Rehearsal Lab

Contents

1 Vectors warm-up

1.1 Row vectors

```
driverSalary = [1000, 2000, 3000, 4000];

driverSalary
driverSalary(1)
driverSalary(2)
driverSalary(end)
driverSalary(1:3)
driverSalary(:)

driverSalary(2) = driverSalary(2)-200;
driverSalary

```

```
length(driverSalary)
size(driverSalary)
```

1.2 Column vectors

Note semicolon instead of comma.

```
driverSalary = [1000; 2000; 3000; 4000];

driverSalary
driverSalary(1)
driverSalary(end)
driverSalary(:)

length(driverSalary)
size(driverSalary)
```

1.3 Operations with vectors

If one operand is a scalar and the other is not, MATLAB applies the scalar to every element of the other operand — this property is known as scalar expansion.

```
% change all elements of a vector (scalar expansion)
driverSalary = driverSalary + 1000
driverSalary = driverSalary - 1000
driverSalary = driverSalary * 1000
driverSalary = driverSalary / 1000

array1 = [10, 20, 30];
array2 = [30, 20, 10];

% element-wise operations
array1 + array2
array1 - array2
array1 .* array2
array1 ./ array2

% adding an element
array1(4) = 40
```

```

% removing an element
array1(2) = []

% vectors concatenation
[array1, array2]

% transpose
array1'
array2'

% another concatenation
[array1'; array2']
```

1.4 Other ways to create vectors

```
1:1:10
% or
1:10

1:2:10

-1:-1:-10

1:-1:10 % empty vector -- we cannot create a vector
        % from 1 to 10 with step -1

linspace(1, 10, 5)

zeros(1, 10)
ones(1, 10)
rand(1, 10)
```

2 Visualising graphs warm-up

2.1 Plotting a single graph

You can plot a graph using `graph` or `digraph` and `plot` functions. Functions `graph` and `digraph` take adjacency matrix as an input. Command `figure` creates a new window with a figure to plot in it. Function `plot` draws a visual representation of the graph in the last opened figure window. Command `close all` closes all open figure windows and is often used in the beginning of a script along with `clear` command. Try the following example:

```
clear; close all;

A = [
    0 1 0 1;
    1 0 1 0;
    0 1 0 1;
    1 0 1 0
    ];

figure;
G = graph(A);
plot(G);
```

Or with a directed graph:

```
clear; close all;

A = [
    0 1 0 1;
    0 0 1 0;
    0 1 0 0;
    0 1 1 0
    ];

figure;
G = digraph(A);
plot(G);
```

2.2 Plotting several graphs

If more than one `plot` command is called for one figure window, by default Matlab will display only the result of the last `plot` call. `figure` can be used to create more than one figure window.

For example:

```
clear; close all;
```

```

A = [
    0 1 0 1;
    0 0 1 0;
    0 1 0 0;
    0 1 1 0
    ];

figure;
G = digraph(A);
plot(G);


figure; % create a second window for the second graph
G2 = digraph(A'); % digraph of the transposed adjacency matrix
plot(G2);
```

Alternatively, a `pause` command can be used to plot graphs in the same window one by one. The `pause` command pauses the execution of the program until the user continues it by pressing Enter in the command window. For example:

```
clear; close all;

A = [
    0 1 0 1;
    0 0 1 0;
    0 1 0 0;
    0 1 1 0
    ];

figure;
G = digraph(A);
plot(G);

pause; % pause after plotting the first graph

G2 = digraph(A'); % digraph of the transposed adjacency matrix
plot(G2);
```

2.3 Storing edge lists

To store an edge list you can use an $n \times 2$ matrix (or $2 \times n$), where each row (or column) consists of two numbers: u and v of the $e = (u, v)$ edge. For example, the following code shows how you can store edge list [(1, 2), (2, 3), (2, 1), (3, 4)] in an $n \times 2$ matrix.

```
edge_list = [1, 2; 2, 3; 2, 1; 3, 4];
```

2.4 Adding new edges

In order to add a new edge, we need to use concatenation. If we want to add a new edge (1, 4) to the edge list from the previous example, we can use the following code:

```
new_edge = [1, 4];
edge_list = [edge_list; new_edge];
```

2.5 Plotting a graph using edge list

```

edge_weights = [1 1 2 2 3];
G = graph(edge_list(:,1), edge_list(:,2), edge_weights);
figure;
plot (G);
```

Task 1

- Create a vector with 10 random numbers from 0 to 100.
- Display the value of the 2nd, the 4th and the 7th elements of the vector.
- Display the elements of the vector on positions 6 to 10.
- Decrease the value of the 2nd element of the vector by 30.
- Increase values of all elements of the vector by 50.

- Remove the 3rd element of the vector.
- Add an element with value '10' to the end of the vector.

Task 2

Create an adjacency matrix for an **undirected** graph with 6 vertices. Plot a graph using this adjacency matrix. Create an edge list for the same graph. Plot a graph using this edge list. Check that it is the same graph.

Task 3

Modify the script from task 4.1 so that it works with a **directed** graph.

Task 4

- Write a program which creates an edge list representation of an adjacency matrix of a directed graph. Test the program on several graphs from Homework 1 of the theoretical part of the course. Plot the graphs.
- Write a program which given an edge list converts it to an adjacency matrix. Test the program on the resulting edge lists from part 1 (you can write the code in the same program as in part 1 and simply use the resulting edge lists from there or start a new script) and check that the resulting adjacency matrix is the same as the original adjacency matrices used in part 1. Plot the graphs.

Hint: if you want to create a new matrix of the same size as an existing one and fill it with zeros, you can use the following code, where `A` is the existing matrix.

```
new_A = zeros(size(A));
```

Hint: in order to plot several graphs use `figure` or `pause`