List of practice tasks (LabVIEW+MATLAB)
1 Noise generator + Averaging by 3 points
2 Time interval measurements + SubVIs
3 Random function qualification
4 Select, case, formula node + Picture
5 Perseptron
6 SOM

Task 6. SOM.

The task is creation and implementation of ***S**elf-**O**rganized Kohonen **M**ap **(SOM)***.

SOM is modification of layer of competitive neurons (Kohonen network). SOM differs in following: (1) neurons are distributed by spatial way (in nodes of rectangle grid randomly; (2) at self-learning stage, weights are tuning for not only winner neuron but group of nearest neurons in some field around it. Aim of SOM is the same as Kohonen layer – defining in learning regime by centers of clusters for input vectors.

Let's consider an example of clustering for two-dimension vectors given randomly.

```
P = rands(2,1000);

% Visual mapping input vectors
plot(P(1,:),P(2,:),'+r')

% Creation of network with 5*6 = 30 neurons; all settings are automatic
net = newsom([0 1; 0 1],[5,6]);

% Setting number of tuning cycles
net.trainParam.epochs = 1000;

% Setting report repetition rate
net.trainParam.show = 200;

% Training of the network
net = train(net,P);
```

Response in the command line:

```
TRAINWB1, Epoch 0/1000
TRAINWB1, Epoch 200/1000
TRAINWB1, Epoch 400/1000
TRAINWB1, Epoch 600/1000
TRAINWB1, Epoch 800/1000
TRAINWB1, Epoch 1000/1000
TRAINWB1, Maximum epoch reached.
```

Let's apply trained network to a sample vector

```
% Plotting centers of clusters
plotsom(net.iw{1,1},net.layer{1}.distances);
p = [0.5; 0.3];
% Simulation for vector p
a = sim(net,p)
```

Result is:
a =
(11,1) 1

Testing vector was related to 11-th class (see figure).

Task 5. Vector classification using perseptron.

Create and debug a virtual instrument in LabVIEW which will classify a vector to one of two classes, using MATLAB script and Neural network toolbox in MATLAB.

There are two samples for every class.

Class 0: (0.3 -0.5), (-0.1 1.0)

Class 1: (-0.5 -0.5), (-0.5 0.5)

Sample of code in MATLAB.

Let's use neural network "***perceptron***" which will be called as **My_net.** Program for perceptron modeling looks as following:

```
% Setting input vectors and their classes
P = [-0.5 -0.5 +0.3 -0.1; -0.5 +0.5 -0.5 +1.0];
T = [1 1 0 0];

% Graphical representation of given vectors
plotpv(P,T);

% Creation of perceptron with limits of changing inputs and single neuron
My_net = newp([-1 -1; 1 1],1);
E = 1;

% Perceptron initialization
My_net = init(My_net);

% Organization of cycle for adaptive tuning perceptron with showing
% graphic of divided line
while (sse(E)), [My_net,Y,E] = adapt(My_net,P,T);
  linehandle = plotpc(My_net.iw{1},My_net.b{1});
  drawnow;
end;

% Prepare data for export to LabVIEW
y(1) = My_net.iw{1}(1);
y(2) = My_net.iw{1}(2);
y(3) = My_net.b{1}(1);
```

Then implement this script in MATLAB node in LabVIEW.

Output vector **y** of Matlab script includes coefficient of separating line according to
$z(x) = -(a1 / a2)\, x + b$, where
a1 = y(1)
a2 = y(2)
b = y(3)

Task 4. Branching operators.
Use Select function, Case structure and Formula node as IF and/or CASE operators)
Create a VI for square root calculation with preliminary check for negative input.
Firstly do this using “select” function, secondly use “case” structure, and at least - use “formula node”.

Task 4a Create VI using Select function as IF operator.

Task 4b Copy and modify VI 5.1 replacing Select function with Case structure.

Task 4c. Copy and modify VI 5.2 replacing Case structure with Formula node.

Task 4d. Graphics.
Create VI which draws ice lane for curling with animation of random moving of a stone.

Task 3.
Create a random function qualification VI on the base of the time interval measurement VI.

1. Start time interval measurement for 0 target several times and calculate average value.
2. Repeat step 1 for every number from 1 to 1000000.
3. Draw waveform graph of average time dependence on target number.

Task2 Random signal and average signal by 3 points

1. Take VI s1random.vi (task1).
2. Create 3 copies of Random number (0-1) function (Functions -> Numeric -> Random number (0-1) outside the While loop.
3. Using context menu of While loop add shift register.
4. Expand source terminal of the shift register up to 3 outputs (on the left side of the cycle frame)
5. Initialize the shift register using wiring 3 Random number (0-1) functions outside the cycle frame to 3 source terminals of the shift register.
6. Delete wire Random number (0-1) – Waveform chart using Position tool
7. Wiring: Random number (0-1) – Destination terminal of the shift register (on the right side of the cycle frame)
8. Join 3 random values into array of 3 random values (vector of 3 components). Functions -> Programming -> Array -> Build array
9. Wiring: Source terminal of the Shift register – input of the Build array (3 wires)
10. Functions -> Programming -> Numeric -> Add array element
11. Wiring: Build array – Add array element (thick continuous line indicate array data type)
12. Functions -> Programming -> Numeric -> Divide
13. Functions -> Programming -> Numeric -> Constant 3
14. Wiring: Add array element – Divide top, Constant 3 – divide bottom.
15. Functions -> Programming -> Cluster & Variant -> Bundle (join to scalar data flow – raw signal and average by 3 points – into cluster data flow required by waveform chart)
16. Wiring: top source terminal of the Shift register – Bundle (top), Divide – Bundle (bottom), Bundle – Waveform chart.
17. Front panel -> Expand Plot legend of the Waveform chart up to 2 rays (“Raw” – white, “Average” – red)
18. Front panel -> Context menu of “Average” ray of the Plot legend of the Waveform chart -> Fill base line -> Zero

19. Save results as s2average.vi

Task1 Random signal.

1. LabVIEW start menu -> Blank VI
2. Functions -> Programming -> Structures -> While loop
3. Functions -> Programming -> Numeric -> Random number (0-1)
4. Controls -> Graph -> Waveform chart
5. Tools -> Connect wire
6. Wiring: Random number - Waveform chart
7. Tools -> Position
8. Controls -> Boolean -> Stop button
9. Wiring: Stop button – Stop condition terminal of While loop
10. Functions -> Programming -> Timing -> Wait until next ms multiple
11. Controls -> Numeric -> Horizontal pointer slide
12. Context menu of Horizontal pointer slide “Period” -> Mapping -> Logarithmic
13. Functions -> Programming -> Numeric -> Multiply
14. Functions -> Programming -> Numeric -> Constant 1000
15. Wiring: “Period” – Multiply, 1000 – Multiply, Multiply – Wait until next ms multiple