

Lecture 10: Timer, Sorting, Finding

BE0B17MTB – Matlab

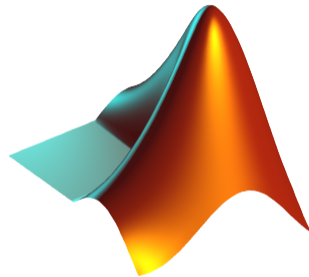
Miloslav Čapek, Viktor Adler, Pavel Valtr, Michal Mašek, and Vít Losenický

Department of Electromagnetic Field
Czech Technical University in Prague
Czech Republic
valtrp@fel.cvut.cz

December 5, 2019
Winter semester 2019/20



1. Time Functions
2. Sorting
3. Finding
4. Exercises





Class `timer`

- ▶ If it is desired to cyclically repeat an action, it is possible to use class `timer`.
 - ▶ Better possibility compared to infinite loop.
- ▶ Great advantage is the fact that `timer` creates its own thread.
 - ▶ It is possible to keep on working with MATLAB on launching, or alternatively launch another `timer`,
 - ▶ it is possible to keep on Working with Matlab even as `timer` is still running,
 - ▶ it is not possible to terminate the thread using CTRL+C.
- ▶ For more information see `>> doc timer`.



Class timer – Demonstration

Example: To display date & time in 1 s interval:

```
tm = timer; % create an instance of timer
tic;
tm.ExecutionMode = 'fixedRate';
tm.TimerFcn = 'disp(datetime); toc;';
start(tm); % start the timer
```

► To stop the timer, use: `>> stop(tm);`

```
27-Nov-2019 13:13:47
Elapsed time is 0.314328 seconds.
27-Nov-2019 13:13:48
Elapsed time is 1.025306 seconds.
27-Nov-2019 13:13:49
Elapsed time is 2.024553 seconds.
27-Nov-2019 13:13:50
Elapsed time is 3.020616 seconds.
27-Nov-2019 13:13:51
Elapsed time is 4.040701 seconds.
27-Nov-2019 13:13:52
Elapsed time is 5.017700 seconds.
27-Nov-2019 13:13:53
Elapsed time is 6.018114 seconds.
27-Nov-2019 13:13:54
Elapsed time is 7.018261 seconds.
27-Nov-2019 13:13:55
Elapsed time is 8.018187 seconds.
>>
```



Class timer – Control Graphics in Separate Thread

```

myLine = line([0 0], [0 0]); view(45, 45); box on;
xlim([-1 1]); ylim([-1 1]); zlim([-1 1]);

thisTimer = timer;           % create timer
thisTimer.StartDelay        = 1; % wait 1 second
thisTimer.Period            = 0.1; % repeat action after 0.1s
thisTimer.ExecutionMode    = 'fixedSpacing'; % spacing
thisTimer.UserData         = 0; % data which we need...
thisTimer.TimerFcn         = {@timer_update, myLine, pi/16};
start(thisTimer);           % start the timer...

fghndl = gcf;                % stop timer if the figure is closed
fghndl.CloseRequestFcn = 'stop(thisTimer); closereq;';

function timer_update(myTimer, ~, myLine, dPhi)
myLine.XData = [1 -1]*sin(myTimer.UserData);
myLine.YData = [1 -1]*cos(myTimer.UserData);
drawnow('update');          % update graphics
myTimer.UserData = myTimer.UserData + dPhi;
end

```



Class timer – Hello World!

- ▶ Create a timer that displays with 0.5 s interval “XX | Hello world!” message, where XX is the number of the message being displayed. The timer will be terminated after reaching 15 displays.





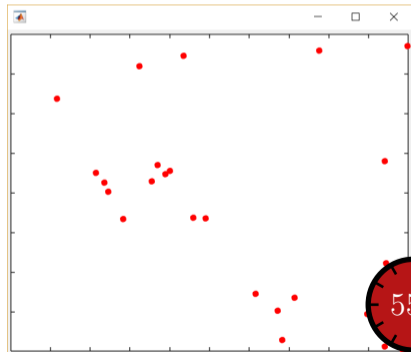
Class timer – Capture Pointer Location I.a

- ▶ Create a code which periodically plots a marker into axes of size `axSize` (1×2 vector) each 0.5 second. Marker will be placed at a position proportional to the location of the pointer on a screen, *i.e.*, at a position
`pos = pointerLocation ./ LCDresolution .* axSize.`
- ▶ Start with editing the following code snippet:

```
clear; clc;

axSize = [500 400];

hFig = uifigure('color', 'white', ...
    'pos', [50 50 axSize]);
hAxes = uiaxes('parent', hFig, ...
    'pos', [0 0 axSize], 'box', 'on', ...
    'XTickLabel', '', 'YTickLabel', '');
hAxes.XLim = [0 1];
hAxes.YLim = [0 1];
%% ADD YOUR CODE HERE....
```



Class `timer` – Capture Pointer Location I.b



- ▶ The completion of the code snippet from the previous slide:



Time Functions in MATLAB

There is a whole variety of time functions but just one of them is enough to measure time duration of a function.

Function	Description
<code>tic – toc</code>	measures time duration between commands <code>tic</code> and <code>toc</code>
<code>clock</code>	returns vector [year month day hour minute seconds]
<code>date</code>	returns date in format <code>dd-mmm-yyyy</code> , variable is of type <code>char</code> (text)
<code>etime</code>	returns time duration between <code>t1</code> and <code>t2</code> , <code>etime(t2,t1)</code>
<code>cputime</code>	returns total CPU time in seconds used by MATLAB application
<code>now</code>	returns current date and time as an integer
<code>timeit</code>	measures time required to run function

```
tic
% code
toc
```

```
t0 = tic;
% code
t1 = toc(t0)
```



Time Functions in MATLAB – Profiling

- ▶ What is the way to measure how long it takes for a program to be executed?
 - ▶ Distinguish between reasonably fast and very fast codes:

```
tic
% code
toc
```

```
tic
  for k =
      1:100
      % code
  end
toc
```

- ▶ Other options? Which one is the best?
- ▶ Mathworks recommends functions `tic` – `toc` (esp. for new CPUs with hyper-threading).

```
t0a = tic;
fft(x);
toc(t0a)
```

```
t0b = clock;
fft(x);
etime(clock, t0b)
```

```
t0c = cputime;
fft(x);
e = cputime - t0c;
```



Time Functions in MATLAB – Advanced Features

- ▶ Conversions between individual ways of displaying date in MATLAB:
 - ▶ Functions `datevec`, `datenum`, and `datestr`.
 - ▶ This is how to transform date into standard form: `>> datevec(now)`.
- ▶ Day of week: `>> weekday(date)`,
 - ▶ notice US way of counting days (1st day = Sunday).
- ▶ The last day of month: `>> eomday(2014, 1:12)`.
- ▶ Calendar: `>> calendar`.

```
>> C = calendar
C =
     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     0     0     0     0
     0     0     0     0     0     0     0

>>
```



Time Functions in MATLAB – Duration

- ▶ Time duration can be generated using functions like `days`, `hours`, `minutes`, etc.
- ▶ Numeric arrays can be converted into duration using functions like `caldays`, etc.

```
randArray = randi(5,1,5);
randDays = caldays(randArray)

nextThreeDays = datetime(...
    'tomorrow', 'Format', ...
    'dd-MMM-yyyy HH:mm:ss') + ...
    days(1:3)
nextThreeDays + randDays(1)
```

```
randDays = |
1×5 calendarDuration array
    5d    5d    1d    5d    4d

nextThreeDays =
1×3 datetime array
    06-Dec-2019 00:00:00    07-Dec-2019 00:00:00    08-Dec-2019 00:00:00

ans =
1×3 datetime array
    11-Dec-2019 00:00:00    12-Dec-2019 00:00:00    13-Dec-2019 00:00:00
```

- ▶ For details see MATLAB → Language Fundamentals → Data Types → Dates and Time.

Time Functions in MATLAB



- ▶ Find the date of the ten thousandth day of your life.
- ▶ What a day of the week is it? (Because of party...)

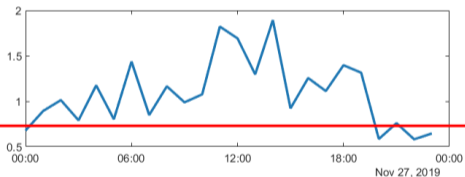




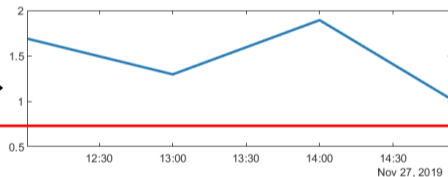
Time Series Data

- ▶ Having data as a function of time, it is possible to display the data as a time series.

```
d = datetime(2019, 11, 27, 0:23, 0, 0);
fx = sin(linspace(0, pi, 24)) + rand(1,24);
plot(d, fx)
```



zoom



- ▶ For details see MATLAB → Language Fundamentals → Data Types → Dates and Time.



Time Data

- ▶ Time entries created using `datetime` are not of class `double`, but of class `datetime`.
 - ▶ It is possible to specify time zones/difference.
 - ▶ All functions support vectorizing.

```
t = datetime
t.Format
t.Format = 'd-h' % nonsense but possible
```

```
t1 = datetime('22/09/15 17:00:00');
t2 = datetime('24/12/15 19:00:00');
t = t1:days(7):t2
```

- ▶ It is possible to create and work with time intervals as well (class `duration`).

```
tInt = days(5) + hours(10)
2*tInt - days(4) + 4*minutes
```

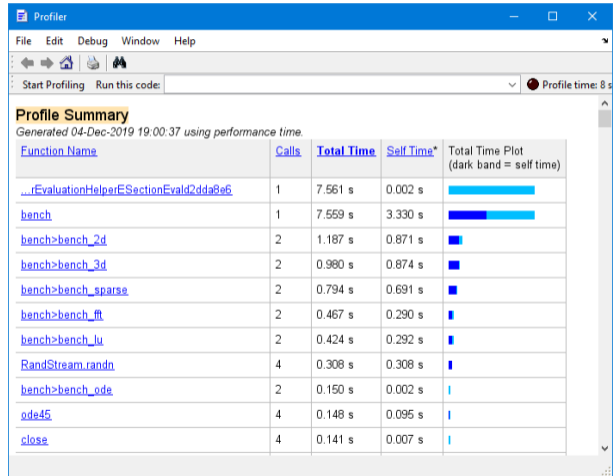
Format	Result
'yyyy-MM-dd'	2019-04-20
'dd/MM/yyyy'	19/04/2020
'dd.MM.yyyy'	19.04.2020
'MMMM d, yyyy'	April 19, 2020
	and many others...



MATLAB Profile

- ▶ Profile execution time for functions.
- ▶ Extremely powerful tool!

```
profile on;
% code to profile
profile off;
profile viewer;
```





Array Sorting I.

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')`

- ▶ The function `sort` can be applied to matrices like

```
A = reshape([magic(3) magic(3)'], [3 3 2]);  
B = 'for that purpose';
```



Array Sorting II.

- ▶ Function `sortrows` sorts rows of a matrix.
 - ▶ Elements of the rows are not swapped – rows are sorted as blocks.

$$\begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}$$

`sort()`

$$\begin{bmatrix} 3 & | & 1 & | & 2 \\ 4 & | & 5 & | & 6 \\ 8 & | & 9 & | & 7 \end{bmatrix}$$

`sortrows()`

$$\begin{bmatrix} 3 & 5 & 7 \\ 4 & 9 & 2 \\ 8 & 1 & 6 \end{bmatrix}$$



is* Functions Related to Sets

- ▶ Function `issorted` returns true if array is sorted.
- ▶ Function `ismember(A,B)` tests whether elements of array B are also elements of array A.

```
ismember([1 2 3; 4 5 6; 7 8 9], [0 0 1; 2 1 4])
```

```
>> ismember([1 2 3; 4 5 6; 7 8 9], [0 0 1; 2 1 4])
```

```
ans =
```

```
3x3 logical array
```

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

```
>> |
```



Summary of `is*` Functions

- ▶ Asterisk stands for whole range of functions...
 - ▶ They return logical (`true/false`) value(s).
- ▶ A selection of the interesting ones (some even have multiple parameters):

Function	Description
<code>ischar</code>	Determines whether item is character array.
<code>isempty</code>	Determines whether array is empty.
<code>isfinite</code>	Determines whether elements are of finite size.
<code>isnan</code>	Determines whether elements are NaN.
<code>isletter</code>	Determines whether elements are alphabetical letters (a-z, A-Z).
<code>islogical</code>	Determines whether input is logical array.
<code>isnumeric</code>	Determines whether elements are numeric values.
<code>isreal</code>	Determines whether input is real array.
<code>isstudent</code>	Determines whether MATLAB version is a student Version.
and many others... (see <code>>> doc is*</code>)	



Function is*

- ▶ Try following examples...
 - ▶ Consider in what situation they could prove useful.

```
>> A = 'pi5_7';  
>> B = pi;  
>> C = [Inf NaN 5.31 true false pi];  
>> D = [[] []];  
>> ischar(A), ischar(B),  
>> isstudent, isunix, computer,  
>> isnan(A)  
>> isnan(C)  
>> ischar(A), ischar(B),  
>> isempty(C), isempty(D),  
>> isfinite(A), isfinite(C),  
>> isletter(A),  
>> islogical(C), islogical([true false]),  
>> isnumeric(A), isnumeric(C)
```



Array Sorting – Bubble Sort I.

- ▶ 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 the loops:

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

```
>> sort(R)
```





Array Sorting – Bubble Sort II.

- ▶ 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.





Searching in an Array – Find

- ▶ Function `find` is a very useful one!
- ▶ Returns positions of non-zero (logical `true`) elements of a matrix.
- ▶ Works for a matrix of arbitrary dimensions and size.
- ▶ **Example:** Find the positions of those elements of vector $\mathbf{A} = \left[\frac{\pi}{2} \quad \pi \quad \frac{3}{2}\pi \quad 2\pi \right]$ fulfilling the condition $\mathbf{A} > \pi$:

```
>> A = pi/2*(1:4)
>> find(A > pi)
```

- ▶ Compare the above command with `>> A > pi`. What is the difference?

- ▶ Works for matrices of arbitrary dimensions and sizes.
- ▶ To find the first/last k non-zero elements in \mathbf{X} :

```
>> ind = find(X, k, 'first')
>> ind = find(X, k, 'last')
```

- ▶ For more details see `>> doc find`.



Array Searching I.

- Sort the vector $\mathbf{v} = [16 \ 2 \ 3 \ 13 \ 5 \ 11 \ 10 \ 8 \ 9 \ 7 \ 6 \ 12 \ 4 \ 14 \ 15 \ 1]$ 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.

To get the vector quickly:

```
>> v = reshape(magic(4)', [1 numel(magic(4))])
```

```
v =
    16     2     3    13     5    11    10     8     9     7     6    12     4    14    15     1

v1 =
1x16 logical array
    0     1     0     0     1     0     0     0     0     0     0     0     0     0     0     0

ans =
    15    12

ans =
     2     5
```





Array Searching II.

- ▶ In the matrix w , defined as
 $w = (8:-1:2)'*(1:1/2:4).*magic(7)$
 find the last 3 values that are smaller than 50.
- ▶ Find out the column and row positions of the values.

$$w = \begin{bmatrix} 240 & 468 & 768 & 20 & 240 & 532 & 896 \\ 266 & 493.5 & 98 & 157.5 & 378 & 661 & 812 \\ 276 & 54 & 96 & 255 & 468 & 735 & 888 \\ 25 & 105 & 160 & 312.5 & 510 & 630 & 900 \\ 52 & 90 & 192 & 330 & 504 & 616 & 64 \\ 63 & 103.5 & 192 & 307.5 & 387 & 31.5 & 144 \\ 44 & 93 & 160 & 245 & 12 & 77 & 160 \end{bmatrix}$$



Decoding a Message

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

```
w = 0.6833; t = 1:10; % time
samples = 2.7 + 0.5*(cos(w*t) - sin(w*t) - cos(2*w*t) + sin(2*w*t) ...
    - cos(3*w*t) + 3*sin(3*w*t) + 2*cos(4*w*t) + 4*sin(4*w*t));
plot(samples, '*')
```

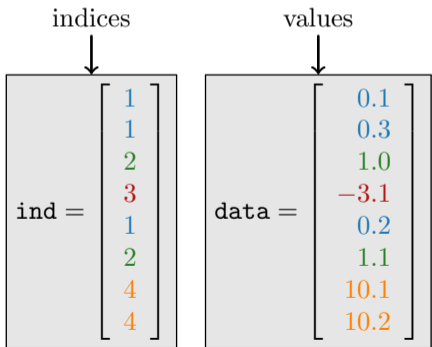
- ▶ Voltage corresponding to characters is within ± 0.5 V tolerance.
- ▶ Decipher the message!

Voltage	Character
1 V	a
2 V	c
3 V	d
4 V	g
5 V	m
6 V	r
7 V	s

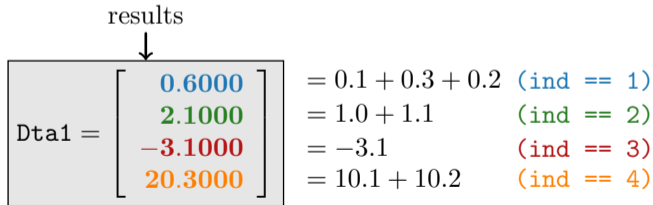


Function accumarray I.

- ▶ 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 data set that is organised in the following way:



```
ind = [1 1 2 3 1 2 4 4].';
data = [.1 .3 1 -3.1 .2 1.1 10.1 10.2].';
Dta1 = accumarray(ind, data)
```





Function `accumarray` II.

- ▶ Basic operation applicable to data from one “box” (data with the same index) is summation.
- ▶ Any other function can be applied, however...

- ▶ **Example:** maximum of a set of elements with the same index:

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

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

- ▶ **Example:** listing of all elements with the same index:

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

```
Dta3 =
 [3x1 double]
 [2x1 double]
 [   -3.1000]
 [2x1 double]
```



Function accumarray III.

- ▶ The function has a wide variety of other features.
- ▶ It is possible, *e.g.*, to use 2D indexing of results.
 - ▶ **Example:** The results are not put in a 1D set of “boxes” but to a 2D array instead:

```
>> ind = [1 1;2 2;1 2;1 3;1 1;3 1];
>> data = [10 22 12 13 1 5.7];
>> Dta4 = accumarray(ind, data)
```

ind =		data =
1	1	10
2	2	22
1	2	12
1	3	13
1	1	1
3	1	5.7

ind == [1 1] 10 + 1 = 11	ind == [1 2] 12	ind == [1 3] 13
ind == [2 1] 0	ind == [2 2] 22	ind == [2 3] 0
ind == [3 1] 5.7	ind == [3 2] 0	ind == [3 3] 0



Function accumarray I.

- ▶ Account transfers in CZK, EUR a USD are denoted as follows:
 - ▶ CZK \sim 1, EUR \sim 2, USD \sim 3.
- ▶ Find out account balance in each currency,
 - ▶ the exchange rate is 26 CZK = 1 €, 25 CZK = 1 \$, find out total balance.

$$dta = \begin{bmatrix} 1 & -110 \\ 1 & -140 \\ 2 & -22 \\ 3 & -2 \\ 2 & -34 \\ 1 & -1300 \\ 2 & -15 \\ 1 & -730 \\ 3 & 24 \end{bmatrix}$$

```
>> dta = [1 -110; 1 -140; 2 -22; 3 -2; ...
          2 -34; 1 -1300; 2 -15; 1 -730; 3 24]
>> K    = [1 26 25]
```


Exercises



Exercise I.a

- ▶ Measurement of temperature was carried out in the course of 5 days every second clock hour. Data were collected at 3 different sites (denoted here as A, B, and 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 I.b

- ▶ 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);
```



Exercise I.c

- ▶ Script for depicting the measured data:

```

%% 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];

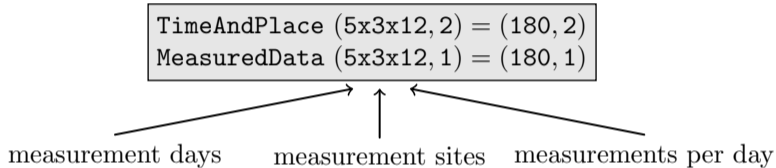
%% plot final data-set
plot(TimeA,PlaceA,'LineWidth',1,'LineStyle','none','Marker','x',...
     'MarkerSize',15); hold on;
plot(TimeB,PlaceB,'LineWidth',1,'LineStyle','none','Marker','*',...
     'MarkerSize',15,'Color','r');
plot(TimeC,PlaceC,'LineWidth',2,'LineStyle','none','Marker','o',...
     'MarkerSize',10,'Color','g');
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');

```



Exercise I.d

- ▶ All the data are contained in 2 matrices:



- ▶ Unfortunately, data in TimeAndPlace are intentionally unsorted.

INDICES	TimeAndPlace =	MeasuredData =	DATA
tindex=10, Place = 1	10 1	15.0797	T(10,1) = 15.0797°C
	4 1	18.9739	
	
tindex=12, Place = 2	12 2	9.9506	T(12,2) = 9.9506°C
	



Exercise I.e

- ▶ Following holds true:
 - ▶ `Place == 1` corresponds to measurement at site A.
 - ▶ `Place == 2` corresponds to measurement at site B.
 - ▶ `Place == 3` corresponds to measurement at site C.
 - ▶ Measurement hour is evaluated as $2*(tindex-1)$.
- ▶ Complete the code to carry out the averaging and plot the data in the existing figure:

```

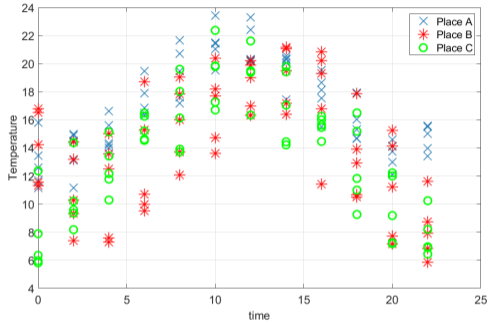
%% PLACE YOUR CODE HERE
%=====
% ...
% dataA = ...
% dataB = ...
% dataC = ...
%=====

%% 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','-');

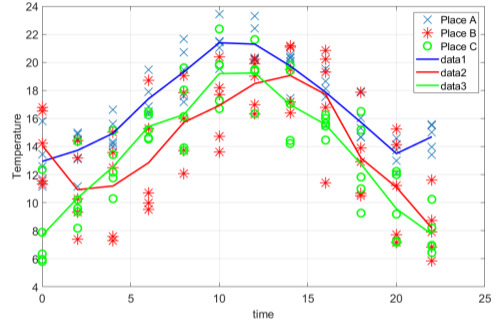
```



Exercise I.f



Measured data.



Measured and averaged data.

Questions?

BE0B17MTB – Matlab
valtrp@fel.cvut.cz

December 5, 2019
Winter semester 2019/20

This document has been created as a part of BE0B17MTB course.
Apart from educational purposes at CTU in Prague, this document may be reproduced, stored, or transmitted only with the prior permission of the authors.

Acknowledgement: Filip Kozak.