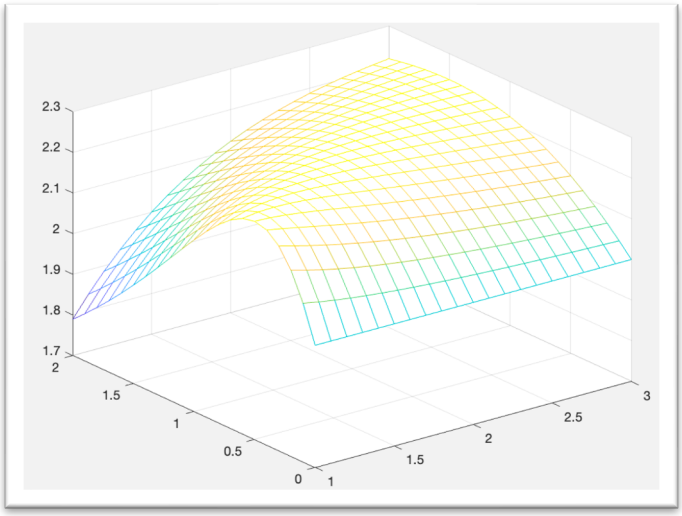
Report

Isil Sonmez

We can see our neuron formula `w'*x`

```
w = [2;1];
x1 = 1:0.1:3;
x2 = 0:0.1:2;

for i = 1:length(x1)
    for j = 1:length(x2)
        %x = [x1(i); x2(j)];
        x=normc([x1(i),;x2(j)]);
        y(j,i) = w'*x;
    end
end
mesh(x1,x2,y)
```

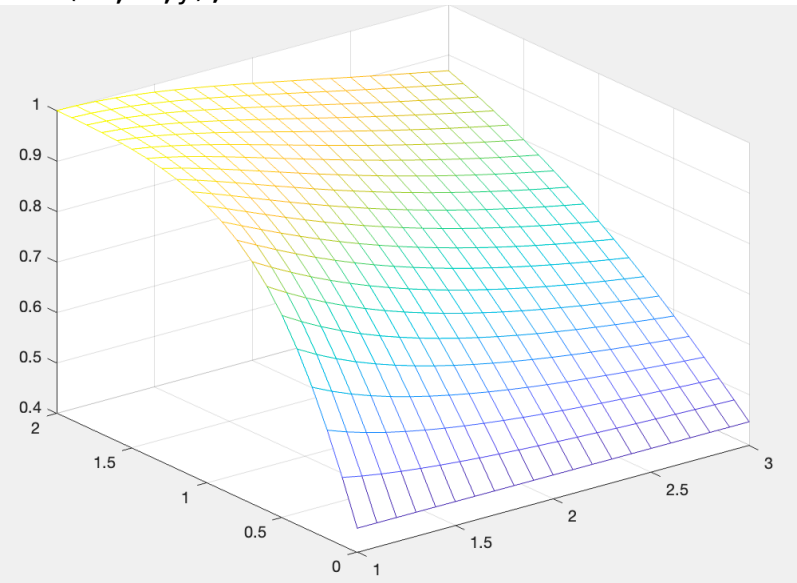


In this code, the mesh plot gives us a visual representation where each spot on the surface represents what the linear function gives back for a certain pair of x1 and x2 values. Because we are keeping w constant and normalized, the shape of this surface mostly depends on how x1 and x2 change.

Our function is linear the surface of the mesh plot will also be linear. If we adjust x1 and x2, we will see the output on the plot change accordingly, but it will still follow a straight line pattern overall.

We can see we used normc function here.

```
w=[1;2];
x1=[1:.1:3];
x2=[0:0.1:2];
for i=1:length(x1)
    for j=1:length(x2)  wn=normc(w)
            x=normc([x1(i);x2(j)])
        y(j,i)=wn'*x;
    end
end
mesh(x1,x2,y);
```

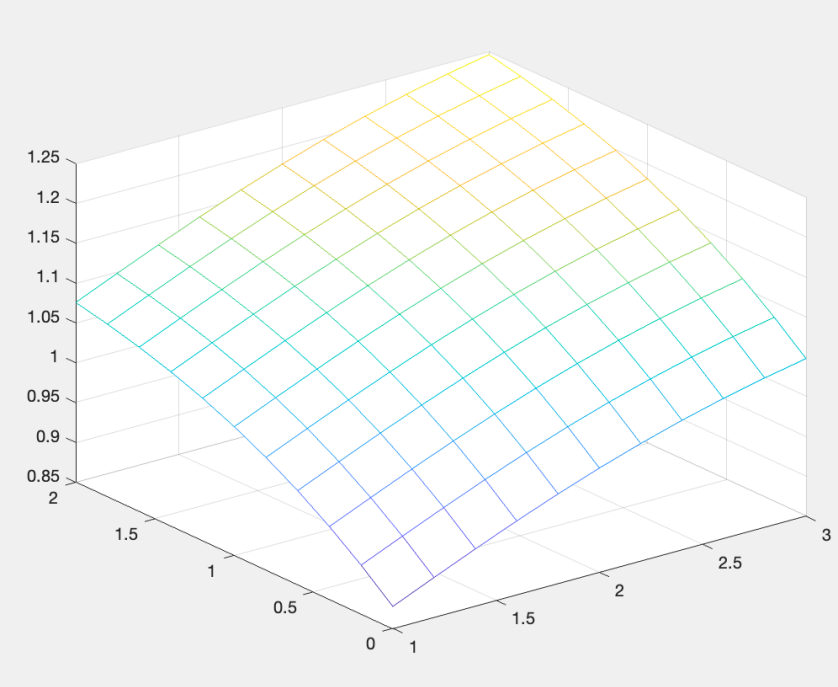


We are using normalization because of data ensures that the scale of feature is consistent.It makes our algorithm convergence faster.

In this code, when we normalize the vectors w and xit makes sure that we are measuring things in good way. This means that no matter how big or small the numbers are in our vectors, we treat them equally.

We used logsig function in this code.

```
W=[2;1];
w=normc(W)
x1=[1:.2:3];
x2=[0:.2:2];

for i=1:length(x1)
 for j=1:length(x2)
 x=[x1(i);x2(j)];
 x=logsig(x);
 y(i,j)=w'*x;

 end
end

mesh(x1,x2,y)
```



In this code, logsig function changes the numbers in the input before multiplying them with other numbers. It's like putting the input through a special machine that adjusts its values in a particular way. This is different from the first code, where the focus was just on making sure all the numbers were treated equally before doing any calculations. So, while the first code was about making sure everything was fair, the second one also changes the numbers themselves before doing the calculations.

We can choose our animals in the categories. Normc function able to assign animals in correct categories.

Lab2

```
fox=[4;-1;-1;-0.9;-1];
salmon=[-1;4;-1;-2;-1];
chicken=[2;-1;0.01;3;3];
cat=[4;0;0;-0.5;-2];
w=[4 2 1; 0.01 -1 3.5; 0.01 2 0.01;-1 2.5 -2; -1.5 2 1.5];
x=[fox,salmon,chicken,cat];
y=w'*x;y
x=normc(x)
w=normc(w)
y = compet(w'*x)
```

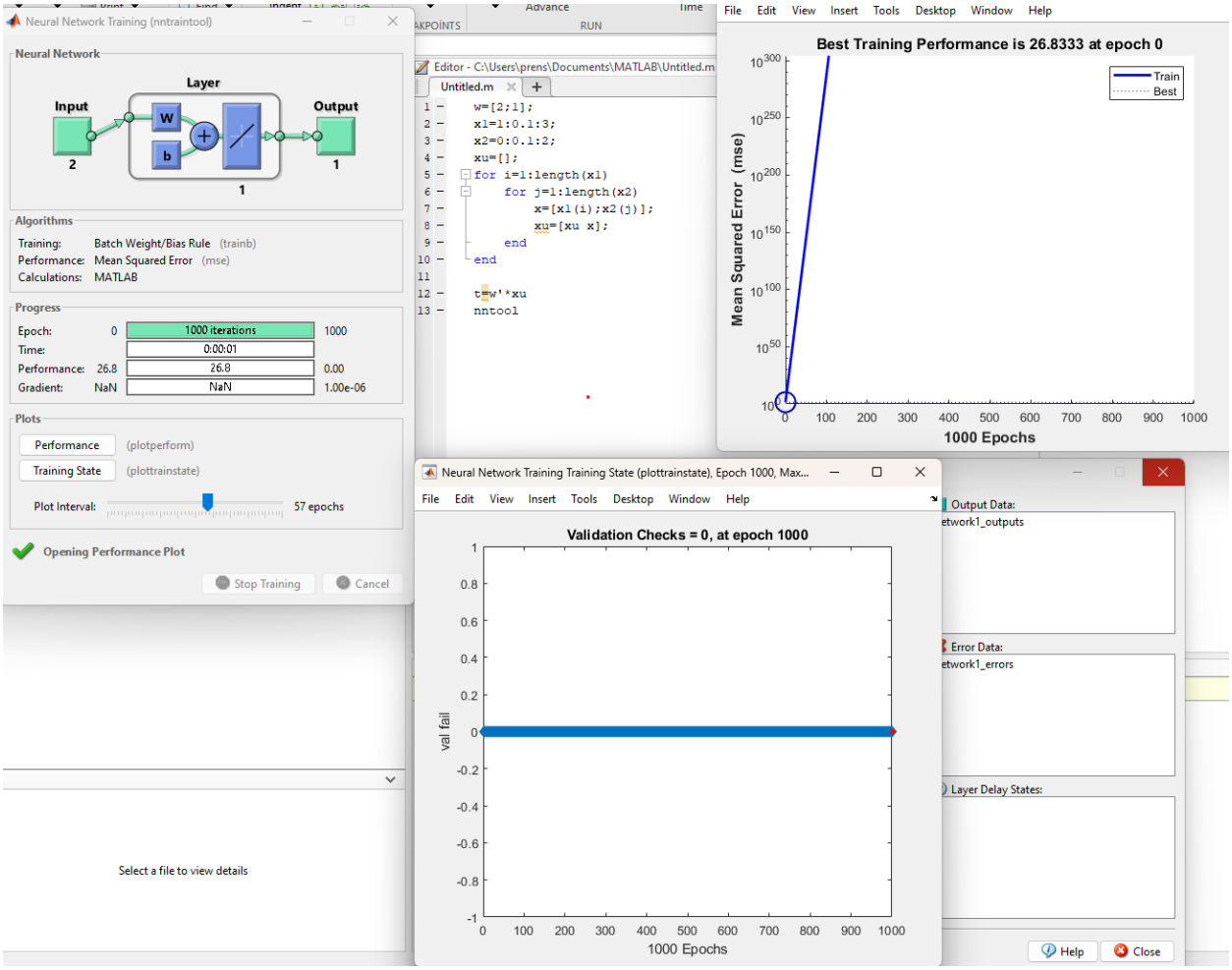
Command Window

```
   -0.2247    0.8341   -0.2085         0
   -0.2247   -0.2085    0.0021         0
   -0.2022   -0.4170    0.6255   -0.1111
   -0.2247   -0.2085    0.6255   -0.4444


w =

    0.9117    0.4558    0.2265
    0.0023   -0.2279    0.7926
    0.0023    0.4558    0.0023
   -0.2279    0.5698   -0.4529
   -0.3419    0.4558    0.3397


y =

     1     0     0     1
     0     0     1     0
     0     1     0     0
```

We can read classification out of y matrix. As we see we classified it correctly,

each column is different animal: fox, salmon, chicken, cat.

Lab3



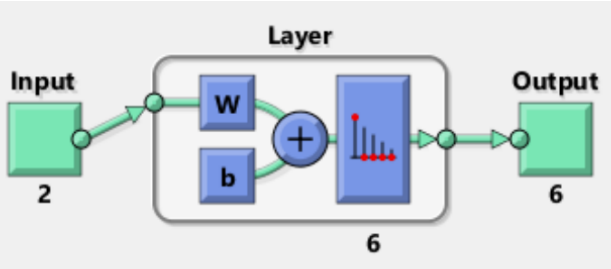
On the graph we can see that our value is increasing with every iteration, our method works properly,

LAB4

In this section, we're assessing our network's performance under conditions where the number of clusters is uncertain. To adapt to this uncertainty, we've expanded the number of neurons to 15. Here's the configuration of our network.

```
X = [-10 10; -5 5];
V = nngenc(X, 8, 10, 0.5);
plot(V(1,:), V(2,:), '+')
p=nngenc([0 1;0 1],6,5,0.05);

nnstart
```

```
Command Window
Error in untitled (line 6)
nntool

>> untitled
Error using nntool
nntool has been removed. Use nnstart instead.

Error in untitled (line 6)
nntool

>> untitled
>> untitled
>> clear
>> untitled
```



Data: network1_outputs_sim

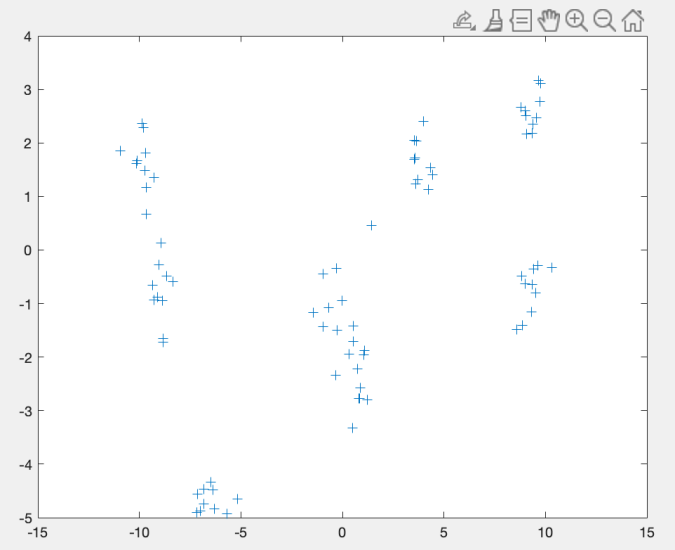
Value

```
[1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0;
 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0;
 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0;
 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1;
 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0;
 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0]
```

Data: network1_outputs

Value

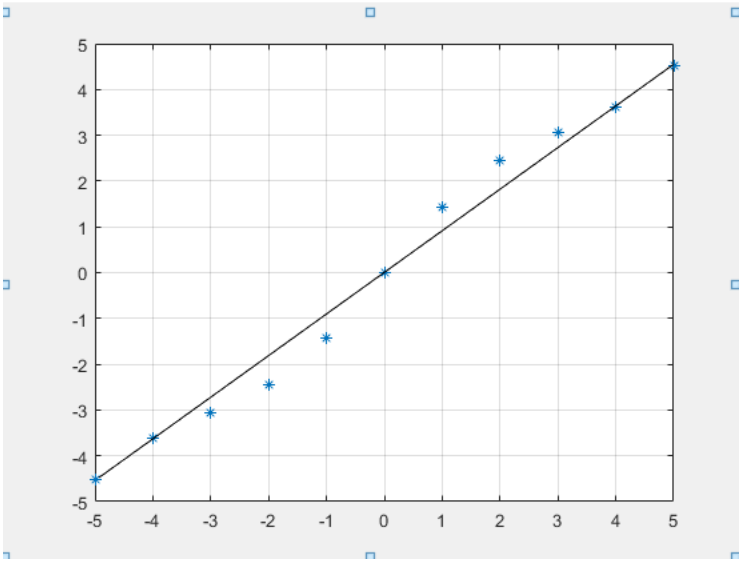
```
[1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0;
 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0;
 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0;
 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1;
 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0;
 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0]
```

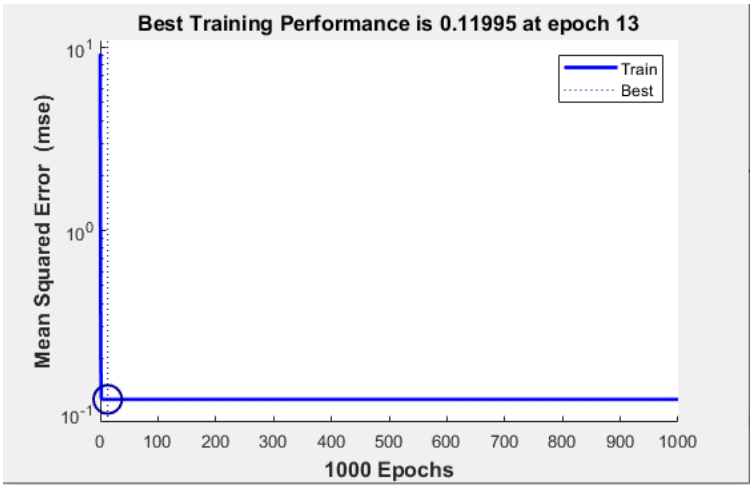


LAB5

FOR LINEAR

```
x=[-5:1:5];
y=[-5:1:5];
z=sin(x);
z=z.*0.5;
y=y+z;
plot(x,y,'*')
grid
```

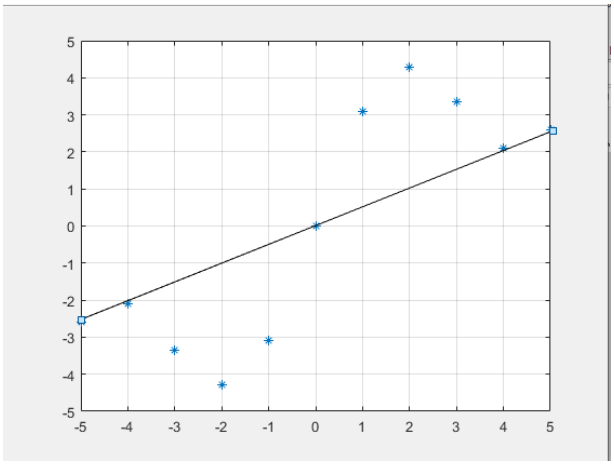


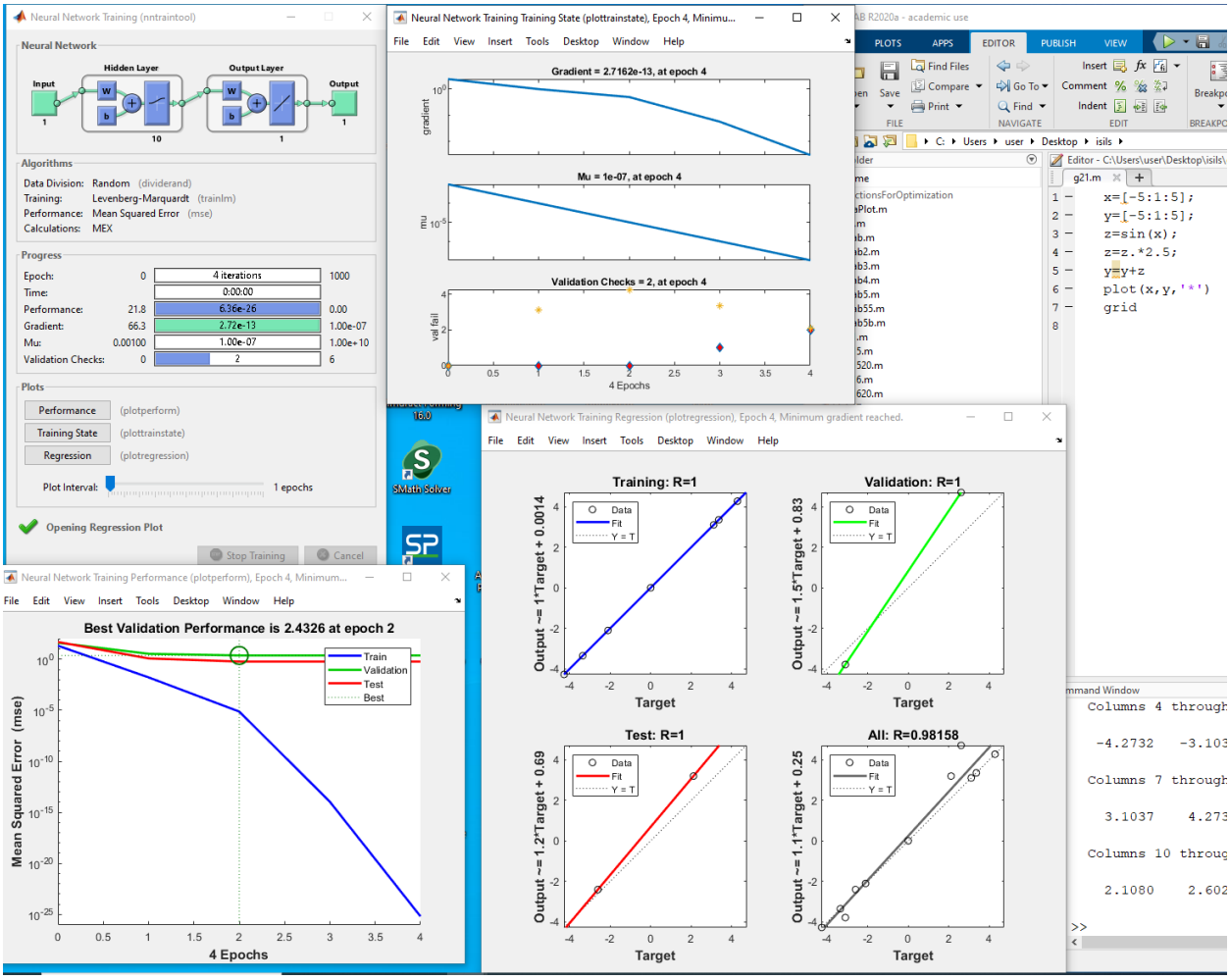


Despite this our approximation is sufficient as we can see in the figure. We could get a more accurate result by reducing learning rate.

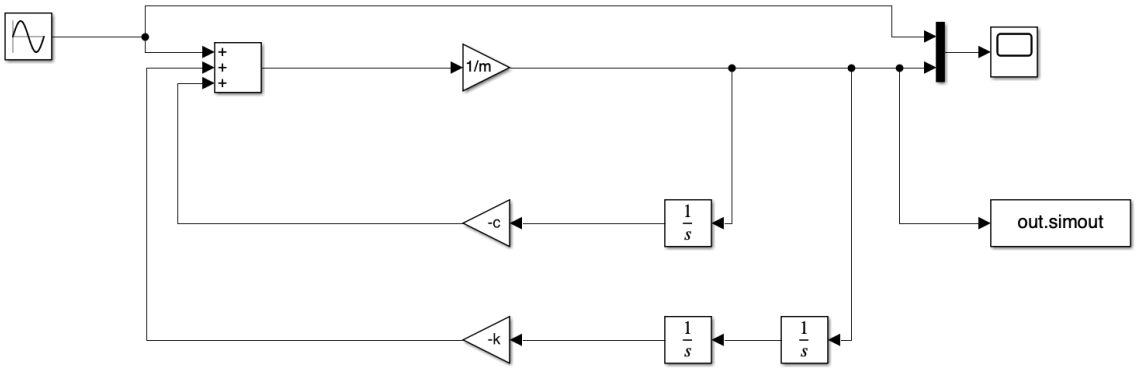
LAB5

FOR NON-LINEAR





Lab6




```
m=2; % mass
li = 0.1;
lj = 0.1;
k=1:li:15; % range of stiffness changes
c=1:lj:15; % range of damping changes
P=[]; T=[];% reset training set
% consider all combinations of parameters (K,C) in the ranges with some
step
for i=1:91
for j=1:91
% combine temporary values of parameters
T=[T [k(i); c(j)]']; % Save as a column of target matrix
end
end
a=sim('lab5model', 'StopTime','10'); % simulate system in ? Seconds
% save the output of the system as a column in input matrix
P=[P a.get('simout')];
```