Rehearsal Lab

Contents

1 Logical indexing warm-up

In logical indexing, you use a logical array as a mask for the vector subscript. MATLAB extracts the vector elements corresponding to the nonzero values of the mask (logical array).

```
v = [1 2 3 4 5]

mask = logical([1 1 0 0 1])
% or mask = [true true false false true]

v(mask) % [1 2 5]
```

Using `find` function you can get the indices of the elements which satisfy the mask.

```
v = [1 2 3 1 2 3]

mask =  [false false true false false true]

equal_to_three = v(mask) % [3 3]

equal_to_three_index = find(mask) % [3 6]

equal_to_three_index = find(mask, 1) % 3
```

You can filter the elements of an array by applying conditions to the array. Applying conditions to the array returns the logical array. You can use this logical array as a mask to filter out not needed elements.

```
a = 1:2:20

mask = a<10 & a~=3

less_than_ten_and_not_three = a(mask)

less_than_ten_and_not_three_indices = find(mask)
```

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 Plotting a graph using edge list

```
edge_list = [1, 2; 2, 3; 2, 1; 3, 4; 1, 4];
G = graph(edge_list(:,1), edge_list(:,2));
figure;
plot (G);
```

2.5 Plotting a graph using edge list and edge weights

```
edge_list = [1, 2; 2, 3; 2, 1; 3, 4; 1, 4];
edge_weights = [1 1 2 2 3];
G = graph(edge_list(:,1), edge_list(:,2),
   edge_weights);
figure;
plot (G, 'EdgeLabel', G.Edges.Weight);
```

2.6 Plotting a graph using edge list and node labels

```
edge_list = [1, 2; 2, 3; 2, 1; 3, 4];
G = graph(edge_list(:,1), edge_list(:,2));
G.Nodes.Name = {'First' 'Second' 'Third' 'Fourth'}';
figure;
plot (G);
```

2.7 Adding edges to a graph

```
edge_list = [1, 2; 2, 3; 2, 1; 3, 4];
G = graph(edge_list(:,1), edge_list(:,2));
% To add edge (1,3) with weight 4:
G = addedge(G,1,3,4)
```

2.8 Adding nodes to a graph

```
edge_list = [1, 2; 2, 3; 2, 1; 3, 4];
G = graph(edge_list(:,1), edge_list(:,2));
% To add node 5:
G = addnode(G,5)
```

2.9 Removing edges from a graph

```
edge_list = [1, 2; 2, 3; 2, 1; 3, 4];
G = graph(edge_list(:,1), edge_list(:,2));
G.Nodes.Name = {'First' 'Second' 'Third' 'Fourth'}';
% To remove edge (2,1):
G = rmedge(G,2,1)
% or
G = rmedge(G,3)
% or
G = rmedge(G,'Second','First')
```

2.10 Removing nodes from a graph

```
edge_list = [1, 2; 2, 3; 2, 1; 3, 4];
G = graph(edge_list(:,1), edge_list(:,2));
G.Nodes.Name = {'First' 'Second' 'Third' 'Fourth'}';
% To remove node 1:
G = rmnode(G,1)
% or
G = rmnode(G,'First')
```

Task 1

Create a vector with the following elements: 1, 4, -2, 7, -4, 5, 10, 15. Using logical indexing and *find* function do the following:

- Display all positive elements of the vector
- Display the indices of all positive elements of the vector
- Display all elements of the vector except the third element
- Display all even elements of the vector
- Display the indices of all even elements of the vector
- Display all elements of the vector with even indices

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 2 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`