Miloslav Čapek
miloslav.capek@fel.cvut.cz
Viktor Adler, Filip Kozák, Pavel Valtr

Department of Electromagnetic Field
B2-634, Prague
Learning how to ...

Set operations

Sorting

Searching

\[ D = A \cap B \cap C \]
\[ A \cap B = \{ x : x \in A \land x \in B \} \]
Set operations

- there exist following operations (operators) in Matlab applicable to arrays or individual elements
  - arithmetic (part #1)
  - relational (part #4)
  - logical (part #4)
  - set (part #7)
  - bit-wise (help, >> doc)

- set operations are applicable to vectors matrices, arrays, cells, strings, tables, ... 
  - mutual sizes of these structures are usually not important

<table>
<thead>
<tr>
<th>Intersection of two sets</th>
<th>intersect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union of two sets</td>
<td>union</td>
</tr>
<tr>
<td>Difference of two sets</td>
<td>setdiff</td>
</tr>
<tr>
<td>Exclusive OR of two sets</td>
<td>setxor</td>
</tr>
<tr>
<td>Unique values in a set</td>
<td>unique</td>
</tr>
<tr>
<td>Sorting, row sorting</td>
<td>sort, sortrows</td>
</tr>
<tr>
<td>Is the element member of a set?</td>
<td>ismember</td>
</tr>
<tr>
<td>Is the set sorted?</td>
<td>issorted</td>
</tr>
</tbody>
</table>
Set operations #1

- **intersection of sets**: `intersect`
  - example: intersection of a matrix and a vector:

  ```
  >> A = [1 -1; 3 4; 0 2];
  >> b = [0 3 -1 5 7];
  >> c = intersect(A, b)
  % c = [-1; 0; 3]
  ```

- **union of sets**: `union`
  - all set operations can be carried out row-wise
    (in that case the number of columns has to be observed)

  ```
  >> A = [1 2 3; 4 5 1; 1 7 1];
  >> b = [4 5 1];
  >> C = union(A, b, 'rows')
  % C = [1 2 3; 1 7 1; 4 5 1]
  ```
Set operations #2

- intersection of a set and complement of another set: `setdiff`
  - all set operations return more than one output parameter - we get the elements as well as the indexes

```matlab
>> A = [1 1; 3 NaN];
>> B = [2 3; 0 1];
>> [C, ia, ib] = setdiff(A, B)
% C = NaN, ia = 4
% i.e.: C = A(ia)
```

- exclusive intersection (XOR): `setxor`
  - all set operations can be carried out either as 'stable' (not changing the order of elements) or as 'sorted' (elements are sorted)

```matlab
>> A = [5 1 0 4];
>> B = [1 3 5];
>> [C, ia, ib] = setxor(A, B, 'stable')
% C = [0 4 3], ia = [3; 4], ib = [2]
```
Set operations #3

- selection of unique elements of an array: `unique`
  - set operations are also applicable to arrays not (exclusively) containing numbers

```matlab
>> A = {'Joe', 'Tom', 'Sam'};
>> B = {'Tom', 'John', 'Karl', 'Joe'};
>> C = unique([A B])
% C = {'John', 'Karl', 'Joe', 'Sam', 'Tom'}
```

- it is possible to combine all above mentioned techniques
  - e.g. row-wise listing of unique elements of a matrix including indexes:

```matlab
>> A = round(rand(10, 3)).*mod(10:-1:1, 3)'
>> [C, ai, ci] = unique(sum(A, 2), 'rows', 'stable')
```

- Interpret the meaning of the above code? Is the 'rows' parameter necessary?
Set operations #1

- consider three vectors \(a, b, c\) containing natural numbers \(x \in \mathbb{N}\) so that
  - vector \(a\) contains all primes up to (and including) 1000
  - vector \(b\) contains all even numbers up to (and including) 1000
  - vector \(c\) is complement of \(b\) in the same interval

- find vector \(v\) so that \(v = a \cap (b + c)\), \(b + c \equiv \{b_i + c_i\}, \quad i \in \{1,500\}\)
  - what elements does \(v\) contain?
  \(v = \quad b_{i-1} < b_i < b_{i+1} \land c_{i-1} < c_i < c_{i+1}, \quad \forall i\)

- how many elements are there in \(v\)?

Columns 1 through 24

\[
\begin{array}{cccccccccccc}
3 & 7 & 11 & 19 & 23 & 31 & 43 & 47 & 59 & 67 & 71 & 79 \\
\end{array}
\]

Columns 25 through 48

\[
\begin{array}{cccccccccccc}
11 & 223 & 227 & 239 & 251 & 263 & 271 & 283 & 307 & 311 & 331 & 347 \\
\end{array}
\]

Columns 49 through 72

\[
\begin{array}{cccccccccccc}
491 & 499 & 503 & 523 & 547 & 563 & 571 & 587 & 599 & 607 & 619 & 631 \\
\end{array}
\]

Columns 73 through 87

\[
\begin{array}{cccccccccccc}
\end{array}
\]
Set operations #2

• estimate the result of following operation (and verify using Matlab):

\[ w = (b \cup c) \setminus a \]

• what is specific about elements of the resulting vector \( w \)?

• with the help of logical indexing and mathematical functions determine how many elements of \( w \) are divisible by 3
Set operations #3

- write previous exercise as a script:

- modify the script in the way to calculate how many elements of w are divisible by numbers 1 to 20
  - use for instance for loop to get the result
  - plot the results using bar function
Set operations #4

Figure 1

Operators
Set operations #5

- Radio relay link operates at frequency of 80 GHz at 20 km distance with 64-QAM modulation
  - phase stability of ±0.5° is required for sufficiently low bit error rate without using synchronization and coding
  - that corresponds to the change of distance between antennas equal to ±5 μm
  - the statistics of link distance with normal distribution containing $1 \cdot 10^6$ elements can be generated as:

    ```
    L = 20e3; % length of path
deviation = 5e-6; % standard deviation
N = 1e6; % number of trials
% random distances
distances = L + randn(1, N)*deviation;
    ```

- How many times is the distance $L$ contained in the vector `distances`?
- How many unique elements are there in `distances`?
- Can the distribution be considered continuous?
Array sorting #1

- sort array elements
  - column-wise, in ascending order:
    
    >> sort(A)

  - row-wise, in ascending order:
    
    >> sort(A, 2)

  - in descending order:
    
    >> sort(A, 'descend')

  - in descending order, row-wise:
    
    >> sort(A, 2, 'descend')

- apply the sorting function, to following matrices (for instance):

  >> A = reshape([magic(3) magic(3)], [3 3 2])
  >> B = 'for that purpose';
Array sorting #2

- **function** `sortrows` sorts rows of a matrix
  - elements of the rows are not swapped - rows are sorted as blocks

\[
\begin{pmatrix}
8 & 1 & 6 \\
3 & 5 & 7 \\
4 & 9 & 2
\end{pmatrix}
\]

**SORT:**
\[
\begin{pmatrix}
3 & 1 & 2 \\
4 & 5 & 6 \\
8 & 9 & 7
\end{pmatrix}
\]

**SORTROWS:**
\[
\begin{pmatrix}
3 & 5 & 7 \\
4 & 9 & 2 \\
8 & 1 & 6
\end{pmatrix}
\]
is* functions related to sets

- function `issorted` returns true if array is sorted

- function `ismember(A, B)` tests whether an element of array B is also an element of array A

```matlab
>> ismember([1 2 3; 4 5 6; 7 8 9], [0 0 1; 2 1 4])
ans =
    1 1 0
    1 0 0
    1 0 0
    0 0 0
```
Array sorting

- try to write your own sorting algorithm `bubbleSort.m`
  - use the *bubble sort* algorithm
  - use the function `issorted` to test whether the resulting array is sorted

if you wish, you can use the following code inside loops:

```matlab
figure(1);
plot(R,'*','LineWidth',2);
pause(0.01);
```

`sort(R)`
Array sorting

- try to get plot as in the figure using `bar` function:
Array sorting – shaker sort

• try to write your own sorting algorithm `shakerSort.m`
• use the *shaker sort* algorithm

Operators
Searching in an array – find

- find function is a very useful one!!
- returns positions of non-zero (logical true) elements of a matrix
  - useful for searching in an array of logical values
  - example: find positions of those elements of vector \( \mathbf{A} = \left( \frac{\pi}{2}, \pi, \frac{3}{2}\pi, 2\pi \right) \) fulfilling the condition \( \mathbf{A} > \pi \)

\[
\begin{pmatrix}
0 & 1 & 1 & 0 \\
\end{pmatrix}
\]

- compare the above command with \( \mathbf{A} > \pi \). What is the difference?
- function find can also search a square matrix etc.
- to find first / last \( k \) non-zero elements of \( \mathbf{X} \):

\[
\begin{pmatrix}
0 & 1 & 1 & 0 \\
\end{pmatrix}
\]

- for more see >> doc find

\[
\begin{pmatrix}
0 & 1 & 1 & 0 \\
\end{pmatrix}
\]
Array searching #1

- sort the vector \( v = (16 \ 2 \ 3 \ 13 \ 5 \ 11 \ 10 \ 8 \ 9 \ 7 \ 6 \ 12 \ 4 \ 14 \ 15 \ 1) \) in descending order and find the elements of the vector (and their respective positions within the vector) that are divisible by three and at the same time are greater than 10

\[
\begin{align*}
\gg v &= \text{reshape(magic(4)', [1 numel(magic(4))])}
\end{align*}
\]

\[
\begin{align*}
v &=
\begin{bmatrix}
16 & 2 & 3 & 13 & 5 & 11 & 10 & 8 & 9 & 7 & 6 & 12 & 4 & 14 & 15 & 1 \\
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
v1 &=
\begin{bmatrix}
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{ans} &=
\begin{bmatrix}
15 & 12 \\
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{ans} &=
\begin{bmatrix}
2 & 5 \\
\end{bmatrix}
\end{align*}
\]
Array searching #2

- in matrix \( w \)
  
  >>> \( w = (8:-1:2)'*(1:1/2:4).*\text{magic}(7) \)

find last 3 values that are smaller than 50
- find out the column and row positions of the values

\[ w = \]

\[
\begin{array}{cccccccc}
240.0000 & 469.0000 & 768.0000 & \boxed{20.0000} & 260.0000 & 532.0000 & 896.0000 \\
266.0000 & 493.5000 & 98.0000 & 157.5000 & 378.0000 & 661.5000 & 812.0000 \\
276.0000 & 54.0000 & 96.0000 & 255.0000 & 469.0000 & 735.0000 & 888.0000 \\
25.0000 & 105.0000 & 160.0000 & 312.5000 & 510.0000 & 630.0000 & 900.0000 \\
52.0000 & 90.0000 & 192.0000 & 330.0000 & 594.0000 & 616.0000 & 64.0000 \\
63.0000 & 103.5000 & 192.0000 & 307.5000 & 387.0000 & \boxed{31.5000} & 144.0000 \\
44.0000 & 93.0000 & 160.0000 & 245.0000 & \boxed{12.0000} & 77.0000 & 160.0000 \\
\end{array}
\]
Application of the \texttt{find} function

- Samples of demodulated signal of a radio receiver can be approximated as:

\[ w = 0.6833; t = 1:10; \texttt{\% time} \]
\[ \text{samples} = 2.7 + 0.5*(\cos(w*t) - \sin(w*t) - \cos(2*w*t) + \sin(2*w*t)) \ldots \]
\[ - \cos(3*w*t) + 3*\sin(3*w*t) + 2*\cos(4*w*t) + 4*\sin(4*w*t)); \]
\[ \text{plot(samples, 'x')} \]

- Voltage corresponding to characters are within ±0.5 V tolerance
- Decipher the message!

\begin{verbatim}
w = 0.6833; t = 1:10; \texttt{\% time}
samples = 2.7 + 0.5*(\cos(w*t) - \sin(w*t) - \cos(2*w*t) + \sin(2*w*t) \ldots 
- \cos(3*w*t) + 3*\sin(3*w*t) + 2*\cos(4*w*t) + 4*\sin(4*w*t)); 
plot(samples, 'x')

chars = 'acdgmrs'; volts = 1:7;
message = blanks(length(samples));
for iVolt = volts
    logCondition = samples > (iVolt - 0.5) & \ldots 
    samples < (iVolt + 0.5);
    indices = find(logCondition);
    message(indices) = chars(iVolt);
end
disp(message)
\end{verbatim}
Function accumarray #1

- the function `accumarray` is able to group data with the same index
- not a very well known function, but an exceptionally useful one
- quite often we deal with a dataset that is organised in the following way:

\[
\begin{align*}
\text{indexes (e.g. measurement number)} & \quad \text{values (measured)} \\
1 & 0.1 \\
1 & 0.3 \\
2 & 1.0 \\
3 & -3.1 \\
1 & 0.2 \\
2 & 1.1 \\
4 & 10.1 \\
4 & 10.2 \\
\end{align*}
\]

>> ind = [1 1 2 3 1 2 4 4].';
>> data = [.1 .3 1 -3.1 .2 1.1 10.1 10.2].';
>> Dtal = accumarray(ind, data)

\[
\begin{align*}
\text{Dtal} & = 0.6000 \quad (\text{ind} == 1) \\
& = 2.1000 \quad (\text{ind} == 2) \\
& = -3.1000 \quad (\text{ind} == 3) \\
& = 20.3000 \quad (\text{ind} == 4)
\end{align*}
\]
Function `accumarray` #2

- basic operation applicable to data from one 'box' (data with the same index) is summation
- any other function can be applied, however
  - e.g. maximum of a set of elements with the same index
  - we use the `max` function

```
>> Dta2 = accumarray(ind, data, [], @max)
```

```
Dta2 =
  0.3000
  1.1000
-3.1000
 10.2000
```

- e.g. listing of all elements with the same index
- we use so called handle function and `cell` data type

```
>> Dta3 = accumarray(ind, data, [], @(x) {x})
```

```
Dta3 =
[3x1 double]
[2x1 double]
[-3.1000]
[2x1 double]
```
Function accumarray #3

- the function has a wide variety of other features
- it is possible, for instance, to use 2D indexation of results
  - the results are not put in a 1D set of 'boxes' but to a 2D array instead

\[
\begin{align*}
\text{>> } \text{ind} &= [1 \ 1; 2 \ 1 \ 2; 1 \ 3; 1 \ 1; 3 \ 1]; \\
\text{>> } \text{data} &= [10 \ 22 \ 12 \ 13 \ 1 \ \pi]; \\
\text{>> } \text{Dta4} &= \text{accumarray}\text{(ind, data)}
\end{align*}
\]

\[
\begin{array}{ccc}
\text{ind} &= [1 \ 1] & \text{ind} &= [1 \ 2] & \text{ind} &= [1 \ 3] \\
10 + 1 &= \textbf{11} & \textbf{12} & \textbf{13} \\
\text{ind} &= [2 \ 1] & \text{ind} &= [2 \ 2] & \text{ind} &= [2 \ 3] \\
0 &= & \textbf{22} & \textbf{0} \\
\text{ind} &= [3 \ 1] & \text{ind} &= [3 \ 2] & \text{ind} &= [3 \ 3] \\
\pi &= & \textbf{0} & \textbf{0}
\end{array}
\]
Function `accumarray`:

- account transfers in CZK, EUR a USD are as follows
  - (CZK ~ 1, EUR ~ 2, USD ~ 3)

- find out account balance in each currency
  - the exchange rate is 26 CZK = 1€, 25 CZK = 1$, find out total balance

\[
\begin{pmatrix}
1 & -110 \\
1 & -140 \\
2 & -22 \\
3 & -2 \\
2 & -34 \\
1 & -1300 \\
2 & -15 \\
1 & -730 \\
3 & 24
\end{pmatrix}
\]

\[
>> \text{dta} = [1 \ -110; \ 1 \ -140; \ 2 \ -22; \ 3 \ -2; \ldots \ 2 \ -34; \ 1 \ -1300; \ 2 \ -15; \ 1 \ -730; \ 3 \ 24]
\]

\[
>> \text{K} = [1 \ 26 \ 25]
\]
## Discussed functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersect</td>
<td>intersection of sets (vectors / matrices)</td>
</tr>
<tr>
<td>union</td>
<td>intersection of sets (vectors / matrices)</td>
</tr>
<tr>
<td>setdiff</td>
<td>Subtraction of sets (intersection of a set and complement of another set)</td>
</tr>
<tr>
<td>setxor</td>
<td>exclusive intersection</td>
</tr>
<tr>
<td>unique</td>
<td>selection of unique elements of an array</td>
</tr>
<tr>
<td>sort</td>
<td>sort vector/matrix elements</td>
</tr>
<tr>
<td>sortrows</td>
<td>sorts rows of a matrix as a whole</td>
</tr>
<tr>
<td>accumarray</td>
<td>group data</td>
</tr>
<tr>
<td>ismember</td>
<td>is given element is member of array?</td>
</tr>
<tr>
<td>issorted</td>
<td>is array sorted?</td>
</tr>
<tr>
<td>find</td>
<td>find elements fulfilling given condition</td>
</tr>
</tbody>
</table>
Exercise #1

- measurement of temperature was carried out in the course of 5 days every second clock hour. Data was measured at 3 different sites (A, B, C)

- find out average daily temperature in given week for all 3 sites
  - i.e., get mean value of measurement at the same hour on the same site

- generate the data using `temperature_measurement.m`
  - see the script on the following slide
  - see the variables required
Exercise #2

script for data generation

clear; close all; clc;

%% allocation
days = 5; hours = 12;
TimeA = zeros(days*hours,1);
TimeB = TimeA;
TimeC = TimeA;

%% creation of time data-set
for kDay=1:days
    TimeA((hours*(kDay-1)+1):(hours*(kDay-1)+12),1) = 2*(randperm(12)-1)';
    TimeB((hours*(kDay-1)+1):(hours*(kDay-1)+12),1) = 2*(randperm(12)-1)';
    TimeC((hours*(kDay-1)+1):(hours*(kDay-1)+12),1) = 2*(randperm(12)-1)';
end

%% place and temperature data-sets
PlaceA = abs(abs(TimeA - 11) - 10) + 10 + 5.0*rand(size(TimeA,1),1);
PlaceB = abs(abs(TimeB - 12) - 10) + 5 + 10.0*rand(size(TimeB,1),1);
PlaceC = abs(abs(TimeC - 11) - 11) + 5 + 7.5*rand(size(TimeC,1),1);

%% generating final variables for the example
TimeAndPlace = [TimeA/2+1 ones(size(TimeA,1),1);...;
                TimeB/2+1 2*ones(size(TimeA,1),1);...;
                TimeC/2+1 3*ones(size(TimeA,1),1)];

MeasuredData = [PlaceA; PlaceB; PlaceC];

%% plot final data-set
plot(TimeA,PlaceA,'LineWidth',1,'LineStyle','none','Marker','x',... 'MarkerSize',15); hold on;
plot(TimeB,PlaceB,'LineWidth',1,'LineStyle','none','Marker','**',... 'MarkerSize',15,'Color','r');
plot(TimeC,PlaceC,'LineWidth',2,'LineStyle','none','Marker','o',... 'MarkerSize',10,'Color','g');
set(gcf,'Color','w','pos',[50 50 1000 600]); set(gca,'FontSize',15);
xlabel('time','FontSize',15); ylabel('Temperature','FontSize',15);
title('Measured Data'); grid on; legend('Place A','Place B','Place C');

and the results …
Exercise #3

- all the data are contained in 2 matrices:
  - TimeAndPlace \((5 \times 3 \times 12, 2) = (180, 2)\)
  - MeasuredData \((5 \times 3 \times 12, 1) = (180, 1)\)

- unfortunately, data in TimeAndPlace are intentionally unsorted

<table>
<thead>
<tr>
<th>INDEXES:</th>
<th>TimeAndPlace</th>
<th>MeasuredData</th>
<th>DATA:</th>
</tr>
</thead>
<tbody>
<tr>
<td>tindex = 10, Place = 1</td>
<td>10 1</td>
<td>15.0797</td>
<td>T(10,1) = 15.0797 °C</td>
</tr>
<tr>
<td></td>
<td>4 1</td>
<td>18.9739</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 1</td>
<td>19.3836</td>
<td></td>
</tr>
<tr>
<td></td>
<td>... ...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12 2</td>
<td>9.9506</td>
<td></td>
</tr>
<tr>
<td>tindex = 6, Place = 2</td>
<td>6 2</td>
<td>19.7588</td>
<td>T(6,2) = 19.7588 °C</td>
</tr>
<tr>
<td></td>
<td>... ...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Exercise #4

- following holds true
  - Place 1 ~ measurement site A
  - Place 2 ~ measurement site B
  - Place 3 ~ measurement site C
  - measurement hour = 2 \times (tindex - 1)

- now try to place your code in the script to carry out the averaging and plot the data in the existing figure

```matlab
%% PLACE YOUR CODE HERE
%==========================================================================
% ...
% dataA = ...
% dataB = ...
% dataC = ...
%==========================================================================

%% plot the averaged data
plot(0:2:22,dataA,'LineWidth',2,'Color','b','LineStyle','-');
plot(0:2:22,dataB,'LineWidth',2,'Color','r','LineStyle','-');
plot(0:2:22,dataC,'LineWidth',2,'Color','g','LineStyle','-');
```
Exercise #5

measured data

measured and averaged data