MATH2221
Mathematics Laboratory II

Lecture 9:
Data Analysis Using MATLAB

Gary Choi

March 18, 2025

Recall: Advanced linear algebra functions in MATLAB

- Vector/matrix norm: norm(v,p), norm(A,p)
- Dot product dot(u,v) (= sum(u.*v)), cross product cross(u,v)
- Reduced row echelon form: rref(A)
- Rank of a matrix: rank(A), rank(A,tol)
- Orthonormal basis for null space: null(A)
- Condition number: cond(A), cond(A,p)
- Eigenvalues/eigenvectors:
  - e = eig(A), [V,D] = eig(A)
  - eigs(A,k)

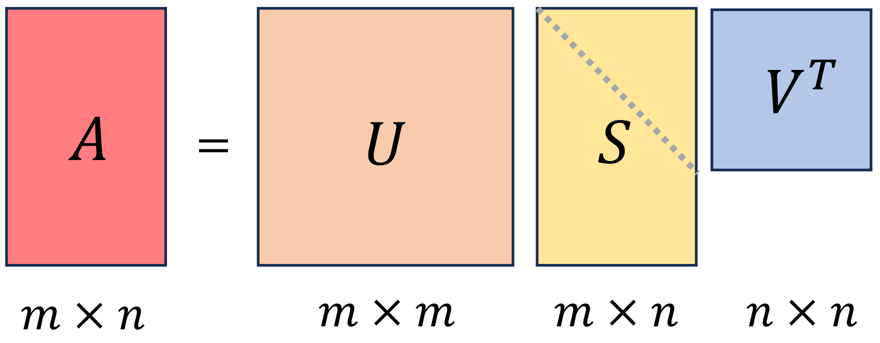
Recall: Matrix factorization in MATLAB

- LU factorization:
  - [L,U] =lu(A) ($A = LU$, L: lower triangular, U: upper triangular)
  - [L,U,P] =lu(A) ($PA = LU$, P: permutation matrix)

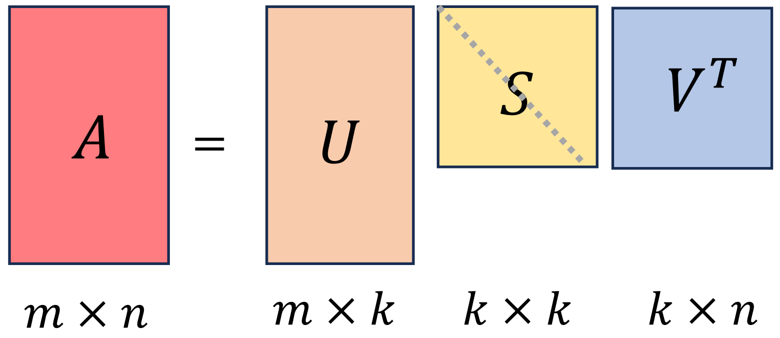
- QR factorization:
  - [Q,R] = qr(A) ($A = QR$, Q: orthogonal matrix, R: upper triangular)

- Singular value decomposition (SVD): $A = USV^T$
  - [U,S,V] = svd(A) (Full SVD)
  - [U,S,V] = svd(A,"econ") (Reduced SVD)

Full SVD



Reduced SVD



Data analysis: polynomial fitting

- **Fitting a degree-n polynomial** $y = p_n x^n + p_{n-1}x^{n-1} + \cdots + p_0$ to the given data points (x_i, y_i) (represented as vectors x, y)
  - p = polyfit(x,y,n)

- Example:

```
>> x = [1, 2.3, 3.4, 2.6, 6, 5.5, 4];
>> y = [2, 3, 5.2, 3, 12, 10.5, 6];
>> p = polyfit(x,y,1); % Linear fit
>> p
p =
   2.1263   -1.5759
```

Best-fit linear polynomial:

$$y = 2.1263x - 1.5759$$

- Example:

```
>> x = [1, 2.3, 3.4, 2.6, 6, 5.5, 4];
>> y = [2, 3, 5.2, 3, 12, 10.5, 6];
>> p = polyfit(x,y,3); % Cubic fit
>> p
p =
  -0.0371    0.7031   -1.3247    2.6517
```

Best-fit cubic polynomial:

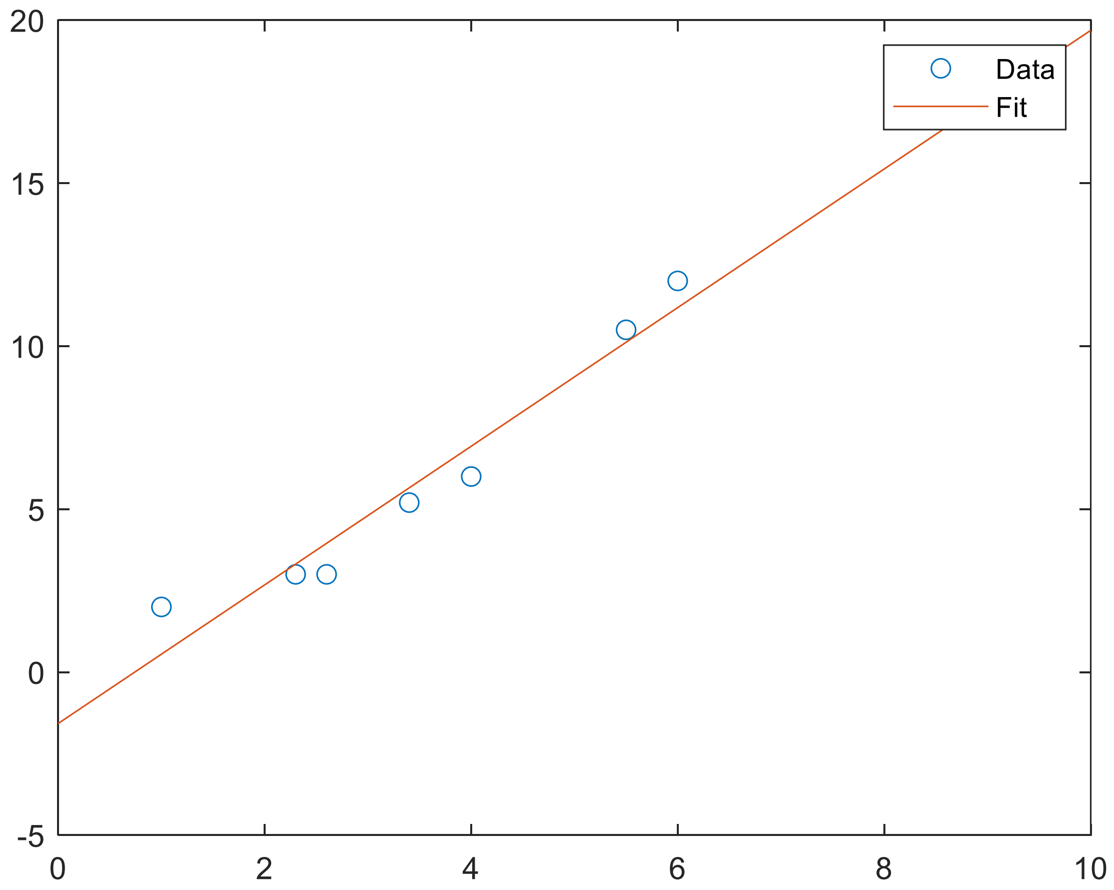
$$y = -0.0371x^3 + 0.7031x^2 - 1.3247x + 2.6517$$

Data analysis: polynomial fitting

- Evaluate the value at a specific x_i (scalar or vector) based on the given polynomial coefficient vector $\boldsymbol{p}$:
  - yi = polyval(p, xi)

- Example:

```
x = [1, 2.3, 3.4, 2.6, 6, 5.5, 4];
y = [2, 3, 5.2, 3, 12, 10.5, 6];
p = polyfit(x,y,1); % Linear fit
x1 = linspace(0,10,100);
y1 = polyval(p,x1);
figure;
plot(x,y,'o');
hold on
plot(x1,y1);
legend('Data', 'Fit')
```

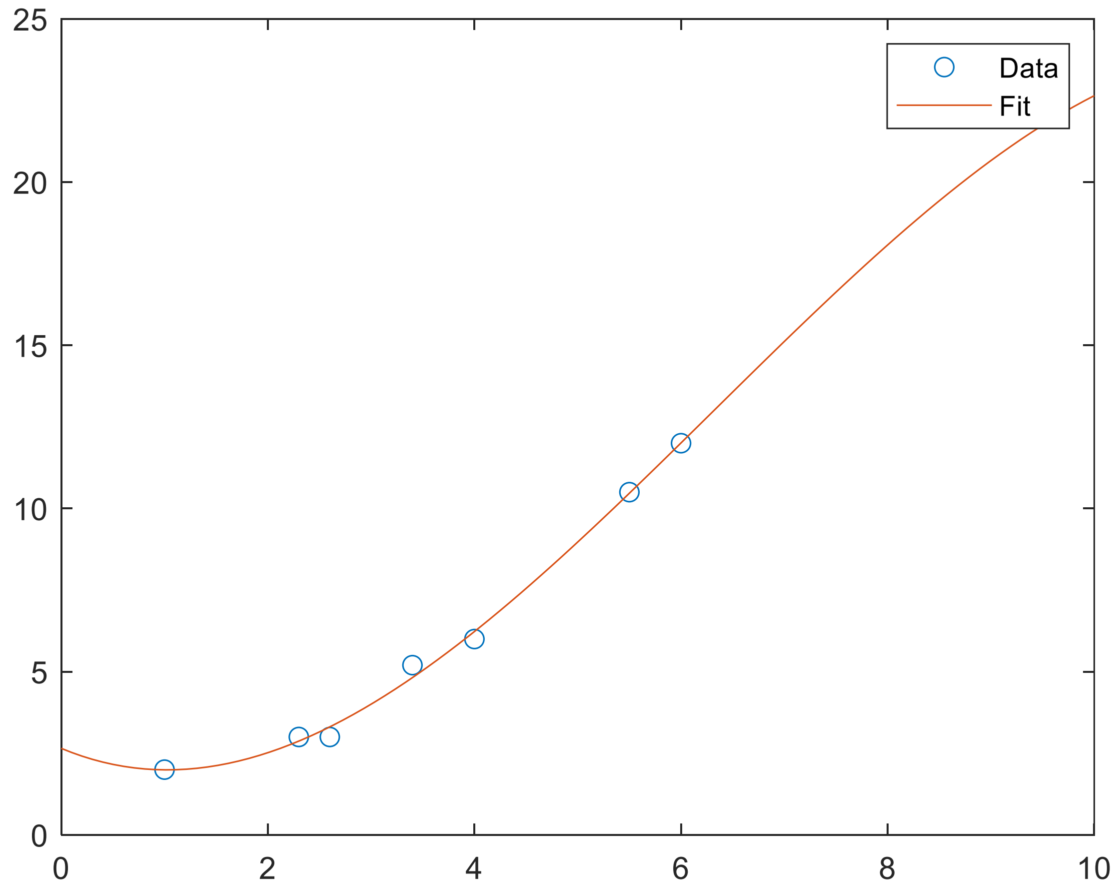


Data analysis: polynomial fitting

- Evaluate the value at a specific x_i (scalar or vector) based on the given polynomial coefficient vector $\boldsymbol{p}$:
  - yi = polyval(p, xi)

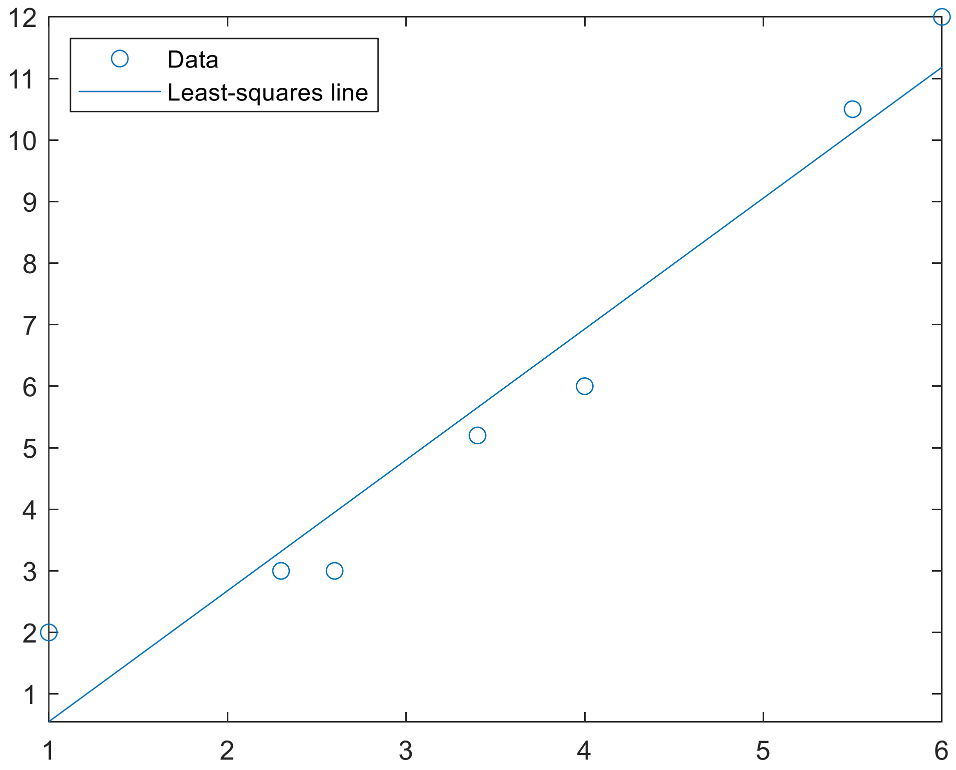
- Example:

```
x = [1, 2.3, 3.4, 2.6, 6, 5.5, 4];
y = [2, 3, 5.2, 3, 12, 10.5, 6];
p = polyfit(x,y,3); % Cubic fit
x1 = linspace(0,10,100);
y1 = polyval(p,x1);
figure;
plot(x,y,'o');
hold on
plot(x1,y1);
legend('Data', 'Fit')
```



Data analysis: polynomial fitting

- Another useful function: lsline
  - Superimposes a least-squares line on the current plot with scatter points
  - Much simpler command (if you only care about the visualization)



- Example:

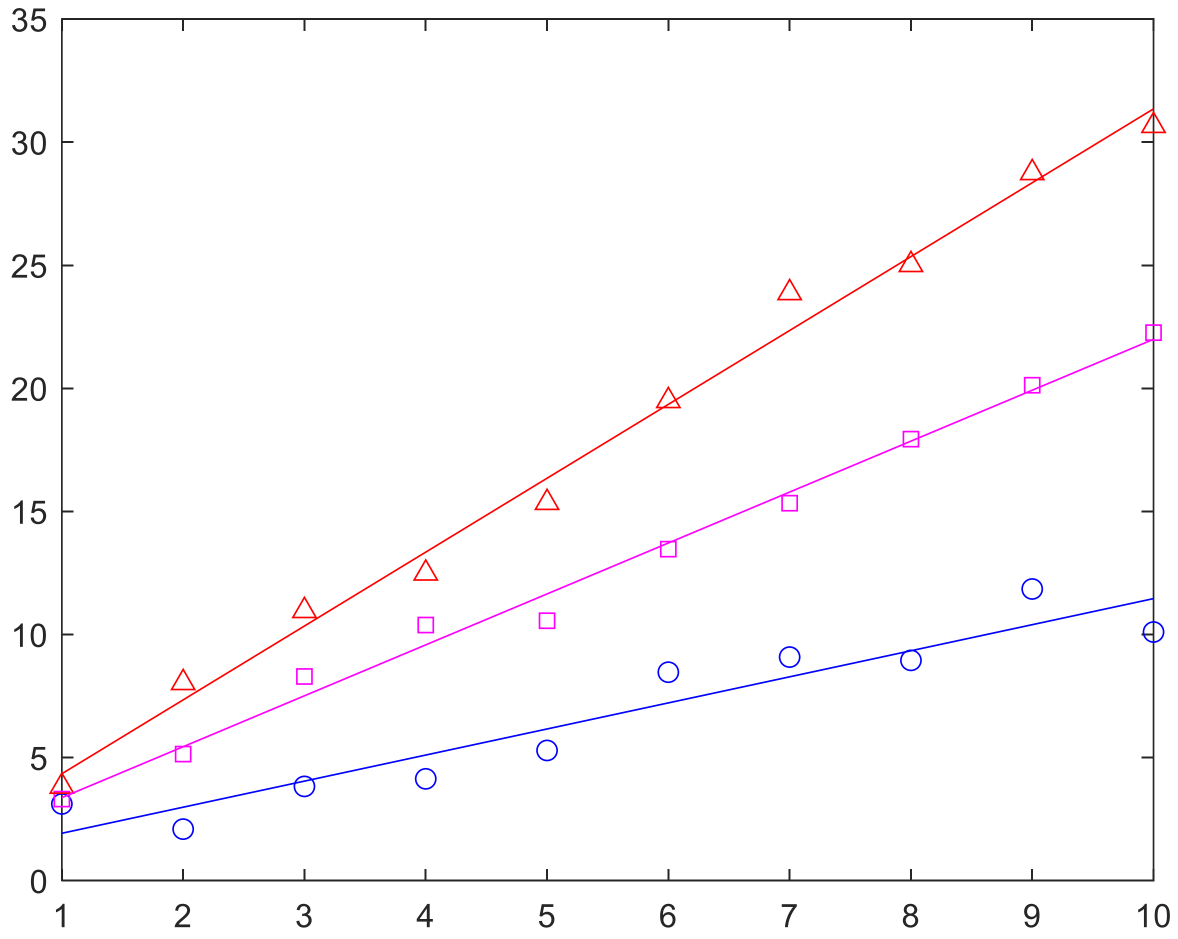
```
x = [1, 2.3, 3.4, 2.6, 6, 5.5, 4];
y = [2, 3, 5.2, 3, 12, 10.5, 6];
figure;
plot(x,y,'o');
lsline;
legend('Data','Least-squares line',...
    'Location','Northwest')
% the location part controls the location of the box
```

Data analysis: polynomial fitting

- lsline can also fit multiple sets of data points on the current plot separately

- Example:

```
x = 1:10;
figure;
y1 = x + rand(1,10)*3;
plot(x,y1,'bo');
hold on
y2 = 2*x + rand(1,10)*3;
plot(x,y2,'ms');
y3 = 3*x + rand(1,10)*3;
plot(x,y3,'r^');
lsline;
```



Data analysis: Other model fitting

- **Fitting with some more general prescribed model**:

$$y = a_1 f_1(x) + a_2 f_2(x) + \cdots + a_k f_k(x)$$

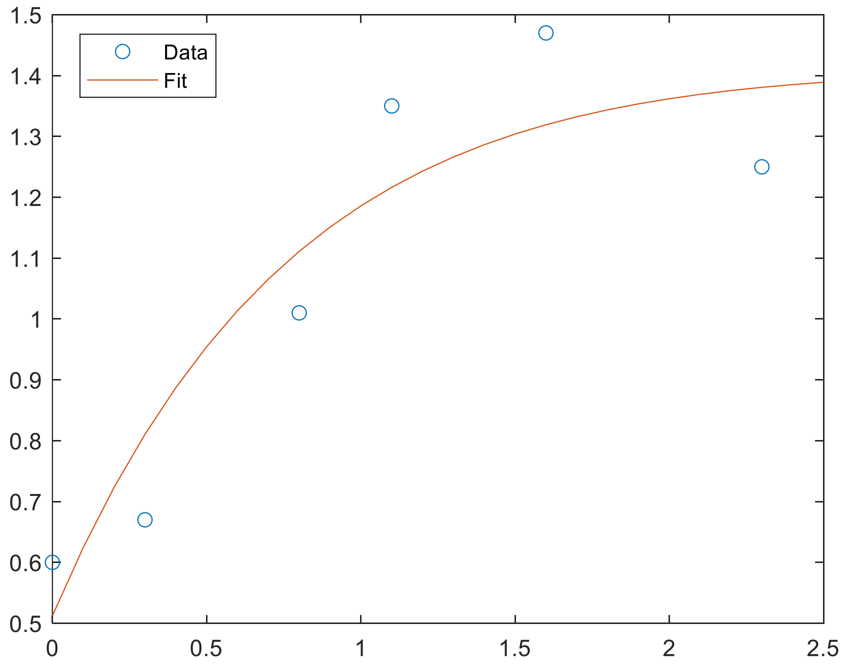
  - Construct column vectors $f_1(x)$, …, $f_k(x)$ and form a matrix M
  - Then use the backslash operator to find the best-fit parameters $a_1, \ldots, a_k$

- Example: Fitting the following data points (t_i, y_i) with $y = a_0 + a_1 e^{-t} + a_2 t e^{-t}$.

```
>> t = [0; 0.3; 0.8; 1.1; 1.6; 2.3];
>> y = [0.6; 0.67; 1.01; 1.35; 1.47; 1.25];
>> M = [ones(size(t)),  exp(-t),  t.*exp(-t)];
>> a = M\y
a =
   1.3983
  -0.8860
   0.3085
```

Best-fit model:

$$y = 1.3983 - 0.8860e^{-t} + 0.3085te^{-t}$$



Data analysis: Multiple regression

- **Multiple regression**: Modelling data with more than one variable

$$y = a_1 f_1(x_1, \dots, x_m) + a_2 f_2(x_1, \dots, x_m) + \cdots + a_k f_k(x_1, \dots x_m)$$

  - Same procedure as before
  - Construct column vectors of $f_1(x_1, \dots, x_m), \dots, f_k(x_1, \dots, x_m)$
  - Then use the backslash operator to find the best-fit parameters $a_1, \dots, a_k$

- Example: Fitting the following data points (x_1, x_2, y) with the model

$$y = a_0 + a_1 x_1 + a_2 \sqrt{x_1} \sin x_2$$

```
>> x1 = [0.2; 0.5; 0.6; 0.8; 1.0; 1.1];
>> x2 = [0.1; 0.3; 0.4; 0.9; 1.1; 1.4];
>> y  = [0.17; 0.26; 0.28; 0.23; 0.27; 0.24];
>> M = [ones(size(x1)), x1, sqrt(x1).*sin(x2)];
>> a = M\y
a =
    0.0831
    0.5256
   -0.3950
```

Best-fit model:

$$y = 0.0831 + 0.5256 x_1 - 0.3950 \sqrt{x_1} \sin x_2$$

Advanced data structures in MATLAB

- So far we have learned:
  - Scalar (1×1 double array): 1, pi, -1.42857, sqrt(2)
  - Vector ($n \times 1$ or $1 \times n$ double array): [1,3.1], [2.1; pi]
  - Matrix ($m \times n$ double array): [1,2; 3,4]
  - Complex double array: [1+2*1i, 3+5.3*1i]
  - Logical array: [1, 0, 0, 1]
  - Character array: 'a' , 'hello'

- Remark: '1' is not equal to 1!
  - '1': a character array
  - 1: a double array

- Conversion between number and character arrays (strings):
  - Number to string: num2str
  - String to number: str2num

Advanced data structures in MATLAB

- Note: All elements of a double/complex/character array must have the **same data type**!

- Example:

```
>> A=[1, 2]   % 1-by-2 double array
A =
     1     2
```

```
>> B=['a', 'b'; 'c', 'd'] % 2-by-2 character array
B =
  2×2 char array
    'ab'
    'cd'
```

- What if we want to store entries with different data types?
  - Cell array
  - Structure array

A new data structure: Cell array

- **Cell arrays** are arrays such that:
  - their entries can be of different types
  - their entries can be accessed by indices

- Basic command for creating a $m \times n$ cell array: A = cell(m,n)
  - Create a cell array with each entry being an empty array [ ]

- The entries of cell arrays are accessed by "{" and "}" rather than "(" and ")"

- Example:

```
>> A = cell(1,3);
>> A{1} = [1,2,3];
>> A{2} = [1,2; 3,4];
A =
  1×3 cell array
   {[1 2 3]}    {2×2 double}    {0×0 double}
```

A	$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	[ ]
	A{1}	A{2}	A{3}

A new data structure: Cell array

- Example:
  Create a cell array called student with surname (character array), given name (character array) and SID (number)

```
>> student = cell(1,3);
>> student{1} = 'Chan';
>> student{2} = 'TaiMan';
>> student{3} = 123456;
>> student
student =
  1×3 cell array
    {'Chan'}    {'TaiMan'}    {[123456]}
```

student	**'Chan'**	**'TaiMan'**	**[123456]**
	student{1}	student{2}	student{3}

A new data structure: Cell array

- Features:
  - Flexible in creation ⇒ no need to worry about incompatible data types/shapes
  - Accessed by indices ⇒ easy to do operations (indexing, reordering, for-loops, …) on the cell array

- Example:

```
S = cell(100,2);
for i = 1:100
    S{i,1} = i;
    if mod(i,3) == 1
        S{i,2} = 'MATH';
    elseif mod(i,3) == 2
        S{i,2} = 'STAT';
    else
        S{i,2} = 'CSCI';
    end
end
```

- Example:

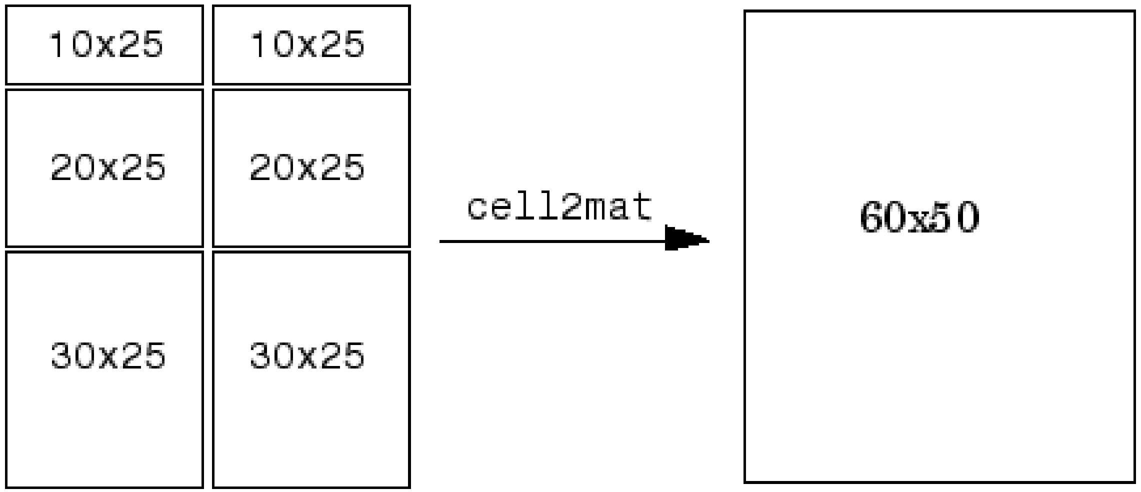
```
>> S([1,50,25,80,71],:)
ans =
  5x2 cell array
    {[ 1]}    {'MATH'}
    {[50]}    {'STAT'}
    {[25]}    {'MATH'}
    {[80]}    {'STAT'}
    {[71]}    {'STAT'}
```

A new data structure: Cell array

- Relevant function for converting a cell array to a numeric array:
  A = cell2mat(C)
  - C: a cell array
  - The elements of the cell array must all contain the same data type
  - The dimensions must be compatible

- Example:

```
>> C = {[1],    [2 3 4];
          [5; 9], [6 7 8; 10 11 12]}
C =
  2×2 cell array
    {[      1]}          {[   2 3 4]}
    {2×1 double}     {2×3 double}
>> A = cell2mat(C)
A =
    1     2     3     4
    5     6     7     8
    9    10    11    12
```



A new data structure: Cell array

- Relevant function for converting a numeric array to a cell array:
  C = mat2cell(A,dim1Dist,...,dimNDist)
  - A: a numeric array
  - dim1Dist, …, dimNDist: specify how to divide the rows, columns etc.

- Example:

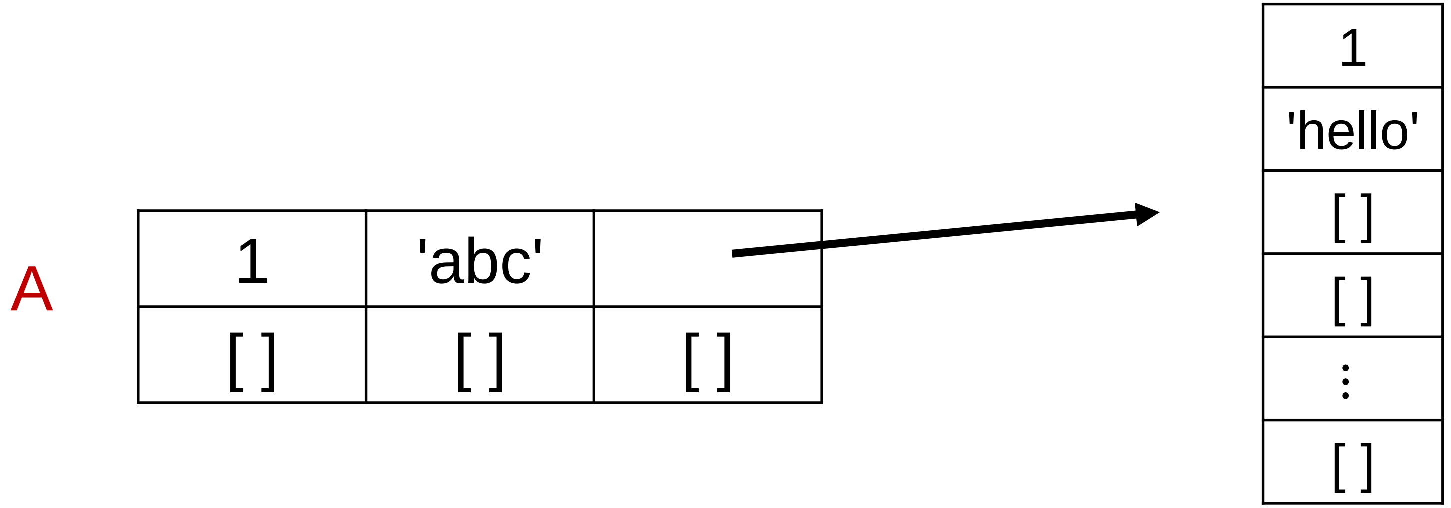
```
>> A = reshape(1:20,5,4)'
A =
    1    2    3    4    5
    6    7    8    9   10
   11   12   13   14   15
   16   17   18   19   20

>> C = mat2cell(A,[1 3],[4 1])
C =
  2×2 cell array
    {[ 1 2 3 4]}      {[      5]}
    {3×4 double}      {3×1 double}
```

Other features of cell array

- We can have cell arrays inside a cell array

- Example:

```
>> A = cell(2,3);
>> A{1,1} = 1;
>> A{1,2} = 'abc';
>> A{1,3} = cell(100,1);        % A{1,3} stores another cell array
>> A{1,3}{1} = 1;               % The first cell of the cell array in A{1,3}
>> A{1,3}{2} = 'hello';         % The second cell of the cell array in A{1,3}
```



Other features of cell array

- We can have sub-cell arrays of a cell array

- Example:

```
>> A = cell(2,3);
>> A{1,1} = 1;
>> A{1,2} = 'abc';
>> A{1,3} = cell(100,1);       % A{1,3} stores another cell array
>> B = A(1,1:2);               % Sub-cell, note that we use ( ) here!
>> B

B =
  1×2 cell array
    {[1]}    {'abc'}
```

A

1	'abc'	100x1 cell
[ ]	[ ]	[ ]

B

1	'abc'

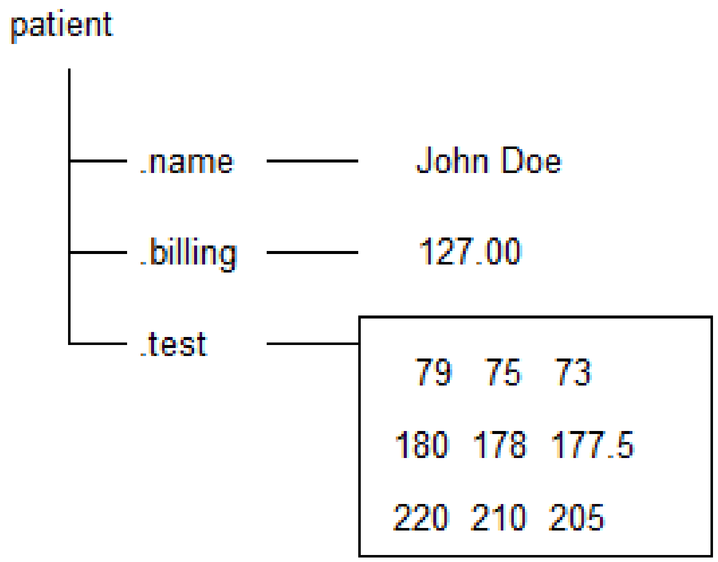
- Subtle difference:
  - A(i,j) gives us a cell array (sub-cell of A)
  - A{i,j} gives us the entry at position i,j (may either be a cell array or a normal array)

Another new data structure: Structure array

- Another way to store multiple data types in one array

- To store values into a structure array, we use the "." operator:
  arrayname.field = value

- Example:
  ```
  >> student = struct; % create a new structure array
  >> student.firstname = 'TaiMan';
  >> student.lastname = 'Chan';
  >> student.id = 123456;
  ```



- Features:
  - Flexible in creation
  - Accessed by field name ⇒ easier to understand

Another new data structure: Structure array

- One can also create a structure array by specifying the fields and values:
  s = struct(field1,value1,...,fieldN,valueN)
  - field1, … fieldN: character arrays representing the field names
  - Values: the content in each field (can be empty)

- Example:

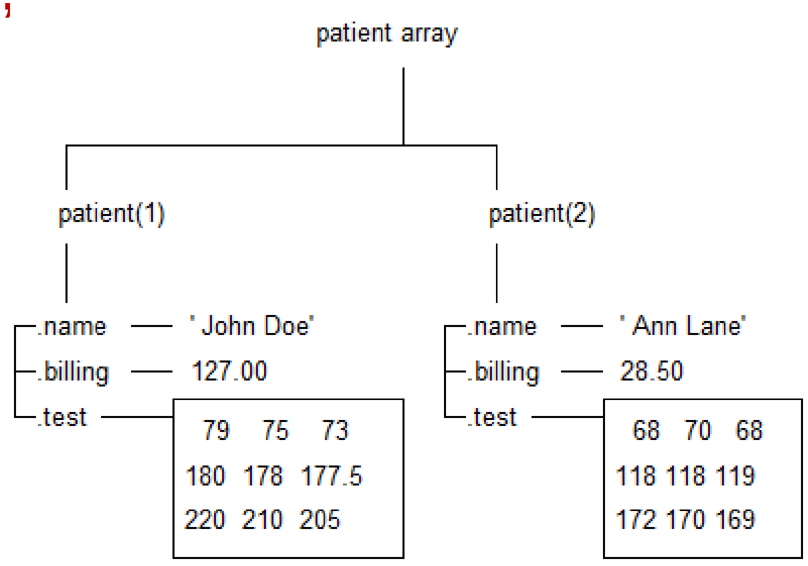
```
>> s = struct('name','TaiMan','ID',[ ])
s =
  struct with fields:
    name: 'TaiMan'
      ID: [ ]
```

Another new data structure: Structure array

- You can create structure array with any size

- Example:
  patient(1).name = 'John Doe';
  patient(1).billing = 127;
  patient(1).test = [79 75 73; 180 178 177.5; 220 210 205];
  patient(2).name = 'Ann Lane';
  patient(2).billing = 28.50;
  patient(2).test = [68 70 68; 118 118 119; 172 170 169];

  Accessing a field of a specific patient:
  \>> patient(1).billing
  ans =
     127



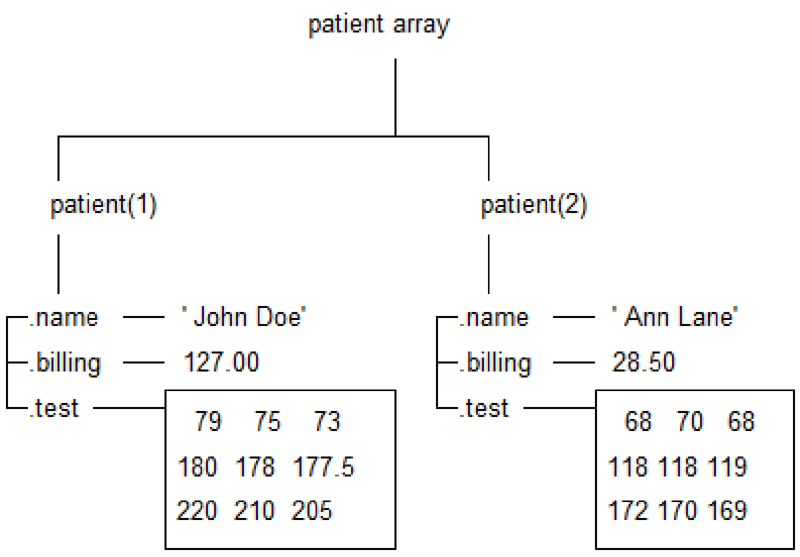
Another new data structure: Structure array

- You can create structure array with any size

- Example:

  Adding a new patient:

```
patient(3).name = 'New Name';
>> patient(3)
ans =
  struct with fields:
        name: 'New Name'
     billing: [ ]
        test: [ ]
```



See more:
https://www.mathworks.com/help/matlab/matlab_prog/create-a-structure-array.html

Data input/output

- How can we save our data generated in MATLAB or load datasets into MATLAB?
  - Simplest way: use the MATLAB-specific .mat format

- Saving data in a .mat file:
  - Save **ALL** variables from the current workspace to a file called filename.mat:
    - save('filename.mat')
  - If we just want to save some of the variables, use:
    - save('filename.mat', 'a', 'b', 'c') % will save the three variables a,b,c only

- Loading data from a .mat file:
  - Load **ALL** variables from filename.mat to your workspace:
    - load('filename.mat')
  - Can also just double-click the .mat file in MATLAB to load it
  - If we just want to load some of the variables, use:
    - load('filename.mat', 'a', 'b', 'c') % will load the three variables a,b,c only

Data input/output

- Saving/loading with other file formats:
  - Excel file (.xls or .xlsx): xlsread, xlswrite
  - CSV file (.csv): csvread, csvwrite

- More generally,
  - For reading/writing numeric data from .txt, .dat, .csv, .xls, .xlsx:
    - readmatrix
    - writematrix

  - For reading/writing cell data from .txt, .dat, .csv, .xls, .xlsx:
    - readcell
    - writecell

- For even more flexible input/output, see:
  - fopen, fclose, textscan, fprintf

Reminder: Lab 7 this week

January

Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	[28]	[29]	[30]	[31]	

February

Sun	Mon	Tue	Wed	Thu	Fri	Sat
						[1]
[2]	[3]	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	1

March

Sun	Mon	Tue	Wed	Thu	Fri	Sat
2	[3]	[4]	[5]	[6]	[7]	[8]
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

April

Sun	Mon	Tue	Wed	Thu	Fri	Sat
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17		

Lecture 1- Lecture 13

Lab 1 - Lab 10 (40%)

Test 1 (30%)
Test 2 (30%)

Thank you!

Next time:

- Image and video processing using MATLAB