delete a column or row
  - A(2, :) = [] ← remove the row 2 completely

More Matrix Manipulations

- Create a bigger matrix
  - A = magic(5)
  - B = [A 1./A]
  - C = [B; 1./B]
- Retrieve elements along the diagonal
  - D = diag(A)
- Generate a diagonal matrix
  - E = diag([1, 2, 3, 4])

More Matrix Manipulations (cont.)

- Multiply column/row vectors
  - V = [1; 2; 3; 4; 5]; B = magic(5);
  - B * B
  - V' * B
- Reshape flattens a matrix into a column vector and then put individual elements into an *m* x *n* matrix
  - A = magic(4)
  - B = reshape (A, 2, 8)

Matrix Generation Functions

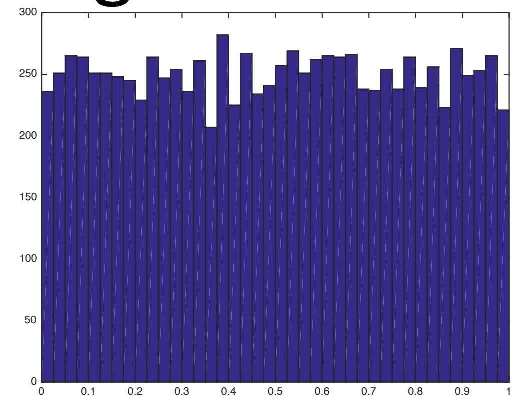
- zeros(m, n): all elements are 0
- ones(m, n): all elements are 1
- eye(n): identify matrix
- pascal(n): Pascal matrix
- vander(v): Vandermonde matrix, columns are powers of v
- rand(m, n): m x n random matrix, uniform distribution
- randn(m, n): m x n random matrix, zero mean, unit variance Gaussian distribution
- hilb(n): n x n Hilbert matrix ← ill-conditioned matrix
- magic(n): n x n matrix with equal sums among rows, columns, and diagonals

Hilbert Matrix

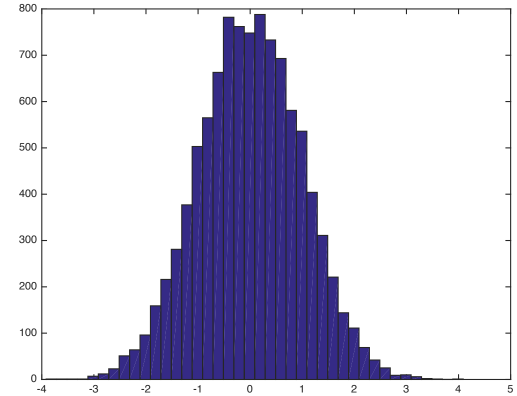
- Hilbert matrix is defined as $H_{i,j} = \frac{1}{i+j-1}$
  - Larger Hilbert matrices are closer to singular (not invertible, determinant is zero)
  - Observe: hilb(3), inv(hilb(3)), hilb(12), inv(hilb(12))
  - Given Ax = b, matrix A is **ill-conditioned** if a small changes on b's values leads to large changes on vector x ← challenging matrices to algorithms

Uniform and Gaussian Distributions

- rand: random numbers following uniform distribution
  - x1 = rand(10000, 1);
  - hist(x1, 40)



- randn: Gaussian distribution
  - x1 = randn(10000, 1);
  - hist(x1, 40)



Matrix Addition and Subtraction

- Matrix addition/subtraction are done between matrices with the same size
  - >> [1, 2, 3] + [4, 5, 6]
  - ans = 5 7 9
- Addition/subtraction between a matrix and a scalar → the scalar is applied to all elements
  - >> 10 + [4, 5, 6]
  - ans = 14 15 16

Matrix Multiplication/Division

- Scalar multiplication/division
  - [1, 2, 3] * 12
  - [1, 2, 3] / 2
- Multiplication of two matrices: the column number of the 1st matrix must be the same as the row number of the 2nd matrix
  - A = [1; 2];
  - B = [3, 4, 5];
  - C = A*B
- Division of two matrices: using inverse matrix or solving equation systems

Power and Element-by-Element Operations

- Matrix power: meaningful for squares
  - magic(5) ^ 2
- Adding a . before +, -, *, / indicates element-by-element operations
  - A = [12; 45];
  - B = [2; 3];
  - C = A.*B
  - D = A./B
  - E = A.^2

Conjugate Transpose

- Conjugate transpose
  - z = [1+i, 2; 3, 1+2i];
  - w = z'
- Transpose
  - w = z.'
- For real numbers, the conjugate transpose is the same as transpose

Vector's L_p-norm

- Vector a's L_p norm is defined as $\|a\|_p = \left(\sum_i |a_i|^p\right)^{1/p}, p \ge 1$
  - p=1 ➔ taxicab distance, or Manhattan distance
  - p=2 ➔ Euclidean Length (length of a)
  - P=inf ➔ maximum distance
- norm(x, p)
  - a = [3 4];
  - norm(a, 1)
  - norm(a, 2)
  - norm(a, inf)

Exercise

- Prove

$$\|a\|_\infty \le \|a\|_2 \le \|a\|_1$$

- On a 2D space, plot the trajectory of

$$\|a\|_1 = 1$$

$$\|a\|_2 = 1$$

$$\|a\|_\infty = 1$$

Matrix's L_p-norm

- Matrix A's L_p-norm is defined as $\|A\|_p = \max_x \frac{\|Ax\|_p}{\|x\|_p}$, where $x \in R^n$
- Command norm can be used for matrices as well
  - A = [1 2 3; 4 5 6; 7 8 9];
  - norm(A, 2)

Sorting a Vector

- Sort the elements in a vector
  - x = [3 5 8 1 4];
  - [sorted, index] = sort(x)
- Sorted is the new vector, index is the position of the sorted elements in the original vector
  - Hence, x(index) is equivalent to the sorted vector
- Exercise: how to get vector x using sorted and index
  - [~, rindex] = sort(index)
  - sorted(rindex)

Locating the Maximum in Each Column

- >> x = magic(5);
  [colMax, colMaxIndex] = max(x)
  colMax = 23   24   25   21   22
  colMaxIndex = 2   1   5   4   3
- >> [maxValue, maxIndex] = max(colMax);
- >> fprintf('Max value = x(%d, %d) = %d\n', colMaxIndex(maxIndex), maxIndex, maxValue);
  Max value = x(5, 3) = 25
- Just to find the max value: max(max(x)) or max(x(:))

Array Element Data Types

- Double by default, but recent versions support additional data types

Data Type	Description
uint8	Unsigned 8-bit integer, range: [0,255]
uint16	Unsigned 16-bit integer, range: [0,65535]
uint32	Unsigned 32-bit integer, range: [0,2^32-1]
int8	Signed 8-bit integer, range: [-128,127]
int16	Signed 16-bit integer, range: [-32768,32767]
int32	Signed 32-bit integer, range: [-2^31,2^31-1]
single	32-bit single precision floating number
double	64-bit double precision floating number
char	16-bit character

Memory Usage of Different Data Types

- x_double = magic(10);
- x_single = single(x_double); ← casting
- x32 = uint32(x_double);
- x16 = uint16(x_double);
- x8 = uint8(x_double);
- whos ← show the memory consumption
- Sample observation: uint8 array consumes 1/8 of memory compared to double

Tips on Choosing Data Types

- Out of range integers
  - uint8(999)
  - uint8(-999)
- Variables in different data types can be compared (after implicit conversions)
  - uint8(20)== 20
- Explicit conversions may be necessary
  - z=uint8(2.54) * 2.5
  - z * 3.33
  - double(z) * 3.33

Strings and Characters

- A string consists of a sequence of characters
- In Matlab, a string can be seen as a row vector of characters, encoded in ASCII
- Matlab uses single quotation marks for string boundary; two strings can be concatenated
  - str1 = 'Taipei zoo displays animals from Taiwan'; str2 = ', Australia, Africa, and desert.'; [str1 str2]
- If you get lost: help strfun

Escape Character and Length

- For a string with single quote, add an extra single quote
  - 'I''ve got homework to do. '
- Use length to calculate the number of characters in a string
  - length('I''ve got homework to do.')

Conversions between ASCII and String

- double: convert human-readable string into ASCII vector
- char: convert ASCII vector into human-readable string
  - s = 'I''ve got homework to do.';
  - a_asc = double(s)
  - s2 = char (a_asc)

Storage Usage of Characters

- `>> whos s2`

```
  Name       Size                Bytes  Class
Attributes
  s2         1x24                   48  char
```

- Each character consumes 2 bytes, even for ASCII
  - Internally Matlab uses unicode, support CJK characters
- Can perform operations on string as if it's a double vector
  - double(s2)
  - s2 + 1

Execute Dynamically-Composed Commands

- Use eval to execute a string as if typing the command in the command window

```
for i = 2:5
    command = (['x', int2str(i) , '=
magic(' , int2str(i) , ') '])
    eval(command);
end
```

Determine if a Variable is Char

- Two possibilities: class or ischar

```
>> class(s2)
ans =char
>> ischar(s2)
ans = 1
>> ischar(s2+1)
ans = 0
```

Storing Multiple Strings

- Save them as a two dimensional array
- But all strings should have the same length
  - departments = ['ee   '; 'cs   '; 'econ']
  - whos departments
- Or use char command
  - departments2 = char('ee', 'cs', 'econ')
  - whos departments2

Storing Multiple Strings (cont.)

- How to remove the extra trailing spaces?
  - deblank
- We will discuss a better solution (cell arrays) soon

```
>> departments(1,:)
ans =ee
>> length(departments(1,:))
ans = 4
>> length(deblank(departments(1,:)))
ans = 2
```

String Comparisons

- Matlab supports strcmp, which returns 1 is two string are equivalent
  - Note that this is the opposite of C’s strcmp

- `>> strcmp('elephant', 'bear’)`
- `ans = 0`
- `>> strcmp('monkey', 'monkey’)`
- `ans = 1`

Other String Related Commands

- strcmpi: ignore upper/lowercase
- strncmp: first n chars
- findstr: find a substring and return the index
- strrep: replace substrings

```
>> strrep('This is a monkey', 'monkey', 'bear’)
ans = This is a bear
```

Parsing Strings

- strtok: based on a delimiter (space by default), get the first token
- strvcat: merge multiple string into a 2-D array with automatic padding

```
input_string = 'ee cs econ stat me';
remainder = input_string;
parsed = '';
while (any(remainder))
    [chopped, remainder] = strtok(remainder);
    parsed = strvcat(parsed, chopped);
end
parsed
```

Conversion among Data Types

- int2str: convert integer into string
- num2str: convert real number into string
- dec2hex: convert decimal into hexdecimal
- hex2num, hex2dec, bin2dec, and others

Conversion between Strings and Arrays

- mat2str: convert matrix into string
- eval: convert string back to matrix

```
A = [1 2 1; 3 5 6 ];
B = mat2str(A)
A2 = eval(B)
isequal(A, A2)
```

Sprintf and Sscanf

- >>str='pi'; newString = sprintf('%s is %d', str, pi)

  newString = pi is 3.141593e+00
- >> str = '2 4.7 5.2';

  mat = sscanf(str, '%f')

  mat =

  2.0000

  4.7000

  5.2000
- %s indicates string, %d represents number, ...

Cell Array and How to Initialize It

- Store variables with different data types in the same array
- Approach #1: use { … } on the right

```
A(1,1) = {'This is the first cell.'};
A(1,2) = {[5+j*6 , 4+j*5]};
A(2,1) = {[1 2 3; 4 5 6; 7 8 9]};
A(2,2) = {{'Tim'; 'Chris'}}
```

Resulting Cell Array

- The resulting 2x2 cell array

A(1,1) : 'This is the first cell' String	A(1,2) : [5+j*6 4+j*5] 1x2 complex number array
A(2,1) : 1 2 3 4 5 6 7 8 9 3x3 integer array	A(2,2) : {'Tim', 'Chris'} 2x2 cell array

Initializing Cell Array

- In contrast to arrays, whereas we use (…) for indexing; for cell arrays, we use {…}
- Approach #2: use { … } on the left (indexing)
  - `A{1,1} = 'this is the first cell.';`
  - `A{1,2} = [5+j*6, 4+j*5];`
  - `A{2,1} = [1 2 3; 4 5 6; 7 8 9];`
  - `A{2,2} = {'Tim'; 'Chris'}`

Initializing Cell Array (cont.)

- Approach #3: putting all elements in a single {...} (on the right)

```
>> B={'James Bond', [1 2;3 4;5 6]; pi, magic(5)}
>> C={rand(3), ones(2); zeros(5), randperm(4)}
```

Merging Two Cell Arrays

- Similar to merging two arrays

```
>> M = [B C]
M =
    'James Bond'     [3x2 double]     [3x3 double]     [2x2 double]
    [    3.1416]     [5x5 double]     [5x5 double]     [1x4 double]
```

Showing the Content of Cell Array

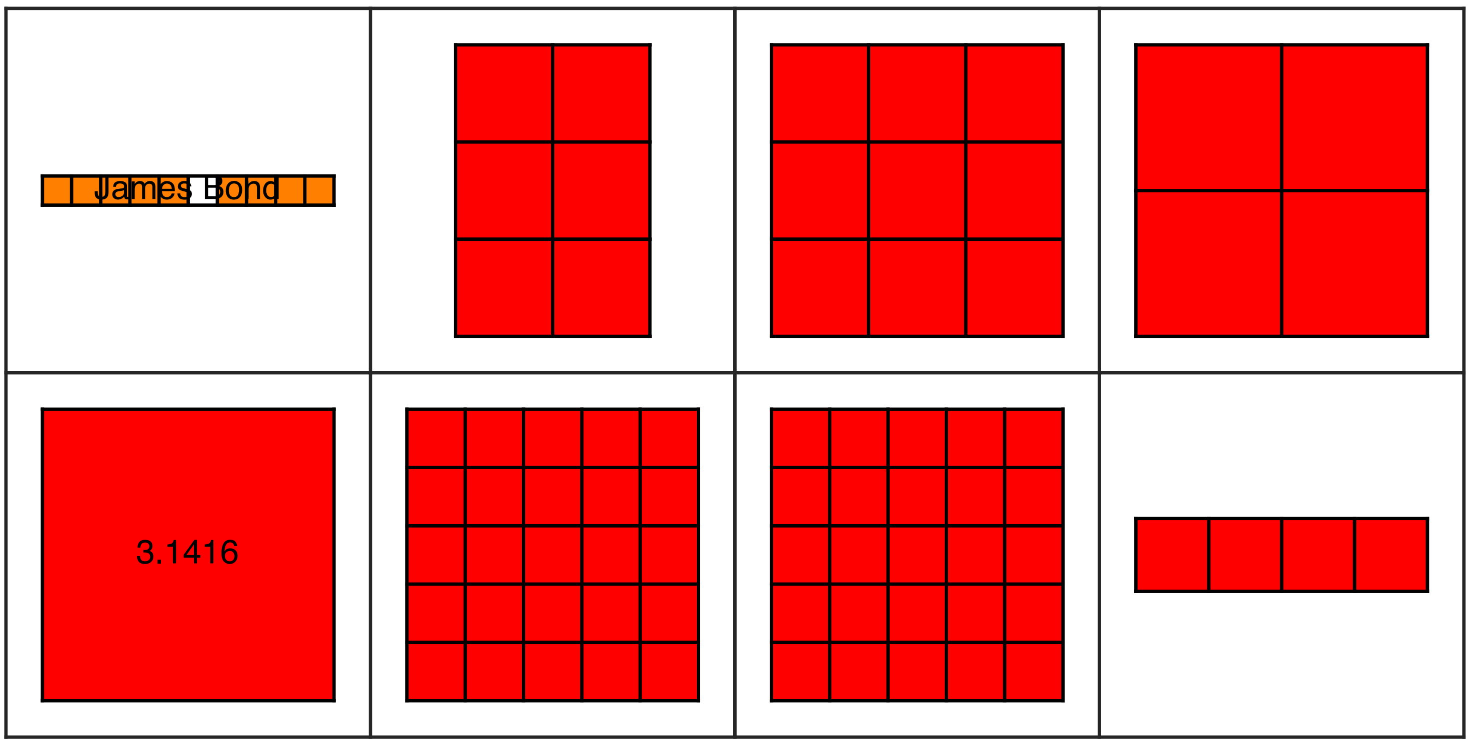
- Directly printing a cell array only shows dimensions and data types
- Print an element in a cell array instead

```
>> M
M = 'James Bond'     [3x2 double]     [3x3 double]     [2x2 double]
    [    3.1416]     [5x5 double]     [5x5 double]     [1x4 double]

>> M{1,2}
ans =
     1     2
     3     4
     5     6
```

- Challenge: print all elements in a command?

Visualization of A Cell Array

- Cellplot(M): visualize cell array M



Display A Cell Array

- celldisp(M): print cell array M

```
>> celldisp(M)
M{1,1} = James Bond

M{2,1} = 3.1416

M{1,2} =
     1     2
     3     4
     5     6
......
```

Access Elements in Cell Arrays

- Get an element from a cell array

```
>> M{1,2}

ans =
       1      2
       3      4
       5      6
```

Access Elements in Cell Arrays (cont.)

- Get an element from an array within a cell array

```
>> M{1,2}(2,1)

ans = 3
```

Access/Remove Multiple Elements

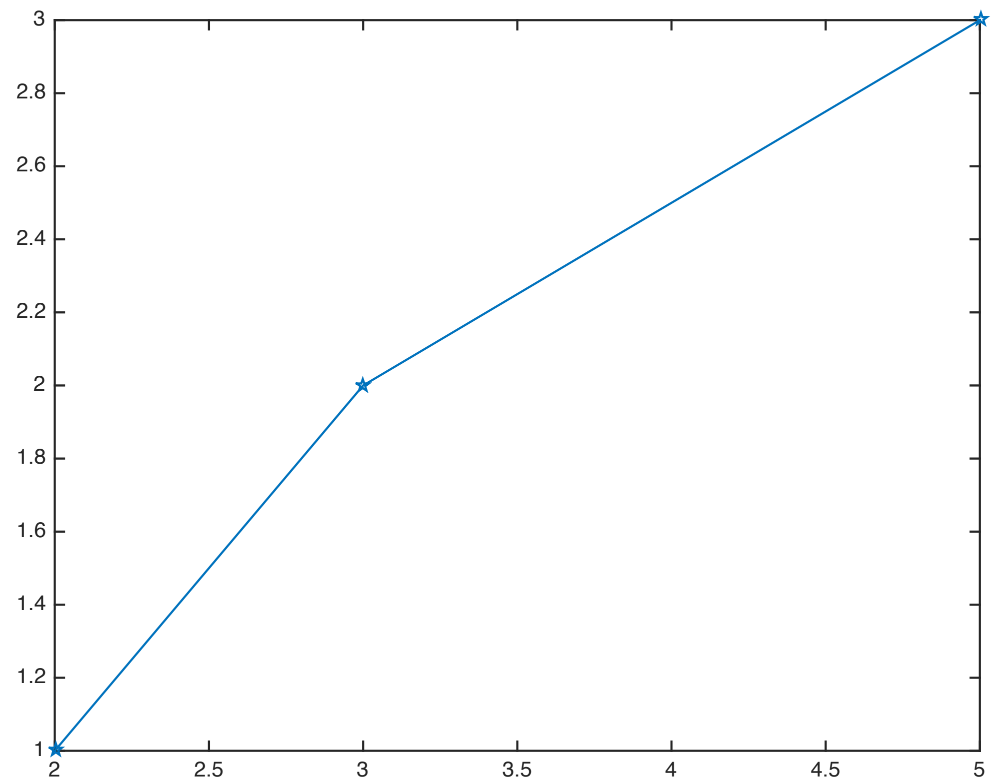
```
>> size(M)
ans = 2     4

>> M(1,:)
ans = 'James Bond'    [3x2 double]    [3x3 double]    [2x2 double]

>> M(1,:) = []
M = [3.1416]    [5x5 double]    [5x5 double]    [1x4 double]

>> size(M)
ans = 1     4
```

Cell Arrays for Comma-Separated Arguments

- F = {[2 3 5], [1 2 3], 'Timmy', 'Annie'};
- >> F{1:2}

  ans = 2 3 5

  ans = 1 2 3
- plot(F{1:2}, '-p')



Comma-Separated Returns

```
>> [Z{1:2}] = max(rand(5))

Z = [1x5 double]    [1x5 double]

>> Z{1,1}
ans = 0.9294    0.8176    0.8116    0.9390    0.8443

>> Z{1,2}
ans = 1     4     3     1     5
```

Resize Cell Arrays

- Use (:)

```
>> M
M =
    'James Bond'    [3x2 double]    [3x3 double]    [2x2 double]
    [    3.1416]    [5x5 double]    [5x5 double]    [1x4 double]

>> M(:)
ans =
    'James Bond'
    [    3.1416]
    [3x2 double]
    [5x5 double]
    [3x3 double]
    [5x5 double]
    [2x2 double]
    [1x4 double]
```

Resize Cell Arrays (cont.)

- Use reshape

```
>> B
B = 'James Bond'     [3x2 double]
    [    3.1416]     [5x5 double]
>> N = reshape(B,1,4)

N = 'James Bond'     [3.1416]     [3x2 double]     [5x5 
double]
```

Pre-allocate and Test Cell Array

- A = cell(4,3); ← create a 4x3 empty cell array
- iscell(A) ← test if A is a cell array

```
>> A=cell(4,3)
A =
    []     []     []
    []     []     []
    []     []     []
    []     []     []

>> iscell(A)
ans =  1
```

Convert Number Array to Cell Array

- C = num2cell(A, dim) ← convert a number array A into a cell array C, where dim is the dimension
- Default: each number becomes an 1x1 matrix in the cell array

```
>> A=[1, 2, 3; 4, 5, 6];
>> C=num2cell(A)
C = [1]     [2]     [3]
    [4]     [5]     [6]
```

Convert Number Array to Cell Array (cont.)

```
>> C=num2cell(A,1)
C = [2x1 double][2x1 double][2x1 double]

>> C=num2cell(A,2)
C = [1x3 double]
    [1x3 double]
```

Convert Number Array to Cell Array (cont.)

- Exercise: mat2cell(.) supports more sophisticated conversions

```
>> X = [1 2 3 4; 5 6 7 8; 9 10 11 12]
C = mat2cell(X,[1 2],[1 3])

X =  1      2      3      4
     5      6      7      8
     9     10     11     12

C = [          1]    [1x3 double]
    [2x1 double]    [2x3 double]
```

Structure Array

- Each structure contains several elements
- Each element has several fields, such as: name, student id, and scores
- These fields may be in different data types

Example of Structure Array

- We create and dump the first element in a structure array

```
>> student.name = 'Bear Hsu';
student.id = '12345678';
student.scores = [10, 20, 30];
>> student
student =
      name: 'Bear Hsu'
        id: '12345678'
    scores: [10 20 30]
```

Example of Structure Array (cont.)

- Add one more element to the structure array

```
>> student(2).name = 'CP Tan';
student(2).id = '12345679';
student(2).scores = [25, 36, 92];
>> student
student =
1x2 struct array with fields:
    name
    id
    scores
```

Print the Content of Structure Array

```
>> student(2)
ans =
      name: 'CP Tan'
        id: '12345679'
    scores: [25 36 92]

>> student(2).scores
ans =
    25    36    92
```

Create Structure Array using Struct

- structureArray = struct(field1, value1, field2, value2,….)
- If value1, value2, …., are cell arrays, matlab will create multiple elements in structure array

```
student = struct('name', {'Bear', 'CP'},
'scores', {[50 60], [60 70]});
student(1)
student(2)
```

Create Structure Array using Struct (cont.)

- If value1, value2, …., are scalar, matlab will expand it and assign to all elements ← like default values

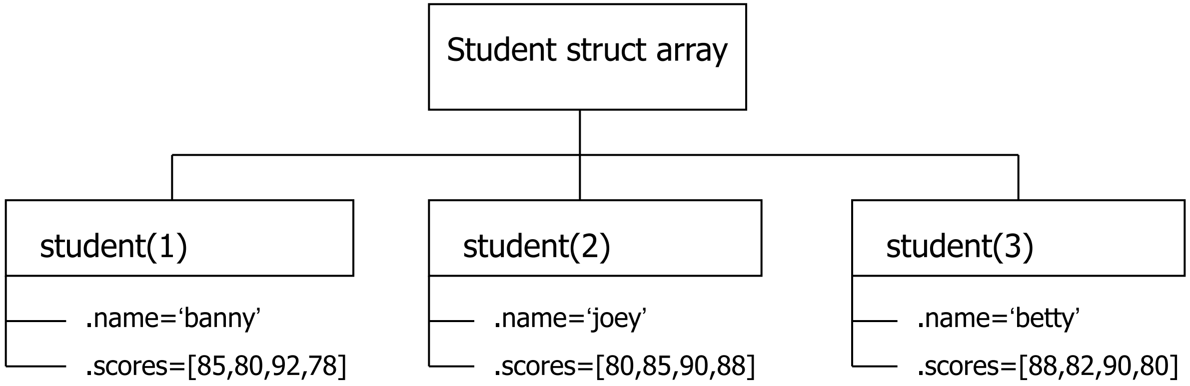
```
student = struct('name', {'Bear', 'CP'},
'scores', [0, 0]);
student(1)
student(2)
```

Nested Struct Arrays

```
student = struct('name', {'Bear', 'CP'},
'scores', {[50 60], [60  70]});
student(2).course(1).title = 'Web
Programming';
student(2).course(1).credits = 2;
student(2).course(2).title = 'Numerical
Method';
student(2).course(2).credits = 3;
student(2).course
```

Another Struct Example

```
student(1) = struct('name', 'Banny', 'scores', [85,80,92,78]);
student(2) = struct('name', 'Joey', 'scores', [80,85,90,88]);
student(3) = struct('name', 'Betty', 'scores', [88,82,90,80]);
```



Use Cell Array to Access Elements

```
>> values = struct2cell(student)
values(:,:,1) =
    'Banny'
    [1x4 double]
values(:,:,2) =
    'Joey'
    [1x4 double]
values(:,:,3) =
    'Betty'
    [1x4 double]
>> size(values)
ans = 2     1     3
```

- If the input struct array is *m* x *n* with p fields, the cell array has dimension *p* x *m* x *n*

Editing Struct Array

```
>> student(2)
ans =
     name: 'Joey'
   scores: [80 85 90 88]

>> student(2).name = 'Bear’;

>> student(2)
ans =
     name: 'Bear'
   scores: [80 85 90 88]
```

Extract Scores from Struct Array

- A = cat(dim, structureField) ← turn numbers into numerical array
  - dim=2 for row arrangement
  - dim=1 for column arrangement

```
>> cat (1, student.scores)
ans =
    85    80    92    78
    80    85    90    88
    88    82    90    80
>> cat (2, student.scores)
ans =
    85    80    92    78    80    85    90    88    88
82    90    80
```

Calculating Average

- Average score per exam
- Average score per student

```
>> mean(cat (1, student.scores))
ans =
   84.3333   82.3333   90.6667   82.0000

>> mean(cat (1, student.scores)')
ans =
   83.7500   85.7500   85.0000
```

Flatten a Field

```
>> allScores = [student.scores]
allScores =
    85    80    92    78    80    85    90    88
88    82    90    80

>> allNames = {student.name}
allNames =
    'Banny'    'Bear'    'Betty'

>> allName = [student.name]
allName =
BannyBearBetty
```

Iterating Through Struct Array

```
>> for i = 1:length(student)
    fprintf ('student %g: %s\n', i,
student(i).name);
end
student 1: Banny
student 2: Bear
student 3: Betty
```

Get and Change Values of a Field

- fieldValues = getfield (structureArray, {arrayIndex}, field, {fieldIndex})
- newStructure = setfield (structureArray, {arrayIndex}, field, {fieldIndex})
- The following two statements are the same

```
>> score3 = getfield(student, {2}, 'scores', {1})
>> score3 = student(2).scores(1);
```

- The following two are the same

```
>> student = setfield(student, {2}, 'scores', {1}, 75);
>> student(2).scores(1)=75;
```

Get All Field Names

- fieldnames(.) returns a cell array of strings

```
>> student = struct('name', 'Roland', 'scores', [80, 90]);
>> allFields = fieldnames(student)
allFields =
    'name'
    'scores'
```

Add Fields

- Matlab automatically fills in empty fields

```
>> student(2).age = 20;
>> student(1)
ans =
      name: 'Roland'
    scores: [80 90]
       age: []

>> student(2)
ans =
      name: []
    scores: []
       age: 20
```

Remove Fields

- New_struct = rmfield(struct, field)

```
>> student2 = rmfield(student, 'scores');
>> fieldnames(student)
ans =
    'name'
    'scores'
    'age’
>> fieldnames(student2)
ans =
    'name'
    'age'
```

Test Variables

- isstruct(.) tests if a variable is a struct
- isfield(.) tests if a struct array contains a field

```
s = struct('name', {'Tim', 'Ann'}, 'scores',
{[1 3 5 ],[2 4 6]});
disp(isstruct(s));
fprintf('isfield(s, ''height'') = %d\n',
isfield(s, 'height'));
```

Convert Cell Array into Struct

- s = cell2struct(values, fields, dim) ← dim indicates which dimension should the fieldname be mapped to

```
fields = {'name', 'age'};
values = {'Tim', 9; 'Annie', 6};
s = cell2struct(values, fields, 2);
s(1)
s(2)
```

Convert Cell Array into Struct (cont.)

```
>> s = cell2struct(values, fields, 1);
s(1)
s(2)

ans =
    name: 'Tim'
     age: 'Annie'



ans =
    name: 9
     age: 6
```

Example: dir(.)

```
>> dirinfo = dir(matlabroot)
dirinfo =
28x1 struct array with fields:
    name
    date
    bytes
    isdir
    datenum

>> dirinfo(1).name
ans =
.

>> dirinfo(1).isdir
ans =
     1
```

Matlab #4 Homework (M4)

Note that due to the larger workload of M4 assignment. I double the points from 3% to 6%.

1. (2%) For a given vector x, we can sort its elements using "sort" command: [x2, index]=sort(x). The sorted vector x2 is actually equal to x(index).

   Write a function "sortInv" that returns x from given x2 and index: x=sortInv(x2, index).

   To test your function, try the following script and the value of z should be 1:

```
x=round(100*rand(1,100));
[x2, index]=sort(x);
x3=sortInv(x2, index);
z=isequal(x, x3)
```

Matlab #4 Homework (M4) cont.

2. (2%) Recall the minion picture we previously worked on, a true-color image (such as a jpg file) of size mxn can be represented as a 3-dimensional array of size mxnx3, where each layer (or page) is the pixel intensity of R (red), G (green), and B (blue), respectively. For certain application of image processing, you need to reshape A into a 2-dimensional matrix B of size 3x(mn), where each column is the RGB intensity vector of a pixel. (For instance, the first column of B is the RGB intensity of pixel (1,1), the second column is the RGB intensity of pixel (2,1), and so on.) Please write a function myReshape.m that can achieve this goal, with the following function prototype: B=myReshape(A). The TA will prepare a few testing images, your function needs to pass all of them to get the point.

Matlab #4 Homework (M4) cont.

3. (2%) Use struct array to maintain data about course grades of Scientific Computing. Please store the following data: student name, student id, assignments grades, midterm grades and final grades. Compute final scores of individual students, dump all data and final scores in a table, and plot a empirical cumulative distribution function (CDF) plot.