Lab 2

Contents

1 Vectors

1.1 Array

An array, is a data structure consisting of a collection of elements (values or variables), each identified by at least one array index or key.

1.2 Vector

Vector is one-dimensional array.

1.3 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)
```

1.4 Column vectors

Note semicolon instead of comma.

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

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

length(driverSalary)
size(driverSalary)
```

1.5 Operations with vectors

```
% change all elements of a vector
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

% vectors concatenation
[array1, array2]

% transpose
array1'
array1''

% concatenation again
[array1; array2]
```

1.6 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)

% and many other
```

1.7 Sum of all elements in a vector

```
clear;

vector = [1, 20, -3, 5, 6];
vector_length = length(vector);

sum = 0;

for i = 1:vector_length
    element = vector(i);
    sum = sum + element;
end
```

2 Matrices

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

2.1 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
length(a) % returns the maximum of the number of columns and the number of rows
numel(a) % returns the number of elements in the matrix
```

2.2 Matrix operations

Very similar to operations with vectors.

```
% 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 operations
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''
```

2.3 Matrix creation

```
zeros(5, 10)
```

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

2.4 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.

2.4.1 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
```

2.4.2 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
        M(row, column) = M(row, column) + 2; % add 2
    end
end
```

2.4.3 Finding the sum of all elements

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

```
clear;

M = [1, 2, 1; 4, 5, 2; 1, 3, 2];
sum_of_elements = 0;

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
        sum_of_elements = sum_of_elements + M(row,
           column);
    end
end

disp(sum_of_elements);
```

2.4.4 Finding max element

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

```
clear;

M = [1, 2, 1; 4, 5, 2; 1, 3, 2];
maximum = M(1, 1);

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);
```

2.4.5 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
```

3 Logical arrays

Logical arrays consist of logical true/false values.

```
a = [true true false]

b = [true false; false true]

c = logical ([1 1 0])

% any nonzero value is logical true
```

```
d = logical ([2 -3 0])

% row vector with 3 elements
e = true(1,3)

% matrix 3x3
f = false(3)
```

Logical function `any(v)` returns true if at least one element of vector v is true.

Logical function `all(v)` returns true if all elements of vector v are true.

```
a = [true true false]

any(a)
all(a)

b = [true true true]
all(b)
```

You can filter the elements of an array by applying conditions to the array. Applying conditions to the array returns the logical array. Applying logical array to the array filters out not needed elements. Using `find` function you can get the indices of the elements which satisfy the conditions.

```
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)
```

4 Vectorization

Matlab is optimized for operations involving matrices and vectors. The process of revising loop-based code to use Matlab matrix and vector operations is called vectorization.

Vectorized code is often shorter and runs much faster than the corresponding code containing loops.

```
% This code computes the sine of 11 values ranging
    from 0 to 1
i = 0;
for t = 0:.1:1
    i = i + 1;
    y(i) = sin(t);
end

% This is a vectorized version of the same code
t = 0:.1:1;
y = sin(t);
```

Vectorization is also performed using logical arrays.

```
a = [2 -4 5 9 -6 4 -5];

% Calculate the sum of all positive elements of the
    array
s = 0;
for i=1:length(a)
    if a(i)>0
        s = s + a(i);
    end
end

disp(s)

% Vectorized version of the same code
s2 = sum(a(a>0));
disp(s)
```

5 Homework 2

Task 1

- Create a vector with 10 random numbers from 0 to 100.
- Display the value of the 4th element of the vector.
- Decrease the value of the 2nd element of the vector by 30.

- Increase values of all elements of the vector by 50.
- Display the elements of the vector on positions 6 to 10.

Task 2

Write a script which computes the product of all positive elements of the vector `v` using `for` loop. Define vector `v` in the code so, that it contains both positive and negative elements.

For example, for a vector `v = [2, -3, -5, 6]` you should get 12.

Task 3

Write code which finds the sum of all positive numbers in a matrix using `for` loop. Define matrix `M` in the code so, that it contains both positive and negative elements.

For example, for a matrix `M = [1, -10, 20; 3, -5, -3];` the result should be 24.

Task 4

Write a script which asks the dimension of the matrix from the user, fills it with random numbers and prints the maximum elements of each row (without using Matlab built-in function `max`). Print the matrix itself also.

Task 5

Vectorize the following code (re-write the code without for loops):

```
tic
i = 0;
for inc = 0: 0.5: 3
    i = i + 1;
    my_vector(i) = sqrt(inc);
end

disp(my_vector)

M = [ 1 5 -7; -4 0 2; 0 -9 5]

[num_of_rows,  num_of_columns] = size(M);
new_M = zeros( num_of_rows,  num_of_columns );
```

```
for i = 1:num_of_rows
    for j = 1:num_of_columns
        new_M(i,j) = sign(M(i,j));
    end
end

disp(new_M)
toc
```

Functions `tic` and `toc` measure the performance of the code. Use them to compare the performance of your code with the performance of the code with for loop.

Note: vectorization is not always possible! Sometimes we have to use loops, for instance in case of more complicated calculations for matrix elements which could not be done with Matlab build-in vectorized functions.