Neural Network
Lecture 8

Jeong-Yean Yang

2020/12/10

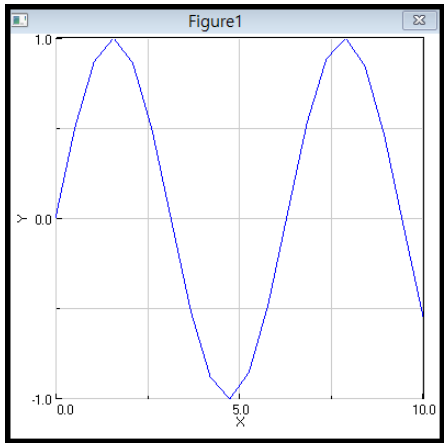
0 H.W. Neural Network

HW.1 Sigmoidal NN for Sin(x)

- 0<x<10 → X=linspace(0,10,20) N=20
- Y = sin(X)

```
x= linspace(0,10,n).T()
y= sin(x);
```

- Find the best NN result with Sigmoidal NN
  - W1 and W2 = zeros or randn
  - How many iterations are required?
  -



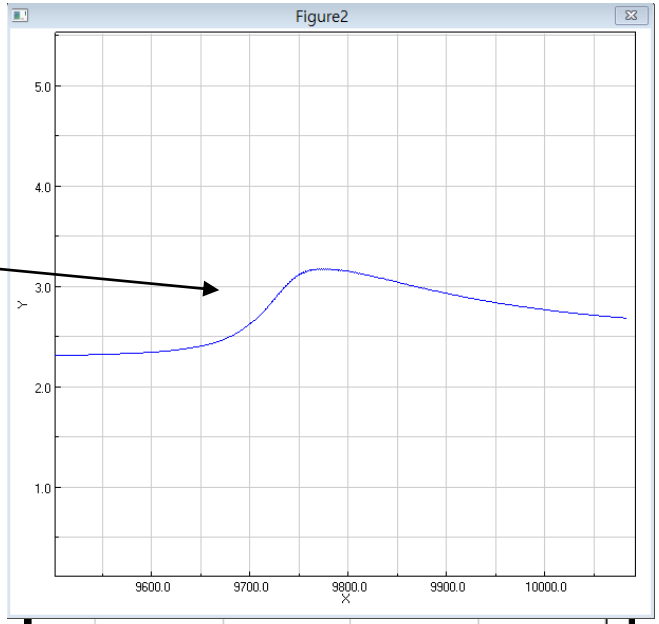
Y= sin(x)

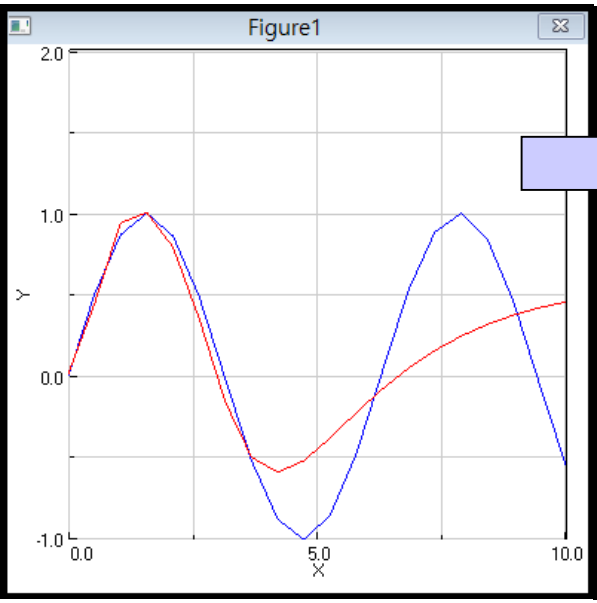
HW 2 Why J shows this Phenomenon?

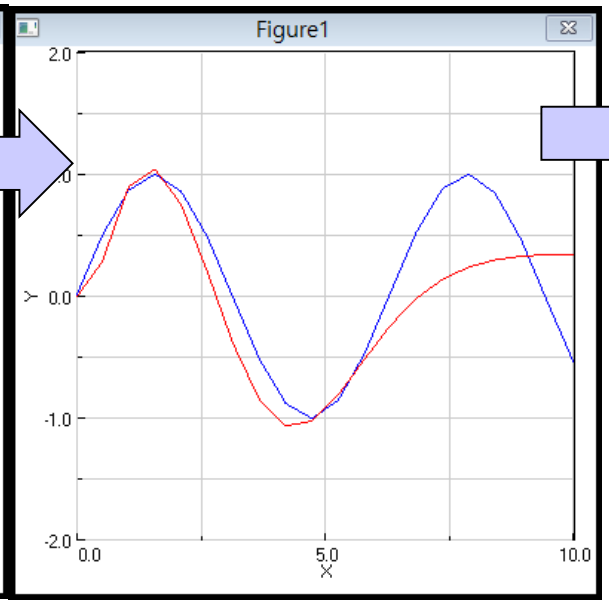
- During Learning process,

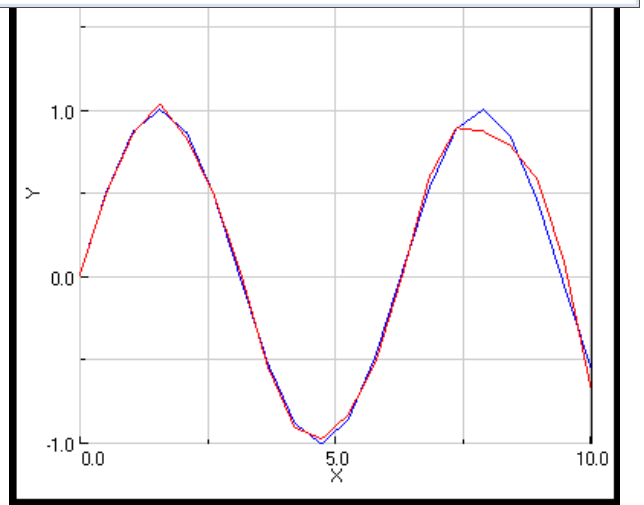
J shows sudden Jump.

Why?



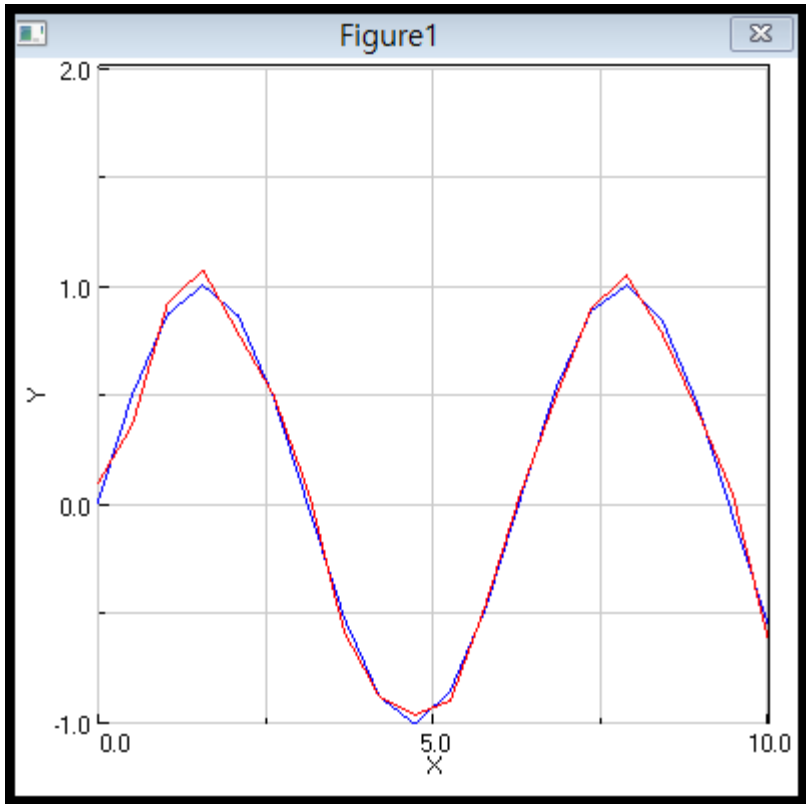






HW3. Using RBF for Y=Sin(x)

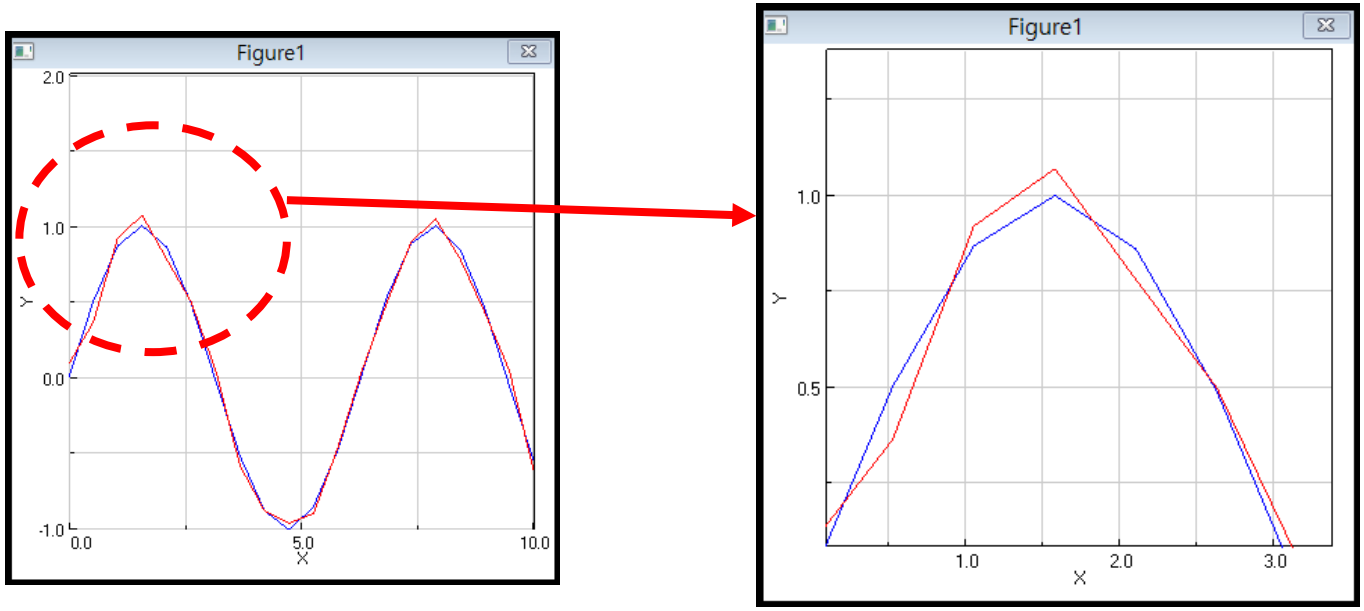
- 0<x<10 → X=linspace(0,10,20) N=20
- Y = sin(X)



HW 4. Tell me What the differences are Between Sigmoidal and RBF NN

- Iteration, Convergence, alpha, Initial value,
- Anything is O.K.

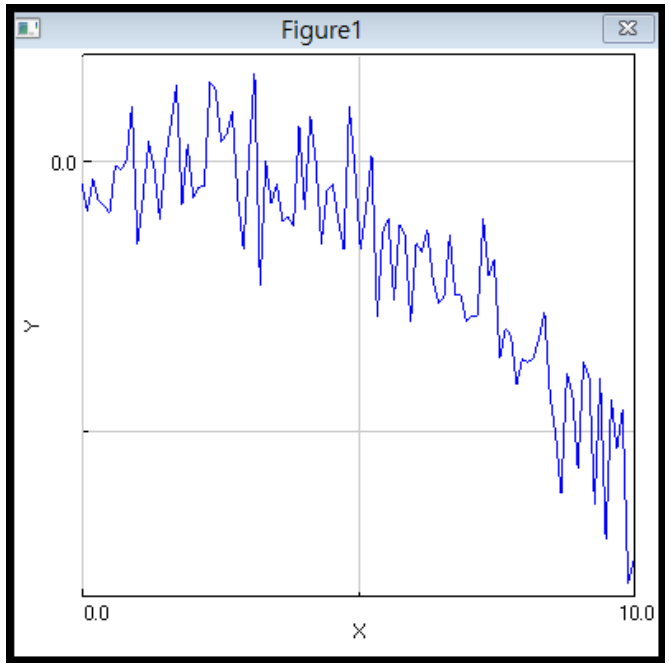
HW.5. Why Sin(x) is Not Smooth? Find the Answer and the **RESULT**



HW. 6 Noisy Data With RBF NN

- n=100
- X=linspace(0,10,n)
- y= -0.1*pow(x-2,2)+randn(n)

- Try RBF with the above x and y.
- RBF becomes what?



1 Unbalanced Cost Function

We used Squared Error

- Remind Differentiation for Gradient Descent Method

$$J = \frac{1}{2}\sum_{i \in D} e_i^{\,2} = \frac{1}{2}\sum_{i \in D} \| y_i - \hat{y} \|^2$$

$$\frac{\partial J}{\partial W_1}, \frac{\partial J}{\partial W_2}$$

$$W_1 \leftarrow W_1 - \alpha \frac{\partial J}{\partial W_1}$$

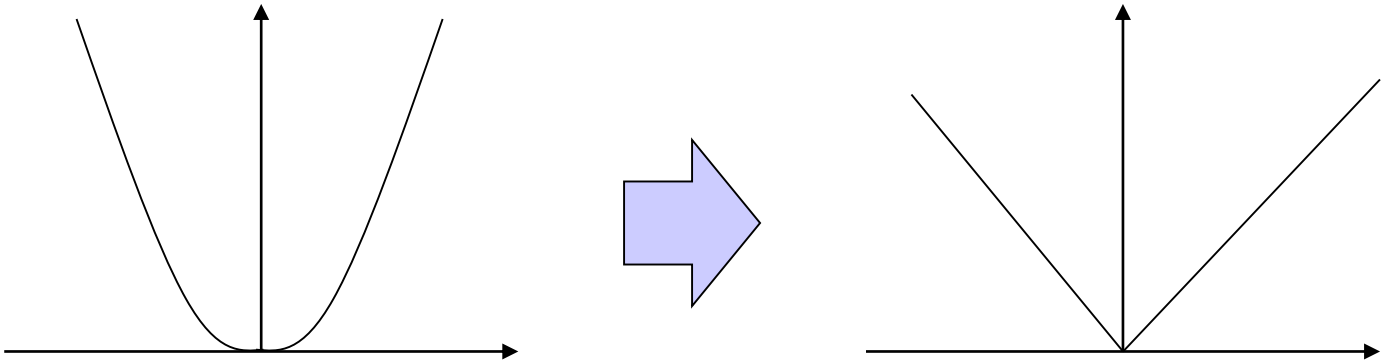
$$W_2 \leftarrow W_2 - \alpha \frac{\partial J}{\partial W_2}$$

- **Question:**

If we use Absolute Error, |e| , then What occurs?

Absolute Error

- Absolute Error is not well used because of Differentiation.
  - It is NOT continuous



$$J = \frac{1}{2}\sum_{i \in D} e_i^{\,2}$$

$$J' = \sum_{i \in D} e_i e'_i$$

$$J = \frac{1}{2}\sum_{i \in D} |\, e_i \,|$$

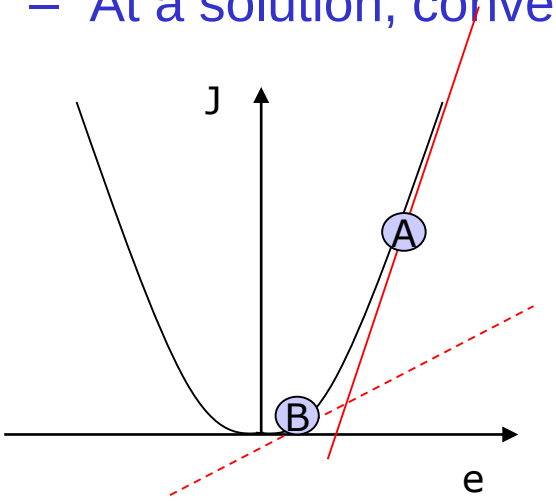
$$J' = \frac{1}{2}\sum \begin{cases} e_i' & e_i > 0 \\ -e'_i & e_i < 0 \\ 0 & e_i = 0 \end{cases}$$

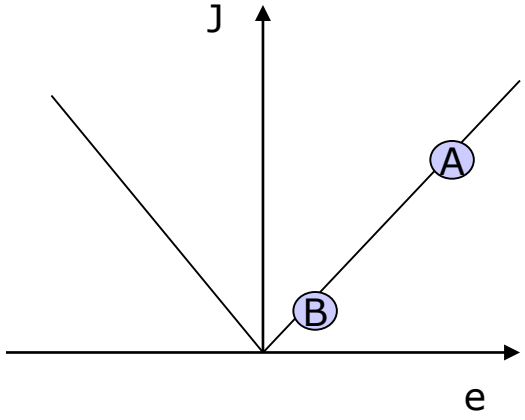
In Spite of All, Why We Concern |e|?

- Convergence
  - At a solution, convergence rate is too slow.

Remind

$$T = -K\,\mathrm{sgn}(s)$$





$$J = \frac{1}{2}\sum_{i \in D} e_i^2$$

$$J'_{w=A} > J'_{w=B}$$

Convergence Rate is NOT constant

$$J = \frac{1}{2}\sum_{i \in D} |e_i|$$

$$J'_{w=A} = J'_{w=B}$$

Convergence rate is constant
→ Sliding into a goal

Benito Fernandez

What Changes in OUR RBF Network?

$$J = \frac{1}{2}\sum_k {e_k}^2 = \frac{1}{2} e^T e$$

$$\frac{\partial J}{\partial W_2} = e^T \frac{\partial e}{\partial W_2}$$

$$\longrightarrow$$

$$J = \sum_k |e_k|$$

$$\frac{\partial J}{\partial W_2} = (+,-,0)\frac{\partial e}{\partial W_2}$$

New equation is required!

$$\frac{\partial J}{\partial W_2} = e^T \frac{\partial e}{\partial W_2} = -e^T \frac{\partial Y}{\partial W_2} = -e^T \frac{\partial [Y_1 \quad I] W_2}{\partial W_2} = -{e^T}_{1\times n}[Y_1 \quad I]_{n\times(h+1)}$$

$$Transpose \rightarrow Vector = \left(\frac{\partial J}{\partial W_2}\right)^T_{(h+1)\times 1} = -\left([Y_1 \quad I]_{n\times(h+1)}\right)^T e_{n\times 1} = -[Y_1 \quad I]^T e$$

How we solve Matrix Row-Column Problem in this |e| Network?

See Derivative of Error in RBF NN

$$J = \frac{1}{2}\sum_k e_k{}^2 = \frac{1}{2} e^T e$$

$$\frac{\partial J}{\partial W_2} = e^T \frac{\partial e}{\partial W_2}$$

$$\left.\frac{\partial J}{\partial W_2}\right|_{1\times(h+1)} = e^T \frac{\partial e}{\partial W_2} = -e^T \frac{\partial Y}{\partial W_2}$$

$$= -e^T \frac{\partial [Y_1 \quad I] W_2}{\partial W_2} = -e^T{}_{1\times n} [Y_1 \quad I]_{n\times(h+1)}$$

$$J = \sum_k |e_k|$$

$$\frac{\partial J}{\partial W_2} = (+,-,0)\frac{\partial e}{\partial W_2}$$

$$\left.\frac{\partial J}{\partial W_2}\right|_{1\times(h+1)} = (\pm,0)^T \frac{\partial e}{\partial W_2} = -(\pm,0)^T \frac{\partial Y}{\partial W_2}$$

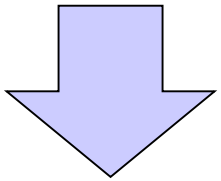
$$= -\begin{pmatrix} 1 \\ -1 \\ 0 \\ \cdots \end{pmatrix}^T_{1\times n} [Y_1 \quad I]_{n\times(h+1)}$$

- Error vector e, is replaced by [+, -, or 0] vector

Derivatives of W1 in RBF NN

$$J = \frac{1}{2}\sum_k {e_k}^2$$

$$\frac{\partial J}{\partial W_1} = -2\sum_k \left(eW_2^T\right)_k \circ \Phi_k \circ Z_k = -2\sum_k \left(eW_2^T\right)_k \circ Y_{1,k} \circ Z_k$$



$$J = \sum_k |\, e_k \,|$$

$$\frac{\partial J}{\partial W_1} = -2\sum_k \left(\begin{pmatrix} 1 \\ -1 \\ 0 \\ \dots \end{pmatrix} W_2^T \right)_k \circ \Phi_k \circ Z_k = -2\sum_k \left(\begin{pmatrix} 1 \\ -1 \\ 0 \\ \dots \end{pmatrix} W_2^T \right)_k \circ Y_{1,k} \circ Z_k$$

Exampl) l8abs1.py

$$J = \sum_k |e_k|$$

$$\begin{pmatrix} 1 \\ -1 \\ 0 \\ \dots \end{pmatrix}_{n\times 1} = \mathrm{sgn}(e)$$

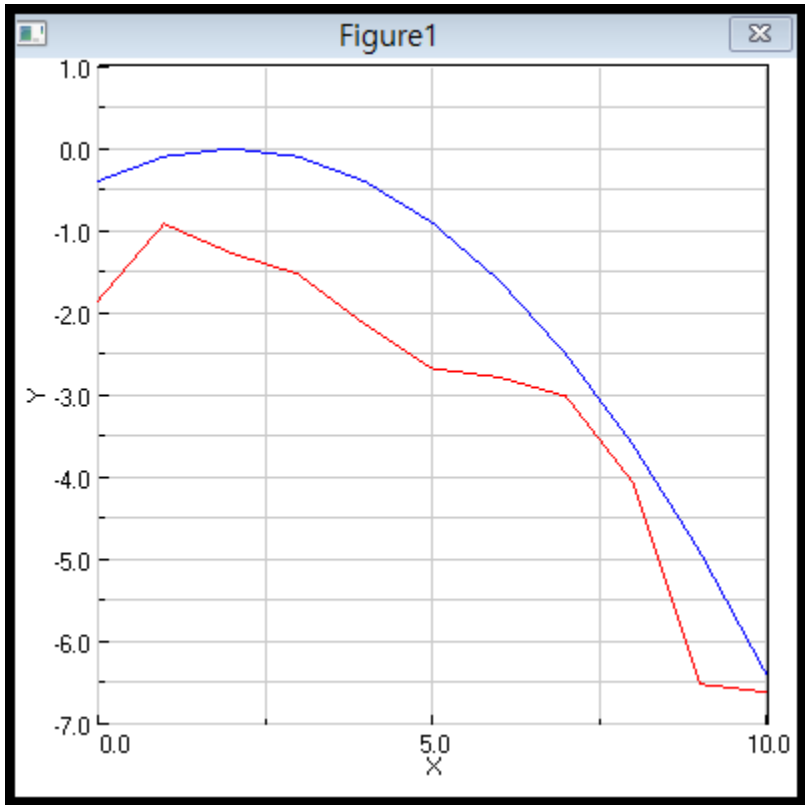
```
Y1[:,1:h]    = exp(-Z.mul(Z))
Y            = Y1*W2

# error
e            = y-Y
J            = sum(ABS(e))
de           = SGN(e)

# weight update
dW2 = -Y1.T()*de
dW1 = -2*((de*W2[1:h,:].T()).mul(Y1[:,1:h])).mul(Z)
dW1 = sum(dW1,1);

W1  = W1-alpha*dW1
W2  = W2-alpha*dW2
```

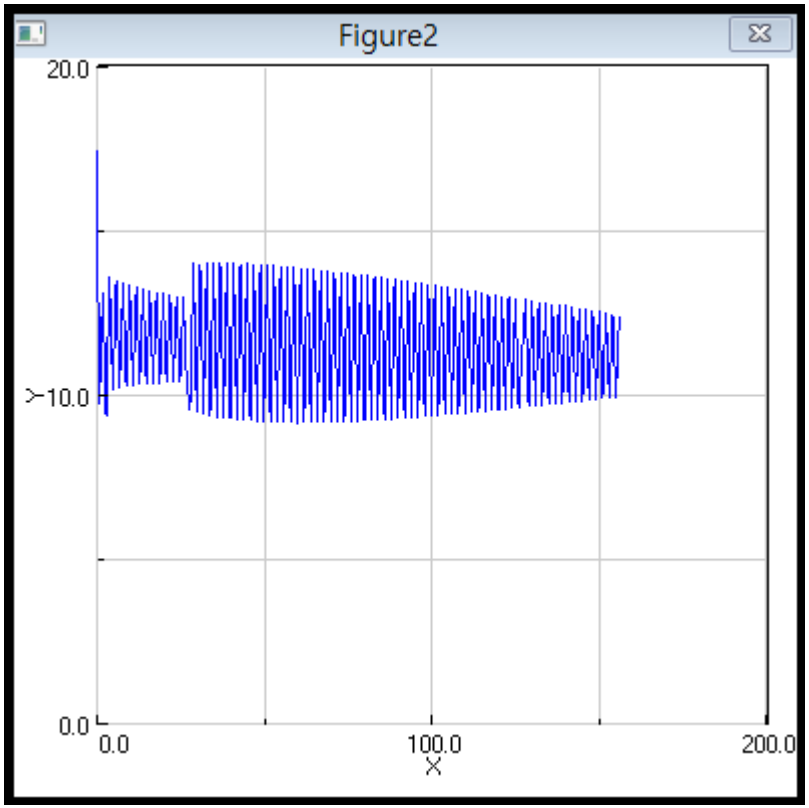
RBF-NN $J = \frac{1}{2}\sum_k e_k^2$

```
dW1 = -2*((e*W2[1:h,:].T()).mul(Y1[:,1:h])).mul(Z);
```

Example) l8abs1.py

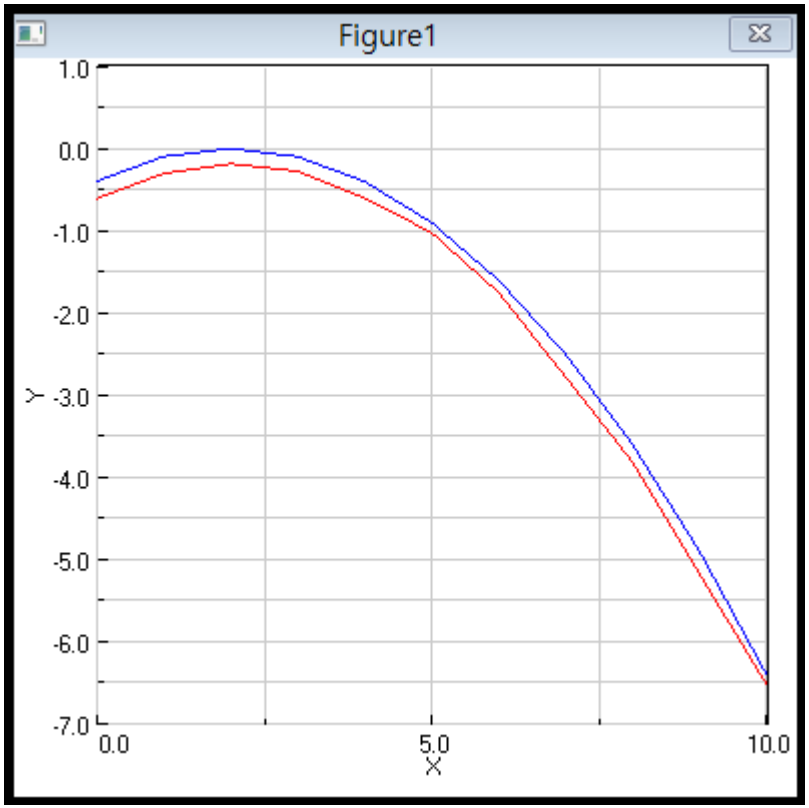
- Alpha=0.1 → Too many oscillations(chattering)

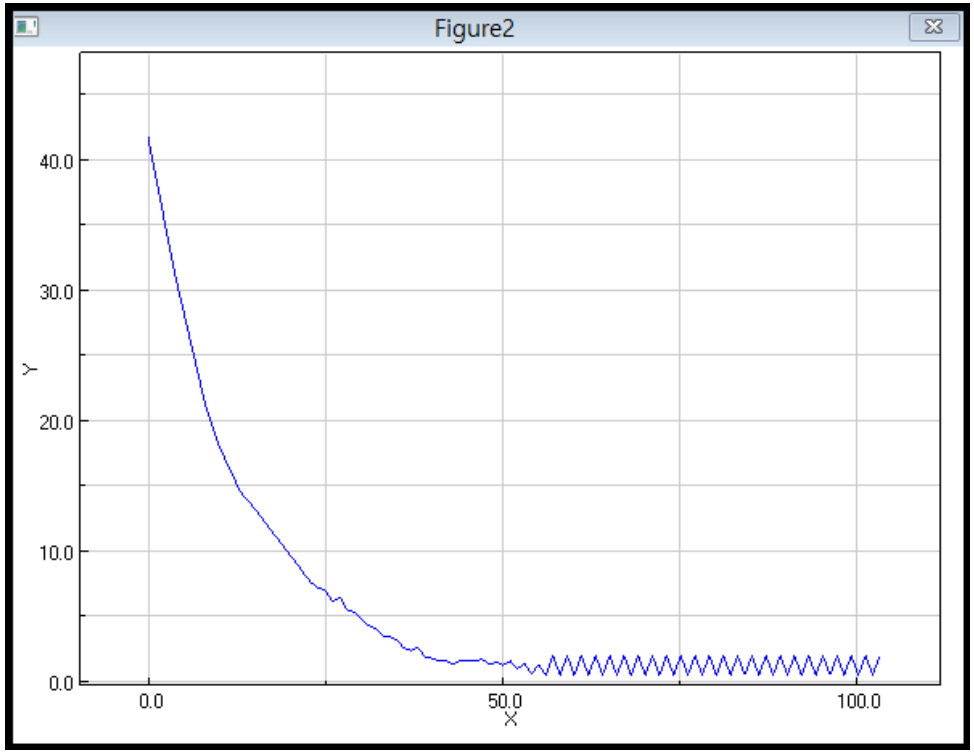




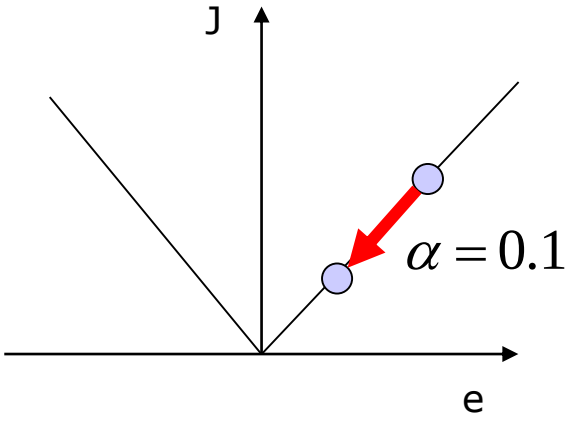
Example) l8abs1.py with Small Alpha

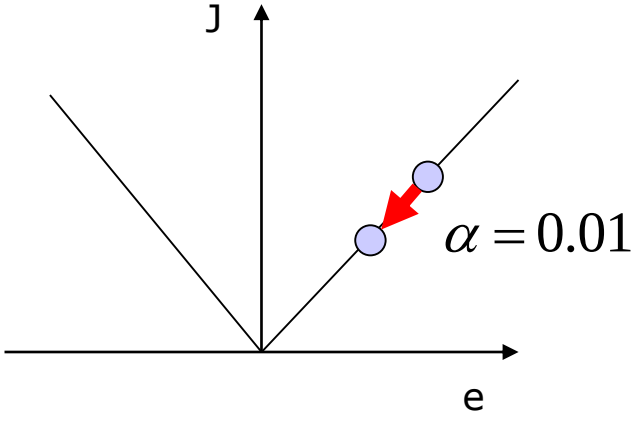
- Alpha=0.01 → Too many oscillation



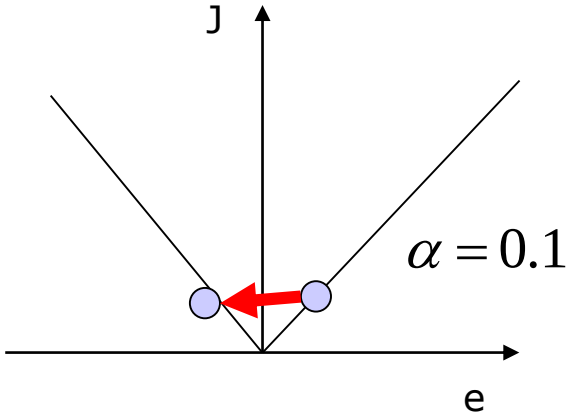


Alpha is the Key for Chattering
Remind Gradient Descent method

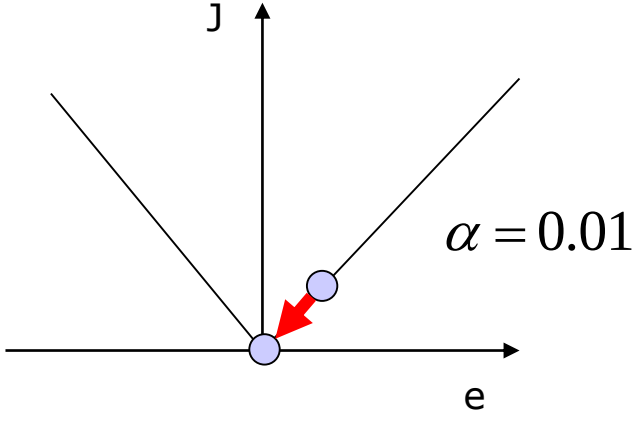




Big alpha moves faster and farther.

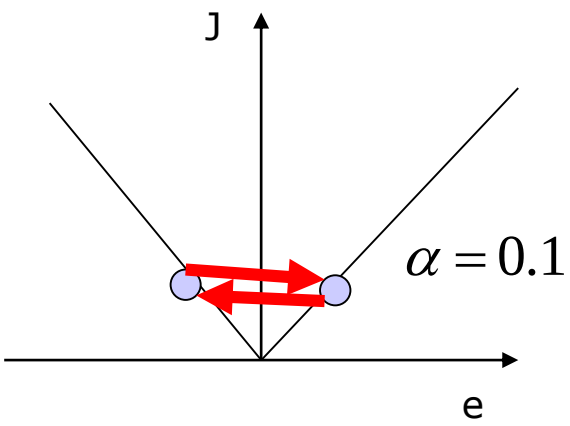


Large Chattering

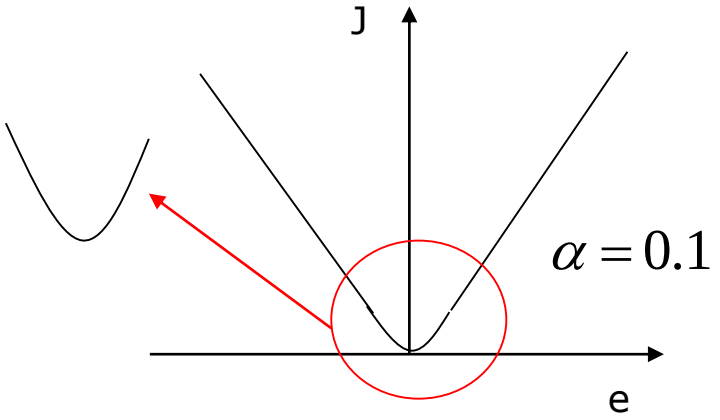


Small chattering

Alternative Strategy for Small Chattering



When near J=0, differentiation is NOT continuous.



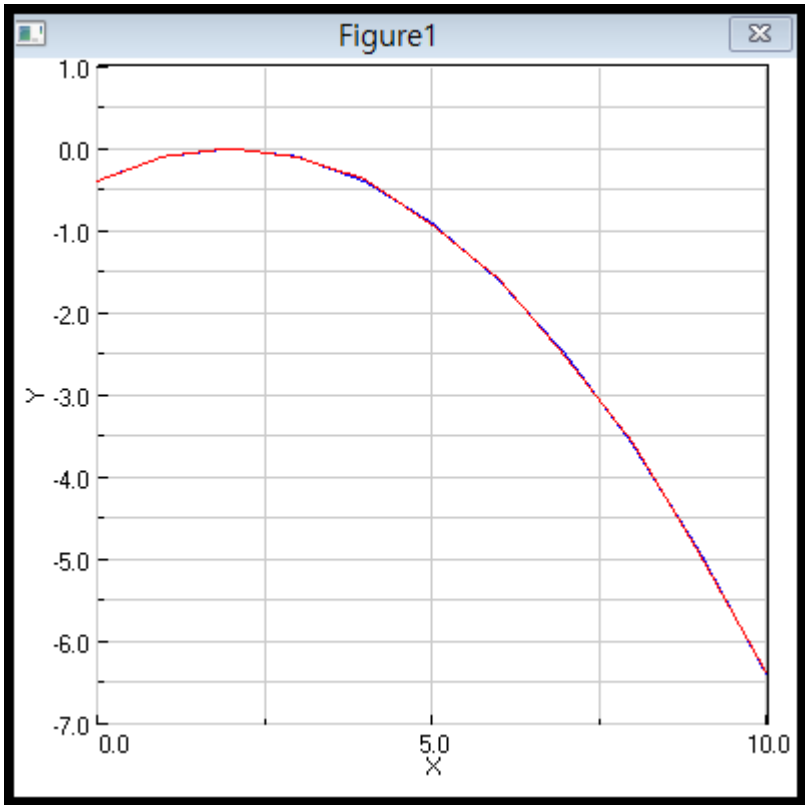
We can use Hybrid method.

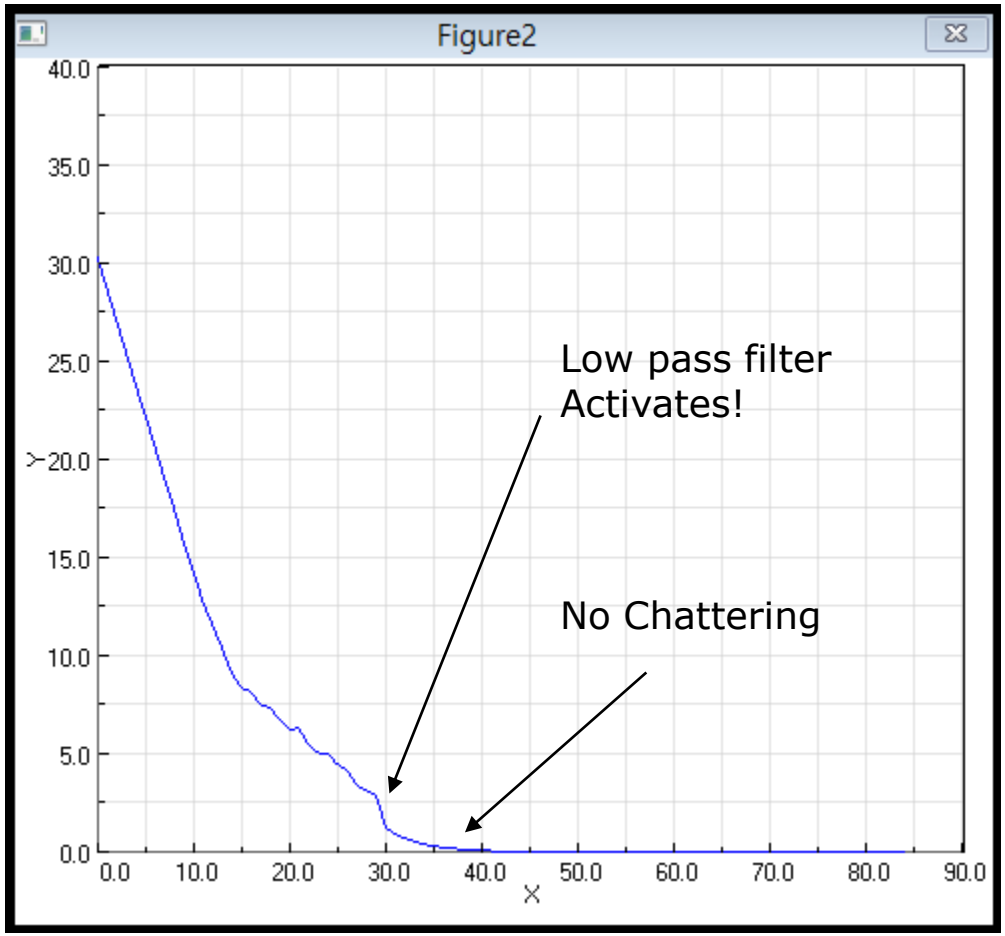
In Small error regions, we use e^2

Insight from Sliding Mode Control(with Low pass filter)

$$\begin{cases} |\mathrm{e}| < \delta : \mathrm{J} = \sum e^2, \ \nabla \mathrm{J} = 2\sum ee' \\ |\mathrm{e}| > \delta : \mathrm{J} = \sum |e|, \nabla J = \sum [1, -1, 0]e' \end{cases}$$

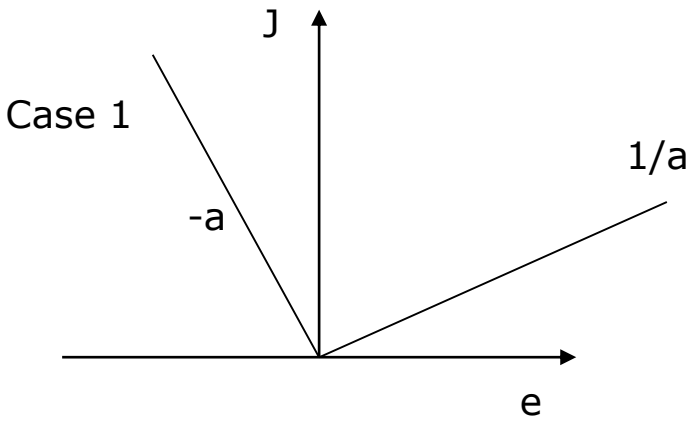
Example) l8abs2.py

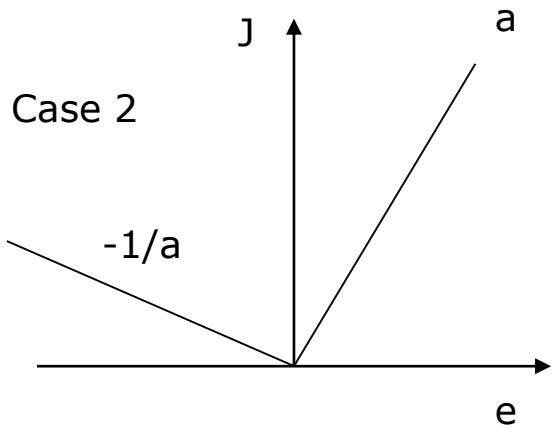




Another Idea of |e|

- Unbalanced Error





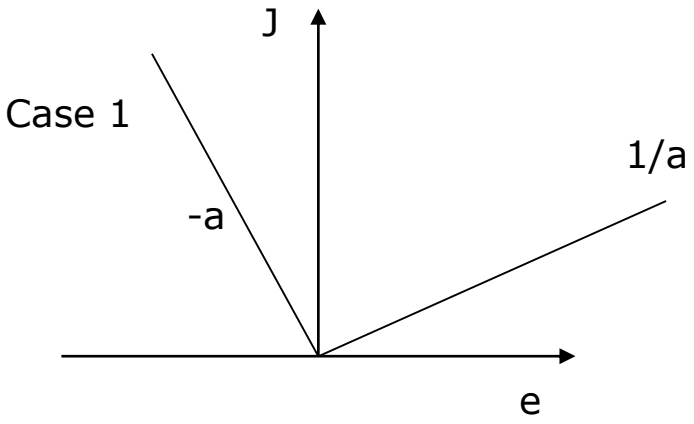
What is it?

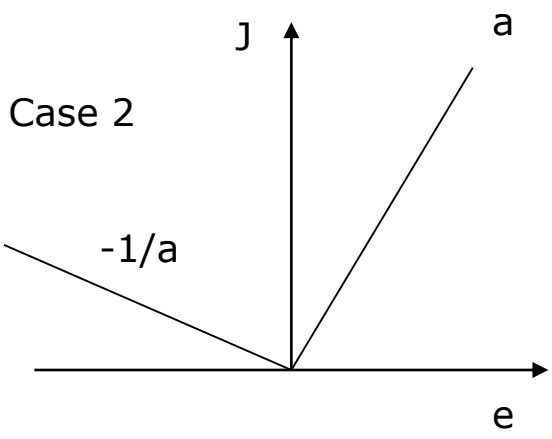
Case 1: if e>0, very generous.
if e<0, very tough.

Example) l8abs3.py

```
# error
e               = y-Y
J               = sum(ABS(e))

for i in range(1,n+1):
    if (e[i]>0):
        de[i]    = 1/a
    elif (e[i]<0):
        de[i]    = -a
    else:
        de[i]    = 0
```

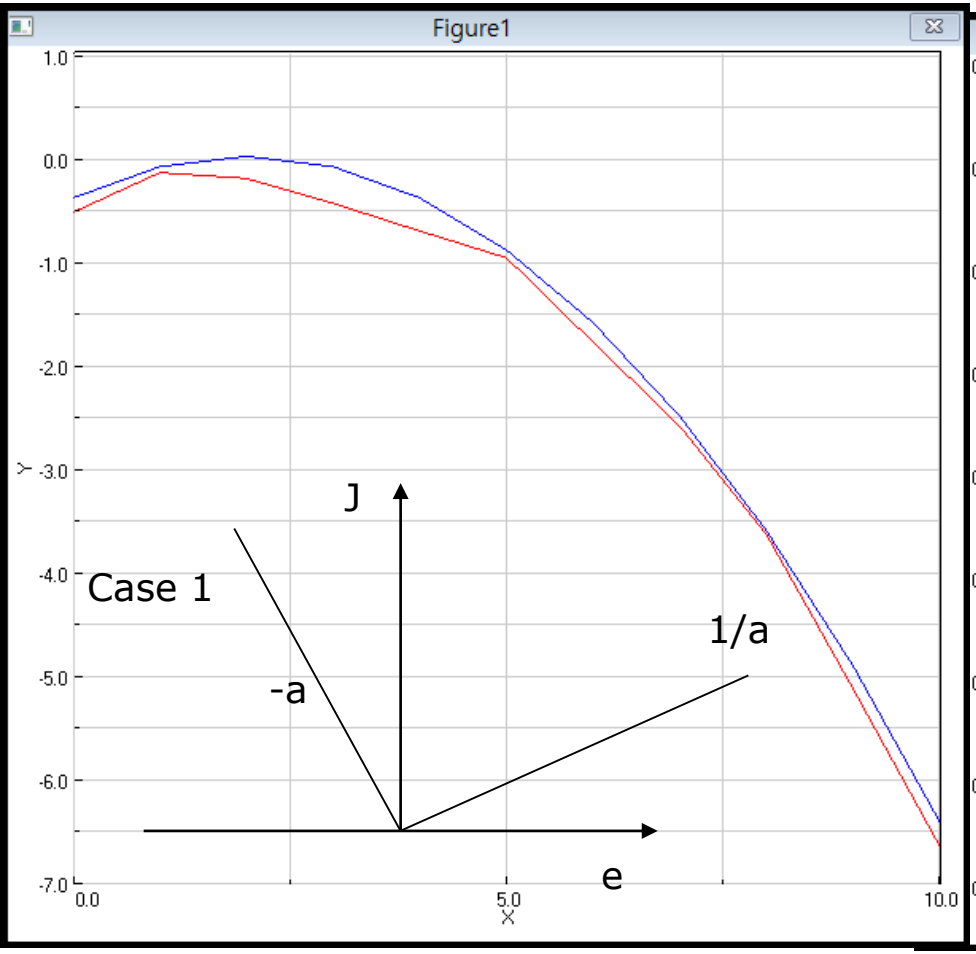
- a= 3 or 1/3

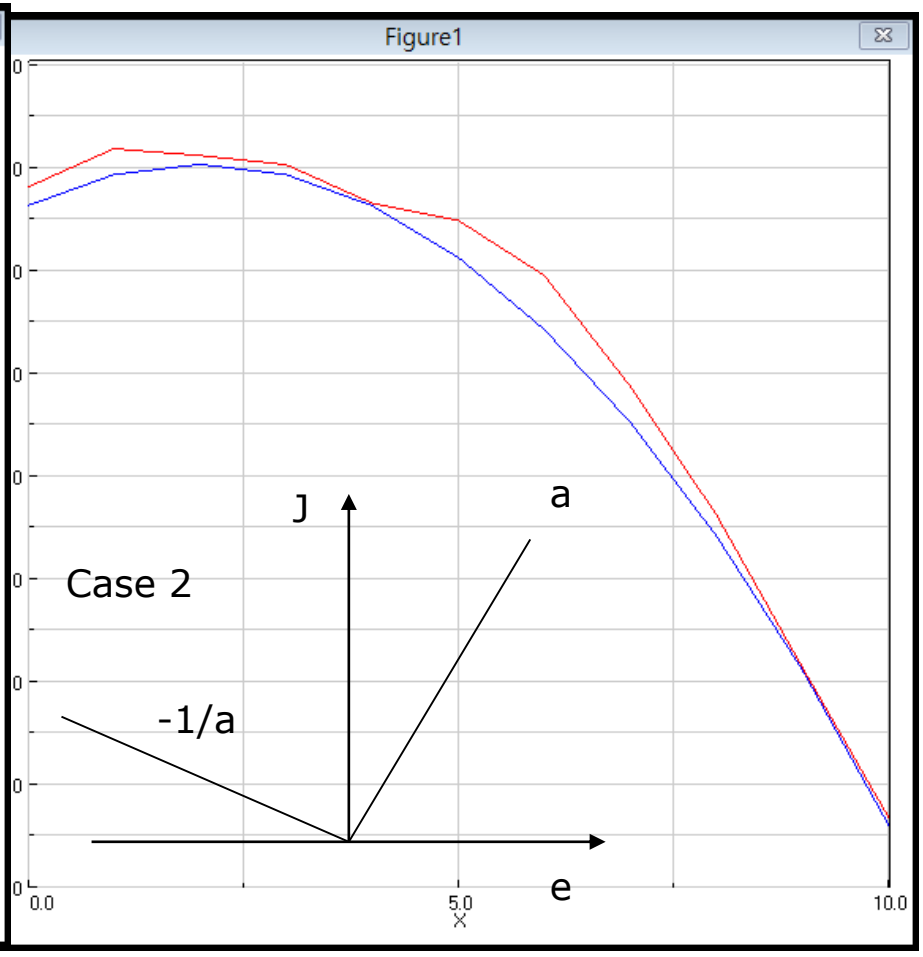




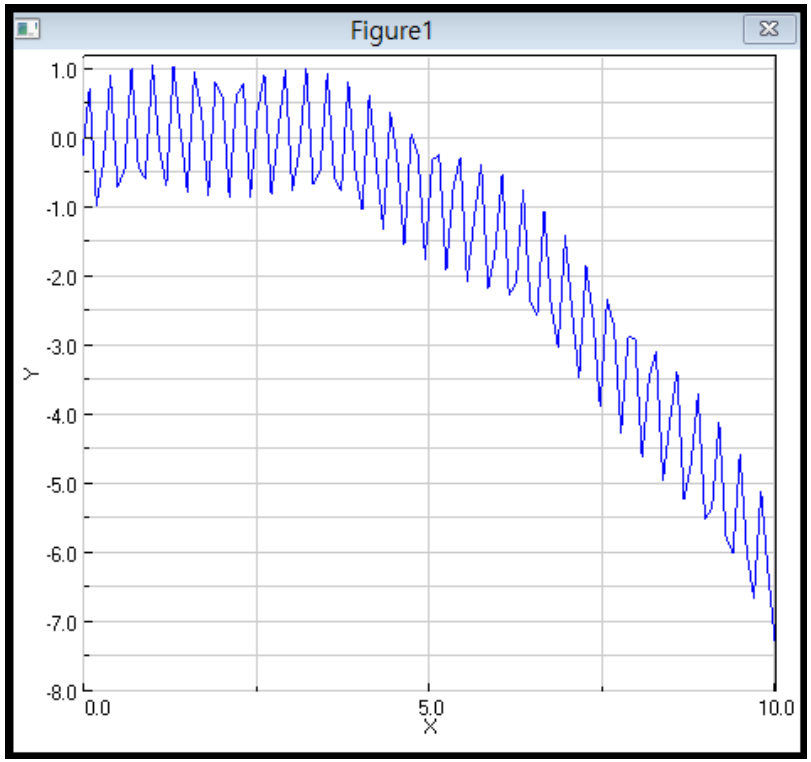
Can You Image the Result?

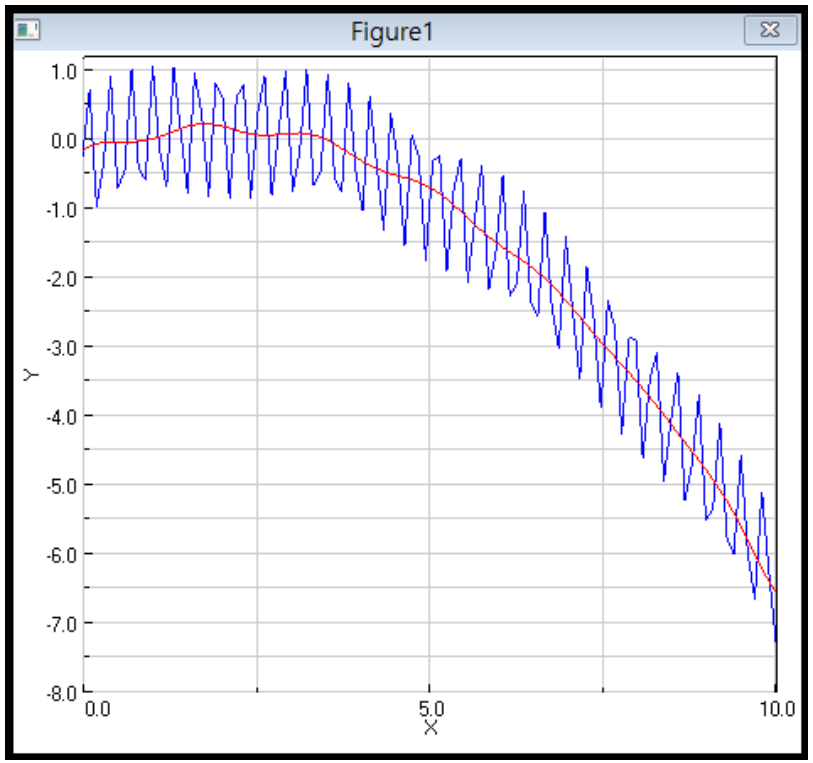
- Example) test a=3 and a=1/3 for case I





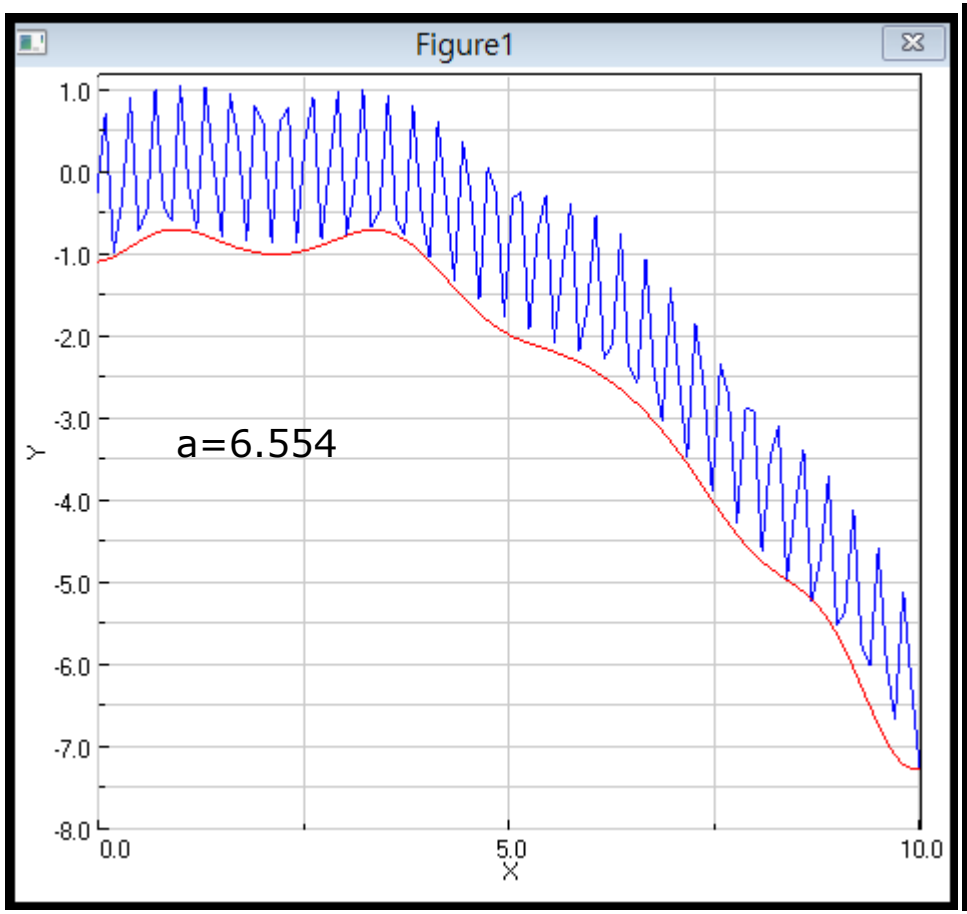
RBF in Noisy Signal (l8abs4.py)

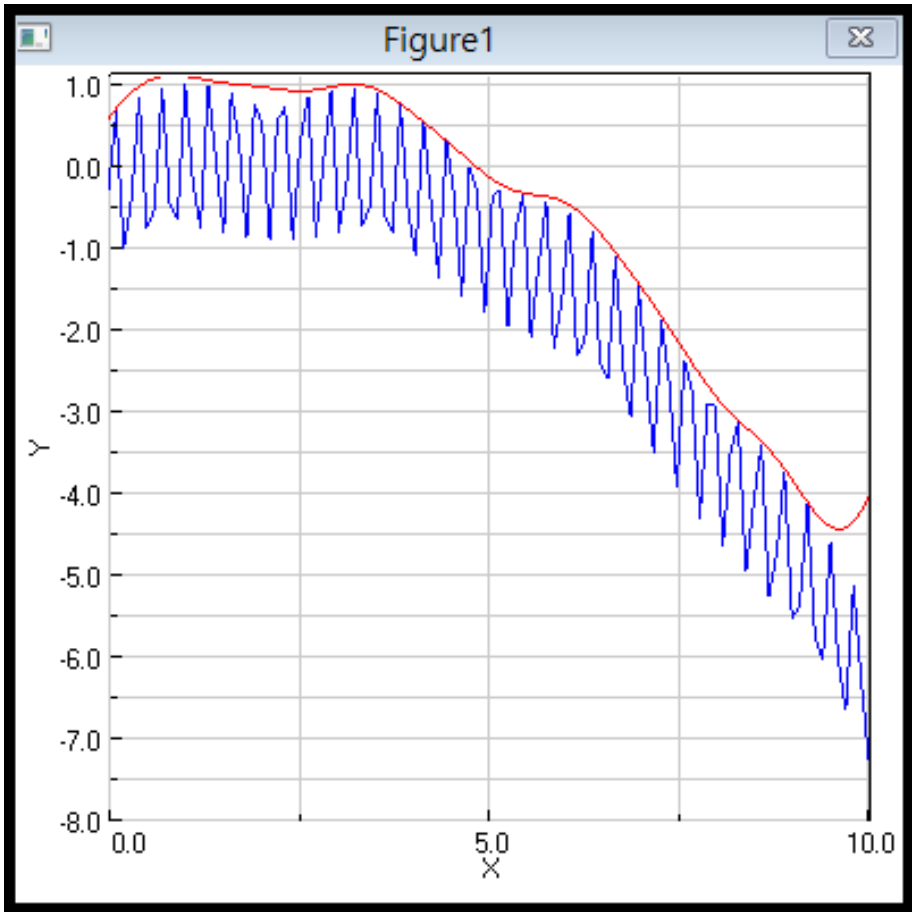




- RBF learning in the Noisy Signal, y= f(x)+ sin(20x)
- The results becomes, **mean value**

Unbalanced Error with Noisy Signal





How we find the magic number, 6.554?