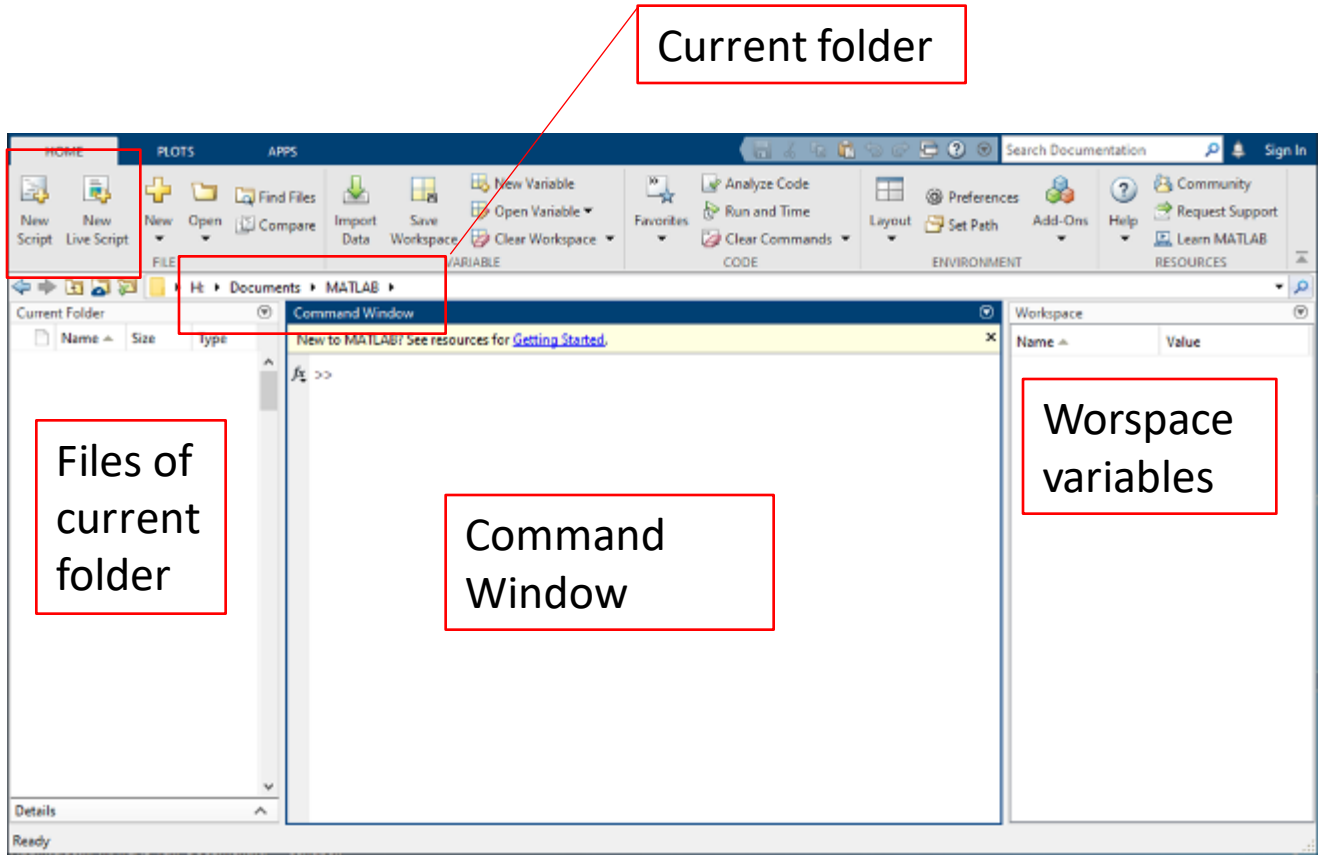
Constraint-based Modeling of Cellular Networks

Exercise 1 – Introduction to MATLAB

20. 10. 2022

Getting started



To run a script, the file must be in the current folder or in a folder on the *search path*. If you want to store and run programs in another folder, add it to the search path.

- addpath(), genpath()

Data structures in MATLAB

- Array
  a = 1
  1x4 double array (**row vector**)
  a = [1 2 3 4], same as b = [5,6,7,8]
  c = [1; 2; 3; 4] (**column vector**)

- Matrix
  a = [1 2 3; 4 5 6] -> 2x3 matrix

- Special functions
  zeros(#rows,#cols), ones(args), nan(args), rand(args), eye()

- Concatenation
  A = [a;b] will concatenate two 1x4 vectors to 2x4 matrix
  B = [a b] will concatenate the two 1x4 vetors to 1x8 vector

Data structures in MATLAB

Tables

- Create table

```
T = table(LastName,Age,Smoker,Height,Weight,BloodPressure)
```

```
T=5×6 table
     LastName      Age    Smoker    Height    Weight    BloodPressu
    ___________    ___    ______    ______    ______    ___________

    {'Sanchez'}    38     true        71        176        124      9
    {'Johnson'}    43     false       69        163        109      7
    {'Li'     }    38     true        64        131        125      8
    {'Diaz'   }    40     false       67        133        117      7
    {'Brown'  }    49     true        64        119        122      8
```

- Access table variables

  e.g. Height

  →T.Height or T{:,4}

- Table properties e.g. list of column and row names

  T.Properties.*field_of_interest*

Data structures in MATLAB

Cell

- Each cell can contain any type of data
- Cell arrays commonly contain either lists of text, combinations of text and numbers, or numeric arrays of different sizes
- Refer to sets of cells by enclosing indices in smooth parentheses, ()
- Access the contents of cells by indexing with curly braces, {}

Struct

- Structure array groups related data using data containers called *fields*
- Each field can contain any type of data
- Access data in a field using dot notation of the form *structName.fielName*

  Creation

  s.a = 1;

  s.b = {‘A‘ , ‘B‘, ‘C‘};

Array and matrix indexing

A = [16 2 3
        4 5 6];

Get element first row second column A(1,2)

→ A(which row, which column)

Using a single subscript to refer to a particular element in an array is called *linear indexing*
e.g. A(4), vectorize matrix Av=A(:) results in Av=[16 4 2 5 3 6]

Refer to multiple elements using colon operator to specify range of the form *start:end*

e.g. A(1:2,2), A(:,1), A(1,end)

Equally spaced vectors you get by the general form *start:step:end*

B = 0:10:100

Matrix and Array Operations

a=[1 2 3 4] (same for matrix)

- a+10; a-10; a*10; a/10
  -> [11 12 13 14]; [-9 -8 -7 -6]; [10 20 30 40]; [1/10 2/10 3/10 4/10]

- Transpose a‘

- Element-wise multiplication vs. matrix multiplication
  .* operator elementwise multiplication
  * Matrix multiplication
  Same for division (./ or /) or power (.^ or ^)

Functions

What is a function?

- Functions contain code that accomplish a specific task
- Functions usually "take in" data, process it, and "return" a result.
- Functions can be used over and over again

Functions

What is a function?

- Functions contain code that accomplish a specific task
- Functions usually "take in" data, process it, and "return" a result.
- Functions can be used over and over again

e.g. function size() returns the number of rows and columns in a matrix

Output of function saved in variable S

S = **size**(input_matrix)

Input matrix saved in variable *input_matrix*

How to find input and output of a function

Documentation called with help *function_name*

```
>> help size
 size   Size of array.
    D = size(X), for M-by-N matrix X, returns the two-element row vector
    D = [M,N] containing the number of rows and columns in the matrix.
    For N-D arrays, size(X) returns a 1-by-N vector of dimension lengths.
    Trailing singleton dimensions are ignored.

    [M,N] = size(X) for matrix X, returns the number of rows and columns in
    X as separate output variables.

    [M1,M2,M3,...,MN] = size(X) for N>1 returns the sizes of the first N
    dimensions of the array X.  If the number of output arguments N does
    not equal NDIMS(X), then for:

    N > NDIMS(X), size returns ones in the "extra" variables, i.e., outputs
                  NDIMS(X)+1 through N.
    N < NDIMS(X), MN contains the product of the sizes of dimensions N
                  through NDIMS(X).

    M = size(X,DIM) returns the lengths of the specified dimensions in a
    row vector. DIM can be a scalar or vector of dimensions. For example,
    size(X,1) returns the number of rows of X and size(X,[1 2]) returns a
    row vector containing the number of rows and columns.

    M = size(X,DIM1,DIM2,...,DIMN) returns the lengths of the dimensions
    DIM1,...,DIMN as a row vector.

    [M1,M2,...,MN] = size(X,DIM) OR [M1,M2,...,MN] = size(X,DIM1,...,DIMN)
    returns the lengths of the specified dimensions as separate outputs.
    The number of outputs must equal the number of dimensions provided.

    When size is applied to a Java array, the number of rows
    returned is the length of the Java array and the number of columns
    is always 1.  When size is applied to a Java array of arrays, the
    result describes only the top level array in the array of arrays.

    Example:
    If
       X = rand(2,3,4);
    then
       d = size(X)              returns  d = [2 3 4]
       [m1,m2,m3,m4] = size(X)  returns  m1 = 2, m2 = 3, m3 = 4, m4 = 1
       [m,n] = size(X)          returns  m = 2, n = 12
       m2 = size(X,2)           returns  m2 = 3

    See also length, ndims, numel.
```

size(), length(), numel()

M = [1 2 3 4;
5 6 7 8]

→size(M) will return

ans =

2 4

→length(M) will return

ans =

4

→numel(M) will return

ans =

8

[rows,cols]=size(M);

Load and save data

- save and load matlab workspace

  save myfile.mat same as save(‘myfile.mat‘) allows to specify which variable to save save(‘myfile.mat‘, ‘B‘)

  load myfile.mat, load(‘myfile.mat‘), load(‘myfile.mat‘, ‘B‘)

Other functions:

readtable(), writetable(), importdata(), ...

Clear the workspace using clear

Loops and conditional assignment

for loop **repeat specified number of times**	**if, elseif, else** **Execute statements if condition is true**	**while loop** **repeat when condition is true**

for *index* = *values*
    statements
end
	if *expression*
	    statements
	elseif *expression*
	    statements
	else
	    statements
	end
		while *expression*
		    statements
		end

Create a matrix of 1s.

Try This Example

Copy Command

```
nrows = 4;
ncols = 6;
A = ones(nrows,ncols);
```

Loop through the matrix and assign each element a new value. Assign `2` on the main diagonal, `-1` on the adjacent diagonals, and `0` everywhere else.

```
for c = 1:ncols
    for r = 1:nrows

        if r == c
            A(r,c) = 2;
        elseif abs(r-c) == 1
            A(r,c) = -1;
        else
            A(r,c) = 0;
        end

    end
end
A
```

```
A = 4×6

     2    -1     0     0     0     0
    -1     2    -1     0     0     0
     0    -1     2    -1     0     0
     0     0    -1     2    -1     0
```

https://de.mathworks.com/help/matlab/ref/if.html

Example 1

For loop and
if, elseif, else conditional assignment

Functions I used to solve given homework

- readtable(), table2array(), array2table(), cell2table()
- find(), strcmp(), contains(), intersect()
- mean(), sum(), log(), repmat()

  Note: mean(*data*, 1) – mean for each column

  mean(*data*, 2) – mean for each row

>> figure (creates new figure panel)

- plot(), histogram(),

  And related: ylabel(), xlabel(), legend(), xlim(), ylim(), ...

  Manipulate figure using set()

  e.g. set(gca,‘FontSize‘,14,‘XTick,1:3, 'XTickLabel',{‘A‘, ‘B‘, ‘C‘}, 'XTickLabelRotation',45)

Most Important function: help

If that was not enough …

Check MathWorks webpage for tutorials on how to get started

https://de.mathworks.com/help/matlab/getting-started-with-matlab.html?searchHighlight=getting%20started&s_tid=srchtitle_getting%2520started_1

Use a functions documentation

>> **help** *function_name*

Ask google!