Lab #5

Contents

1 Matrices

Matrices are two-dimensional arrays. Each value in a matrix is identified by a pair of numbers: row and column.

1.1 Working with matrices

Matlab syntax for an element at row `r` and column `c` of the matrix `M` is `M(r, c)`.

Examples:

```
a = [10, 20.11; 3.18, pi];

a
a(1, 1) % element at the first row and first column
a(2, 1) % element at the second row and first column
a(end, 1) % element at the last row and first column
a(1:2, 1) % first and second rows of the matrix a and
    the first column
a(1, :) % first row of the matrix a
a(:, 2) % second column of the matrix a

a(2, 2) = -2; % changing value at the second row and
   second column
                % to -2
a(2, :) = a(2, :) + 3; % add 3 to all elements in the
    second row
a(:, 1) = a(:, 1) - 1; % add -1 to all elements in
   the first column
a

size(a) % returns size of the matrix
size(a, 1) % returns the number of rows in the matrix
size(a, 2) % returns the number of columns in the
   matrix
```

1.2 Matrix operations

Very similar to operations with vectors (see Lab 2).

```
% change all elements of the matrix a
a = a + 10
a = a - 10
a = a * 10
a = a / 10

matrix1 = [10, 20; 30, 40];
matrix2 = [30, 20; 10, 66];

% element-wise operations
```

```
matrix1 + matrix2
matrix1 - matrix2
matrix1 .* matrix2
matrix1 ./ matrix2

% matrix concatenation
[matrix1, matrix2]
[matrix1; matrix2]

% adding a column
matrix = [1, 2; 1, 2];
column = [3; 3];
matrix = [matrix, column];
% or
matrix = [matrix column];

% adding a row
matrix = [1, 1; 2, 2];
row = [3, 3];
matrix = [matrix; row];

% removing a row
matrix = [1, 1, 1; 2, 2, 2; 3, 3, 3];
matrix(3, :) = [];

% removing a column
matrix = [1, 2, 3; 1, 2, 3; 1, 2, 3];
matrix(:, 3) = [];

% transpose
a'
a''
```

1.3 Matrix creation

```
zeros(5, 10)
ones(6, 10)
rand(7, 10)
```

1.4 Iterating through all elements of the matrix

In order to work with elements of the matrix one-by-one, we need to use nested loops.

1.4.1 Print each element separately

In the following example we print each element separately:

```
clear;

M = rand(4, 4);

number_of_rows = size(M, 1);
number_of_columns = size(M, 2);

% for each row
for row = 1:number_of_rows
    % for each column
    for column = 1:number_of_columns
        disp(M(row, column)); % print value
    end
end
```

1.4.2 Adding 2 to each element

In the following example we add 2 to each element:

```
clear;

M = zeros(4, 4);

number_of_rows = size(M, 1);
number_of_columns = size(M, 2);

% for each row
for row = 1:number_of_rows
    % for each column
    for column = 1:number_of_columns
```

```
12          M(row, column) = M(row, column) + 2; % add 2
13      end
14  end
```

1.4.3 Finding the sum of all elements

In the following example we are calculating the sum of all elements in the matrix:

```
1  clear;
2
3  M = [1, 2, 1; 4, 5, 2; 1, 3, 2];
4  sum_of_elements = 0;
5
6  number_of_rows = size(M, 1);
7  number_of_columns = size(M, 2);
8
9  % for each row
10 for row = 1:number_of_rows
11     % for each column
12     for column = 1:number_of_columns
13         sum_of_elements = sum_of_elements + M(row,
               column);
14     end
15 end
16
17 disp(sum_of_elements);
```

1.4.4 Finding max element

In the following example we are looking for the largest element:

```
1  clear;
2
3  M = [1, 2, 1; 4, 5, 2; 1, 3, 2];
4  maximum = M(1, 1);
5
6  number_of_rows = size(M, 1);
```

```
number_of_columns = size(M, 2);

% for each row
for row = 1:number_of_rows
    % for each column
    for column = 1:number_of_columns
        if M(row, column) > maximum
            maximum = M(row, column);
        end
    end
end

disp(maximum);
```

1.4.5 Print matrix line by line

In the following example we are printing the matrix one row at a time:

```
clear;

M = rand(4, 4) ;
number_of_rows = size(M, 1) ;

% for each row
for row = 1:number_of_rows
    disp(M(row, :)) ; % print value
end
```

2 Plotting graphs

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);
```

2.1 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);
```

3 Task #9

Create in Matlab adjacency matrices of any two different graphs of different size (you can use graphs from the lecture homework, any random graph, etc.). Plot these graphs.

4 Task #10

In a script:

- Create an adjacency matrix `A` of the graph from the Figure 1 of Homework 1 from the theoretical part of the course.
- Create a new adjacency matrix `B = A;` and write code which modifies `B` by adding an edge from vertex 2 to vertex 1 using matrix manipulations (do not write the matrix again by hands, make sure that the code will work the same way even if you use a different `A`).
- Create a new adjacency matrix `C = A;` and write code which modifies `C` by adding edges from vertices 2-6 to vertex 1 using a for loop (do not write the matrix again by hands, make sure that the code will work the same way even if you use a different `A`).
- Create a new adjacency matrix `D = A;` and write code which modifies `D` by removing all incoming edges to vertex 6 using a for loop (do not write the matrix again by hands, make sure that the code will work the same way even if you use a different `A`).
- Plot these graphs.

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

5 Task #11

Write code which finds the product of all negative numbers in a matrix.

For example, for a matrix `M = [1, -10, 20; 3, -5, -3];` the result should be -150, and for a matrix `M2 = [-1, -2, -3, -4; 9, 8, 7, 6; -5, 1, 6, 4; 1, 6, 3, -5];` the result should be 600.