Solving ODE’s with PINN

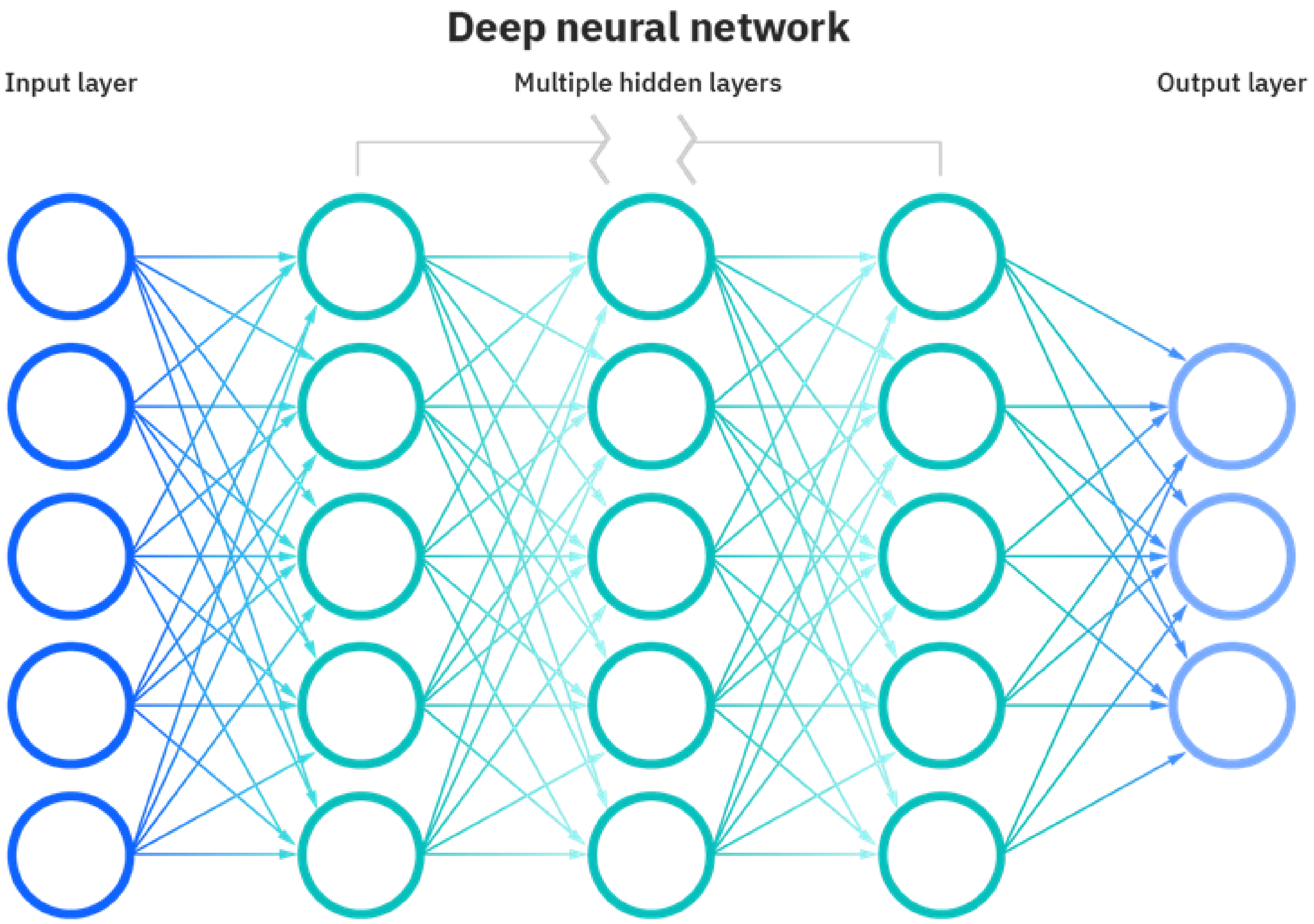
AN APPLICATION OF PHYSICS-INFORMED NEURAL NETWORK

Why we use this method?

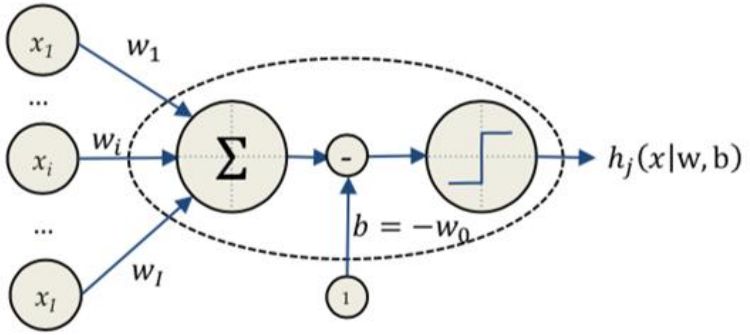
- GOAL: Solving the inverse problem (model parameter estimation) for second order ODE
- HOW: PINNs can be used to directly learn the solution of system dynamics

  RK implement directly in the RNN cell compensete the few available data
- BENEFITS: Reduce the heavy computational burden associated with time-domain simulations

- Neural Networks are a subset of Machine Learning and their structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.



- Neural Network is composed of layers, which can be divided into cells (neurons). If we zoom on one, we can notice that a neuron is actually a linear regression of all the neurons in the previous layer.



$$h_j(x|\mathrm{w},\mathrm{b}) = \mathrm{h_j}\left(\Sigma_{i=1}^{I} w_i \cdot x_i - b\right) = \mathrm{h_j}\left(\Sigma_{i=0}^{I} w_i \cdot x_i\right) = h_j(w^T x)$$

How it works

- A neural network has to be trained to be efficient and significant.
- During the training, we minimize a defined loss function in order to obtain the parameters.
- In our problem ,we use a recurrent neural network (RNN).
- RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer.

Trade Off

- The **learning rate** is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function
- An **epoch** means training the neural network with all the training data for one cycle

Runge Kutta method

To solve second order ODE the method is:

$$\begin{bmatrix} \dot{\mathbf{y}}_{n+1} \\ \mathbf{y}_{n+1} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{y}}_n \\ \mathbf{y}_n \end{bmatrix} + h \sum_i b_i \boldsymbol{\kappa}_i \,, \qquad \boldsymbol{\kappa}_i = \begin{bmatrix} \mathbf{k}_i \\ \bar{\boldsymbol{k}}_i \end{bmatrix},$$

where

$$\mathbf{k}_1 = f(\mathbf{u}_n, \dot{\mathbf{y}}_n, \mathbf{y}_n)$$

$$\bar{\boldsymbol{k}}_1 = \mathbf{y}_n$$

$$\mathbf{k}_i = f\left(\mathbf{u}_{n+c_i h}, \dot{\mathbf{y}}_n + h \sum_j^{i-1} a_{ij} \mathbf{k}_j, \mathbf{y}_n + h \sum_j^{i-1} a_{ij} \bar{\boldsymbol{k}}_j\right)$$

$$\bar{\boldsymbol{k}}_i = \mathbf{y}_n + h \sum_j^{i-1} a_{ij} \bar{\boldsymbol{k}}_j.$$

The matrix A is called RK matrix, **b** and **c** are the weights and nodes.

Quick overview of the method

- Explicit method: k_i depends only on k_j with j<i
- Consistent method: $\sum_{i=i}^{s} b_i$=1
- Convergence order= O(h^p) where p is less then the number of stages

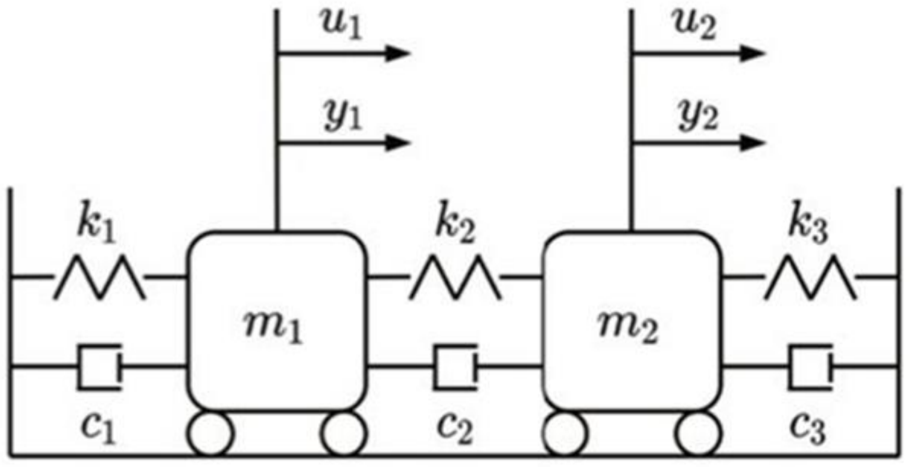
In order to respect these properties we obtain:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1/6 \\ 1/3 \\ 1/3 \\ 1/6 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 0 \\ 1/2 \\ 1/2 \\ 1 \end{bmatrix},$$

Masses problem

- Model parameter identification of a dynamic two-degree-of-freedom system through Runge–Kutta integration.
- We consider the motion for two masses linked together springs and dashpots
- We assume that while the masses and spring coefficients are known, the damping coefficients are not: we want to extimate them

$$\mathbf{P}(t)\frac{d^2\mathbf{y}}{dt^2} + \mathbf{Q}(t)\frac{d\mathbf{y}}{dt} + \mathbf{R}(t)\mathbf{y} = \mathbf{u}(t)$$

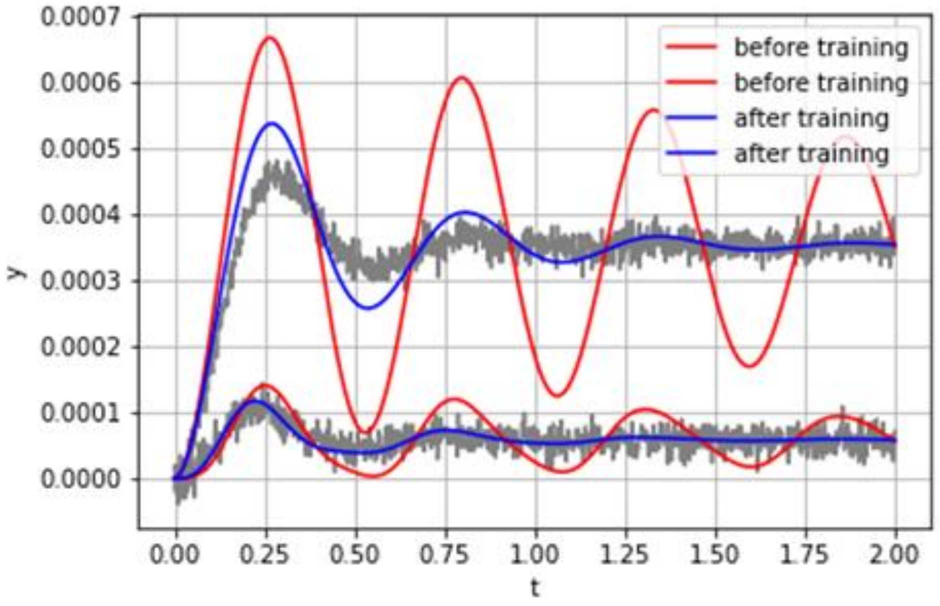


(a) Two degree of freedom system

Our strategy

- We choose the neural network parameters (in the example were given)
- We test the convergence of the runge kutta method with different time steps
- We try to add a gaussian noise with different std in different time intervals

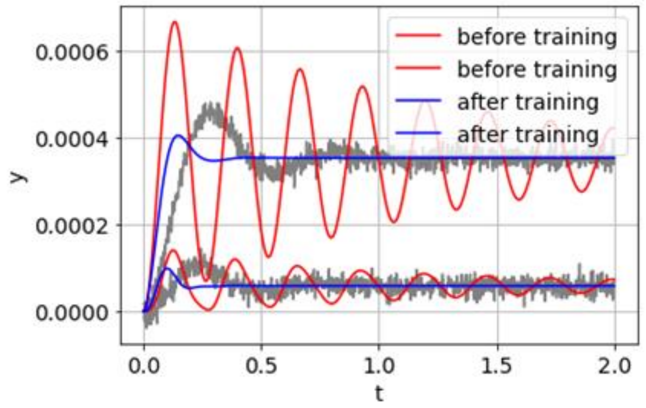
h=0,002
Output = [115.1 71.6 16.7]

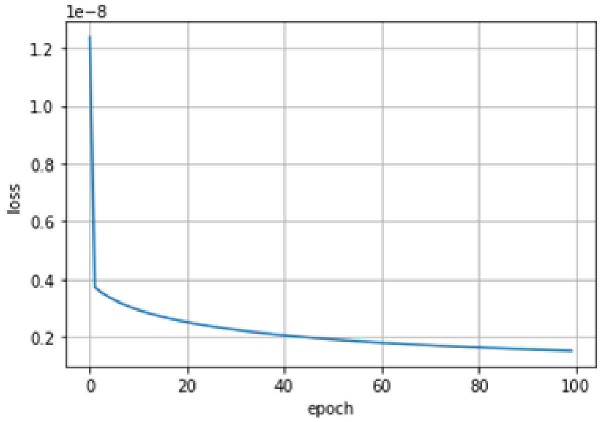


Different time step of RK

We notice that we can't use a bigger time step, because the output value is really bad

h = 0,004
Output= [180.62221 ,
102.55373 ,
21.072803]

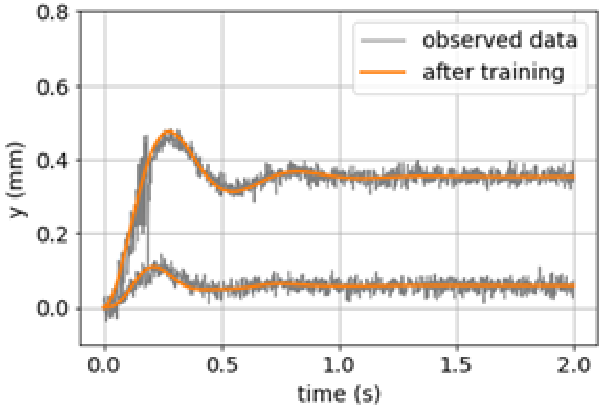


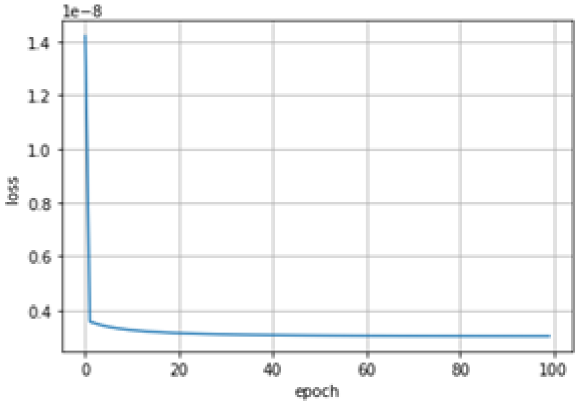


Gaussian noise

time interval [0,0.2]

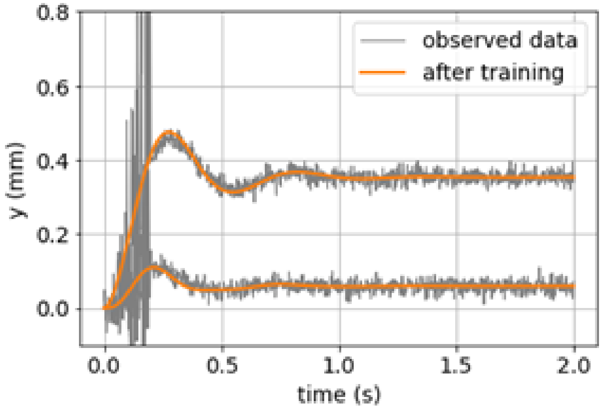
std=5

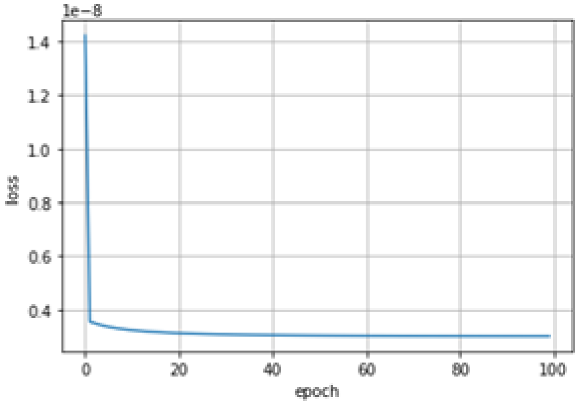




Output:
[117.79279 ,
72.11025 ,
17.402443]

std=100



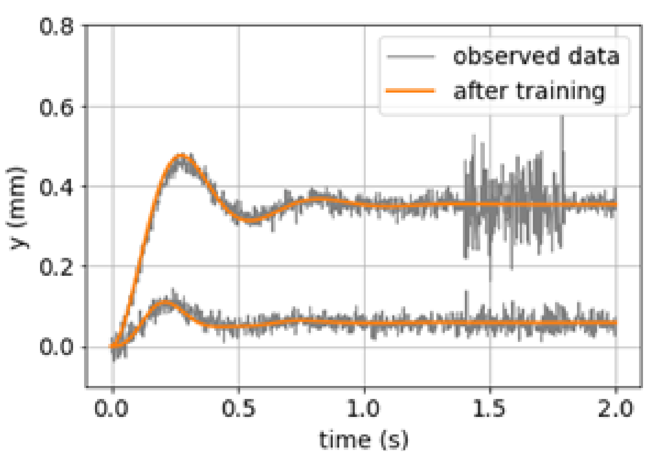


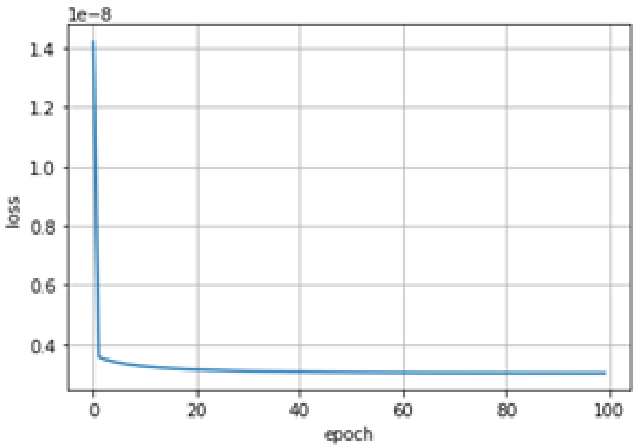
Output:
[109.87381 ,
72.30827 ,
14.474802]

Gaussian noise

time interval [1.4,1.7]

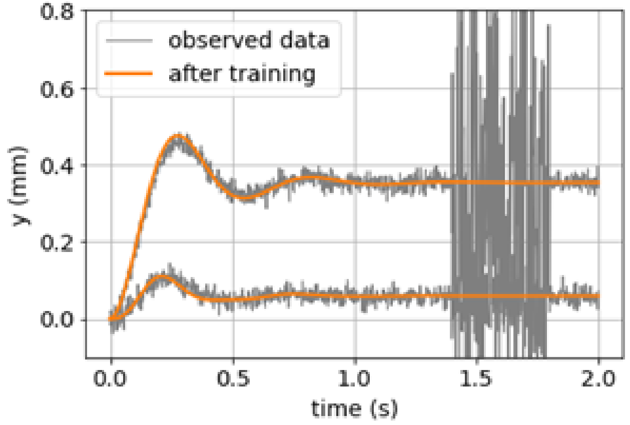
std=5

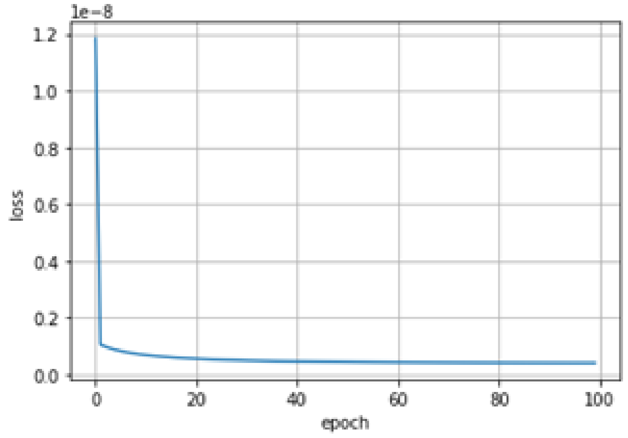




Output:
[115.152145,
71.78918 ,
16.636393]

std=100



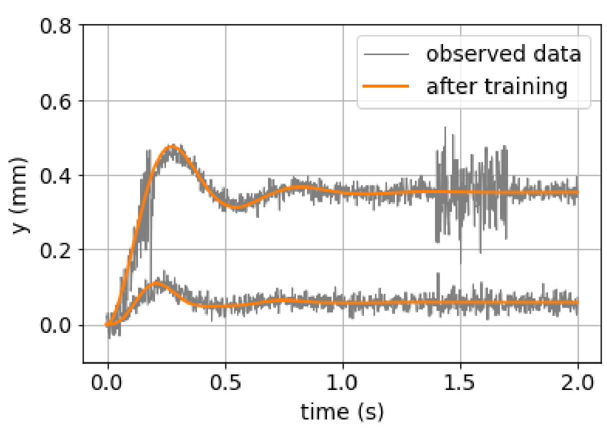


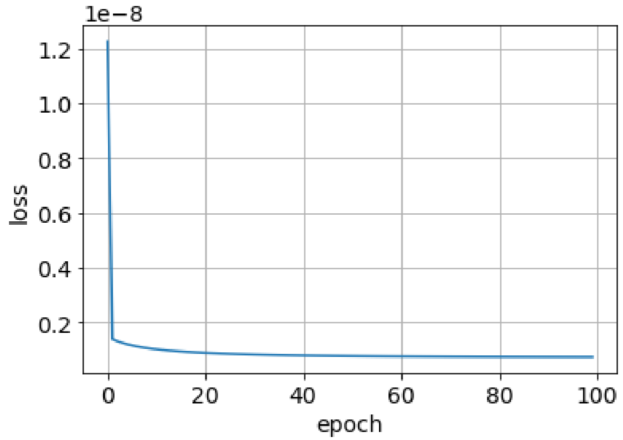
Output:
[114.94353 ,
72.04542 ,
16.565477]

Gaussian noise

time interval [0,0.2] U [1.4,1.7]

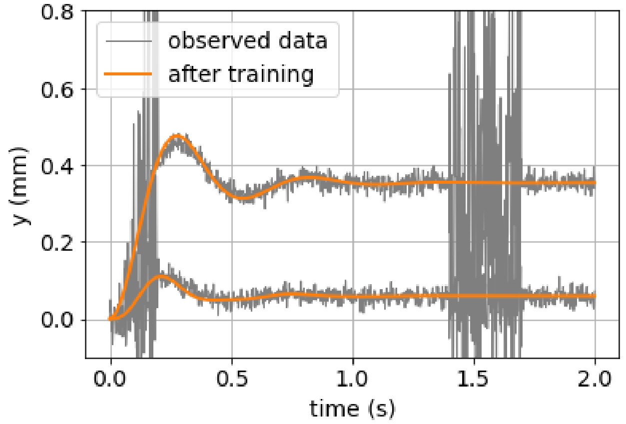
std=5

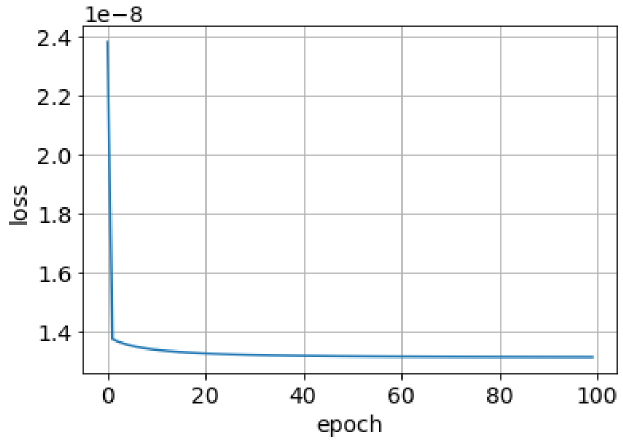




Output:
[117.8953 ,
72.30025,
17.36585]

std=100





Output:
[109.54126 ,
73.11582 ,
14.2527895]

RLC circuit

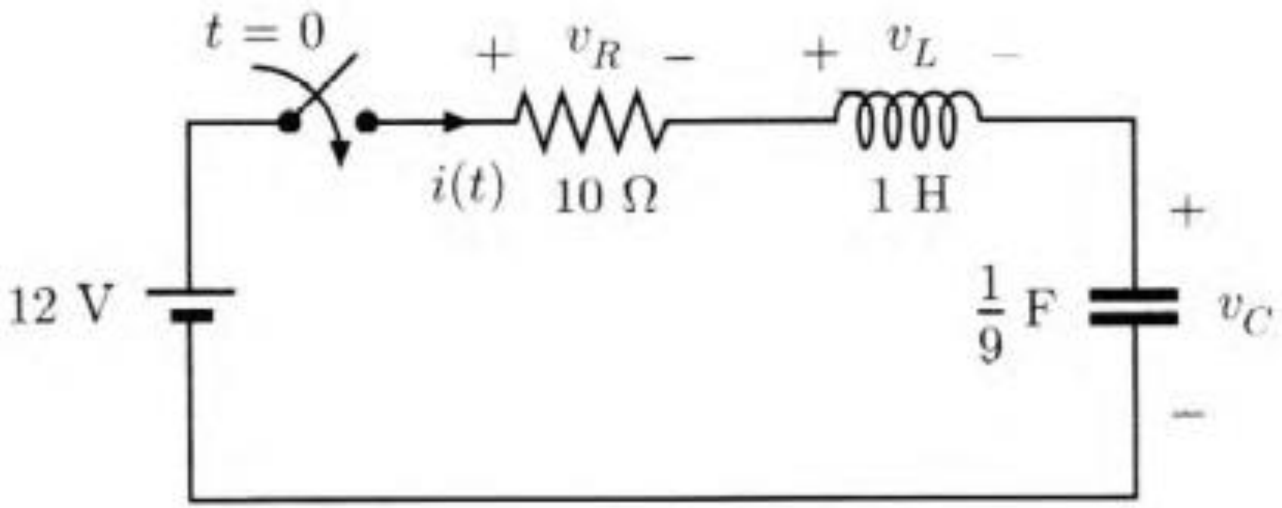
$$L\frac{d^2q}{dt^2}+R\frac{dq}{dt}+\frac{1}{C}q=E(t)$$

Goal: estimate the parameter R, resistance

Input: measurement of the tension

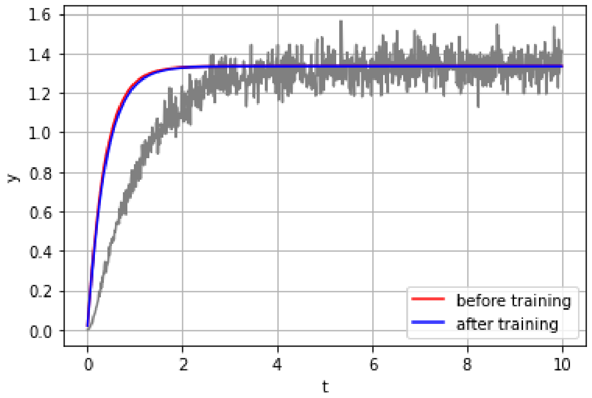
Output: measurement of the charge on the capcitor

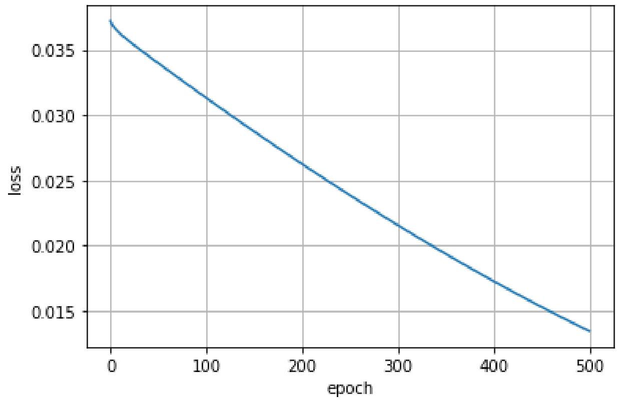
RLC circuit, constant V



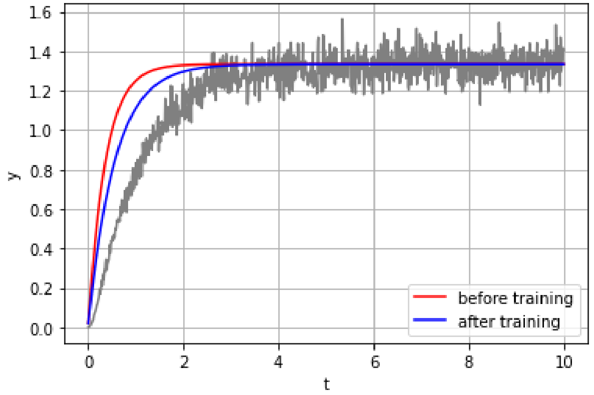
Issuses with learning rate and epochs

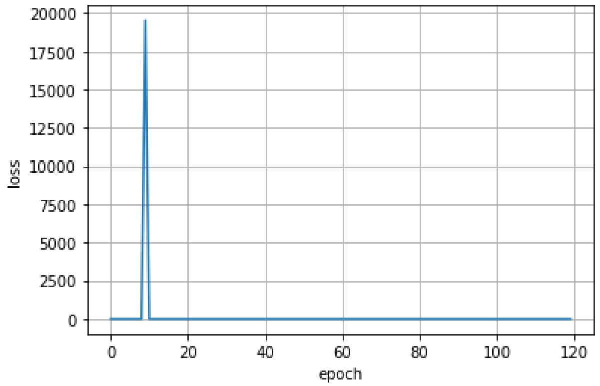
Learning rate=0.01
Epochs=500
Too small: the NN learns slowly





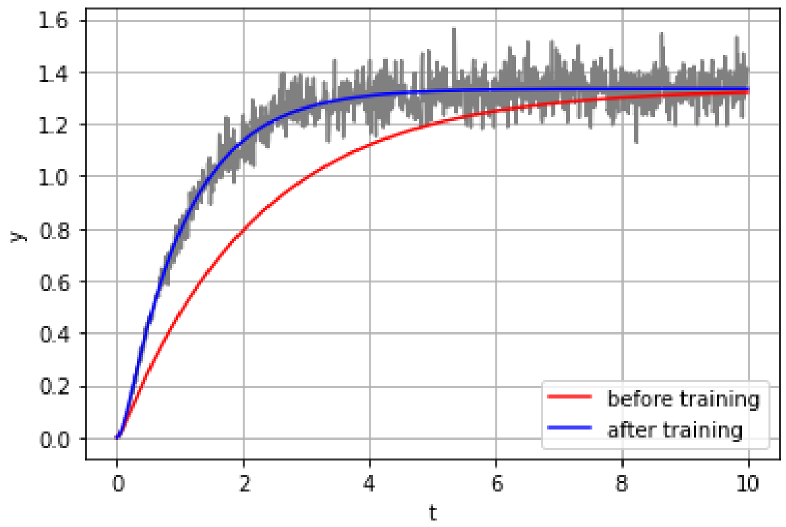
Learning rate= 1
Epochs=120
Too big: the NN doesn’t converge

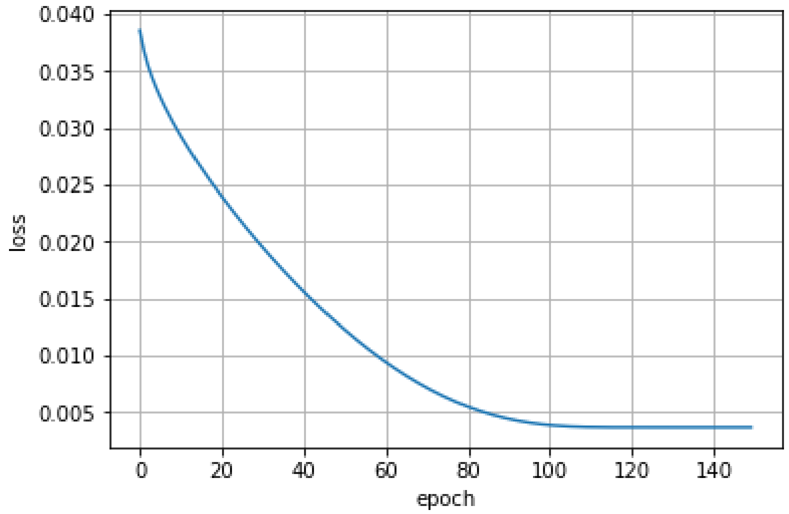




Optimal choice for NN

Learning rate= 0.1
Epochs= 120



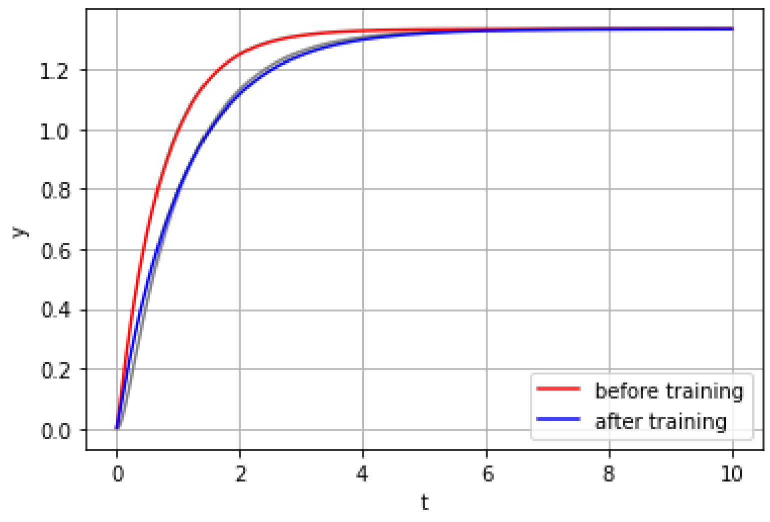


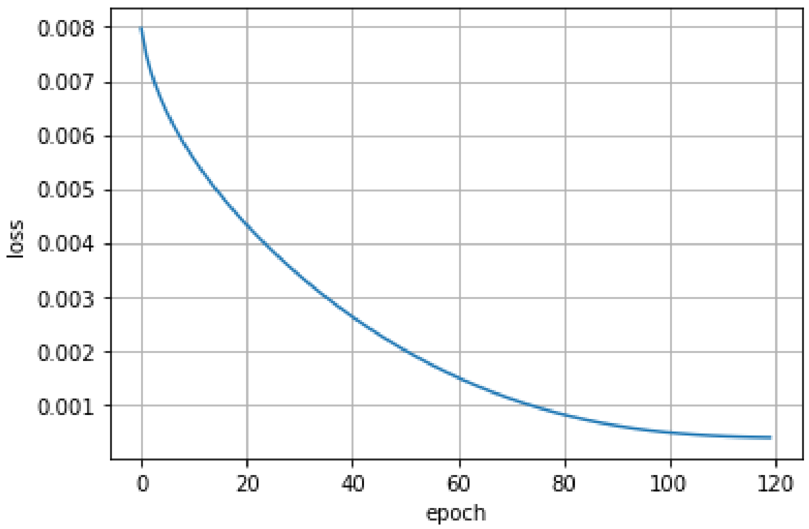
Output= [9.985753]

Limit of RK time step

We find the maximum step using the data without noise, the exact values.

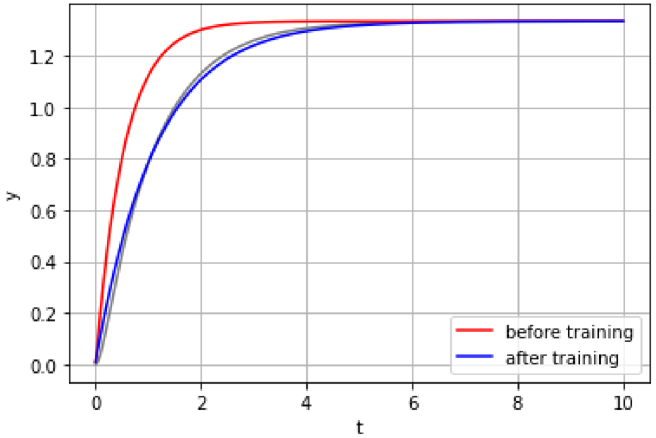
h=0.006

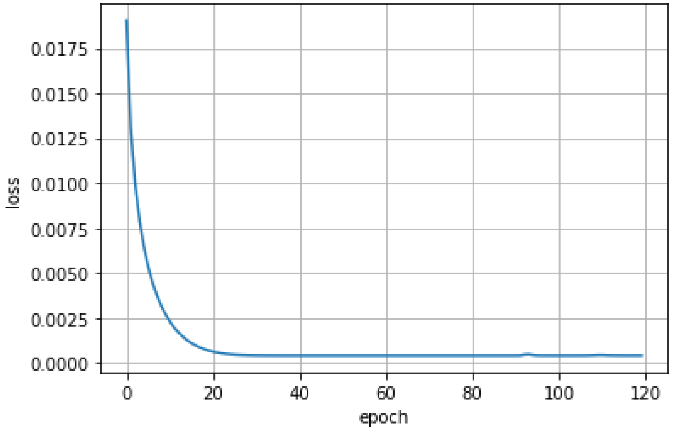




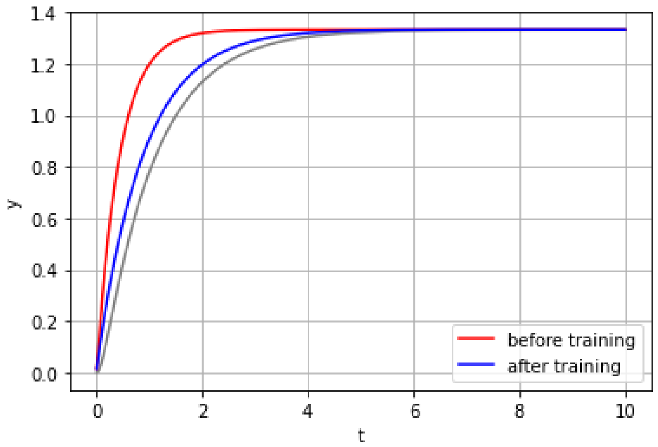
Limit RK time step

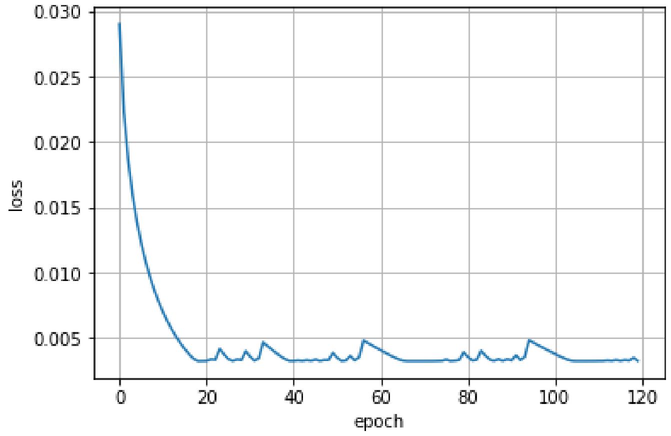
h= 0.008





h= 0.010

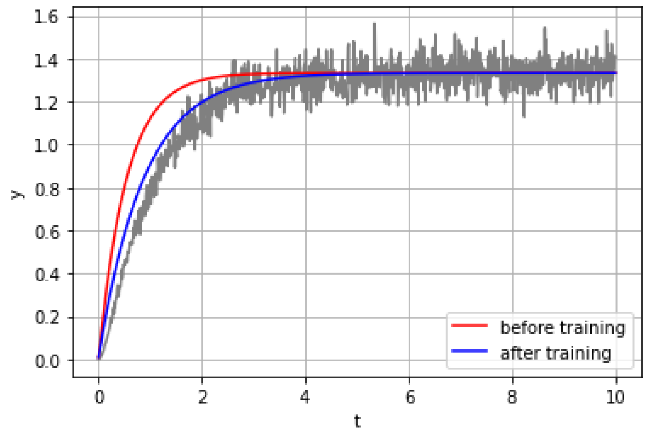


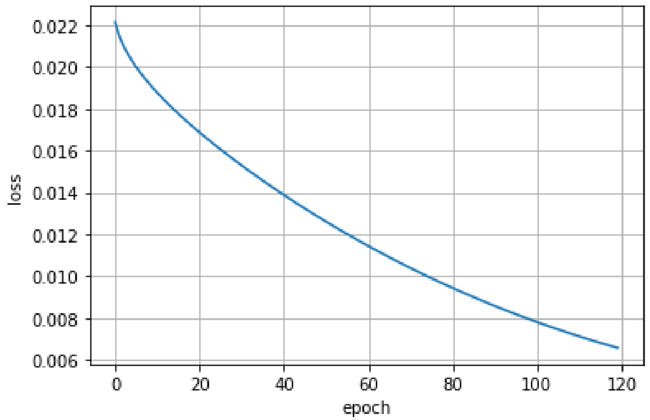


Theorecitlca vs Real

The theoretical limit is with h=0.08 but with the same h and the observed data we obtain

Output=[31.9423]





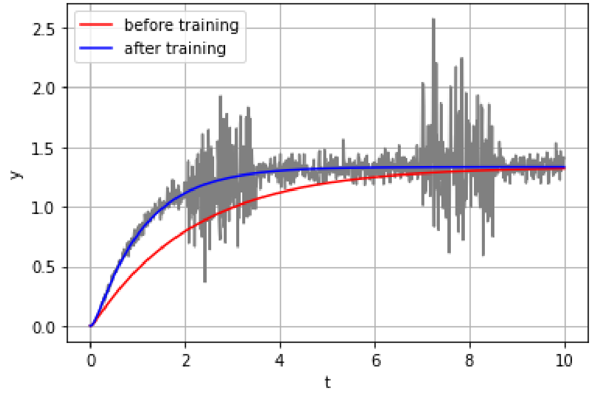
The estimation still bad with smaller h, it is good only with the minimum step (h=0.002).

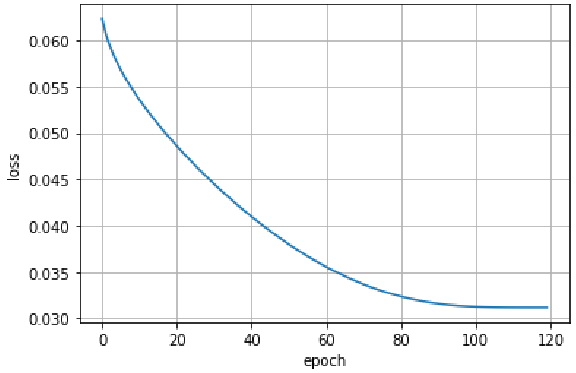
Noise

T=[2,3.5]U[7,8.5]

Std=50

Output=[10.420536]

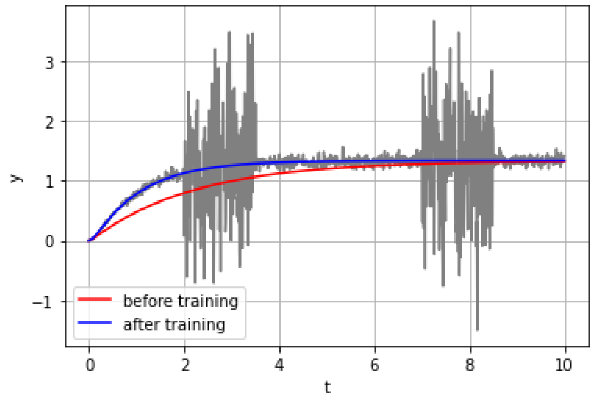


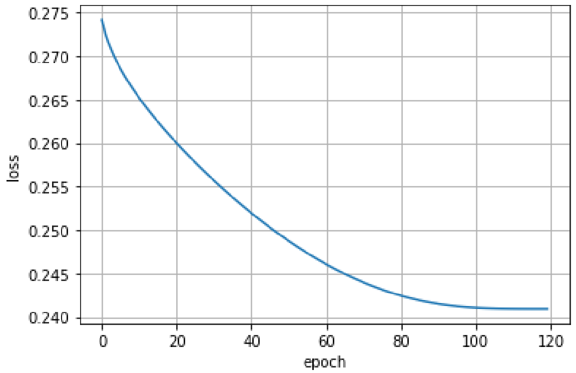


T=[2,3.5]U[7,8.5]

Std=100

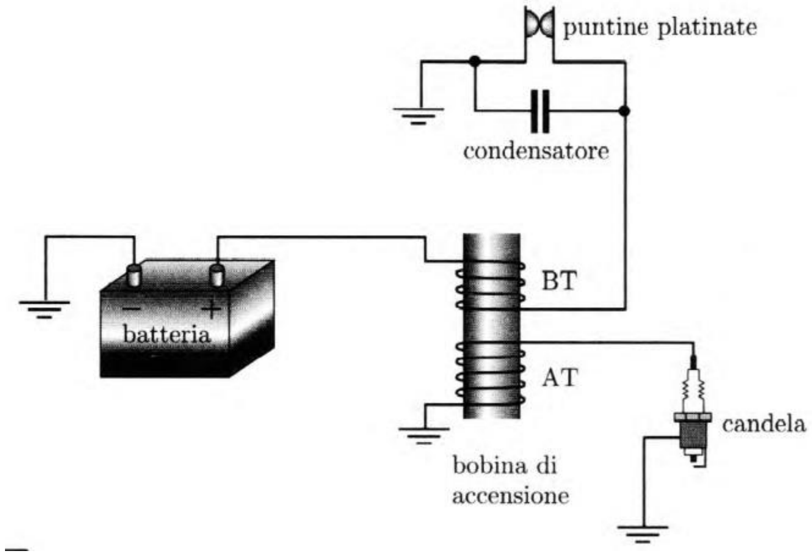
Output=[10.189874]



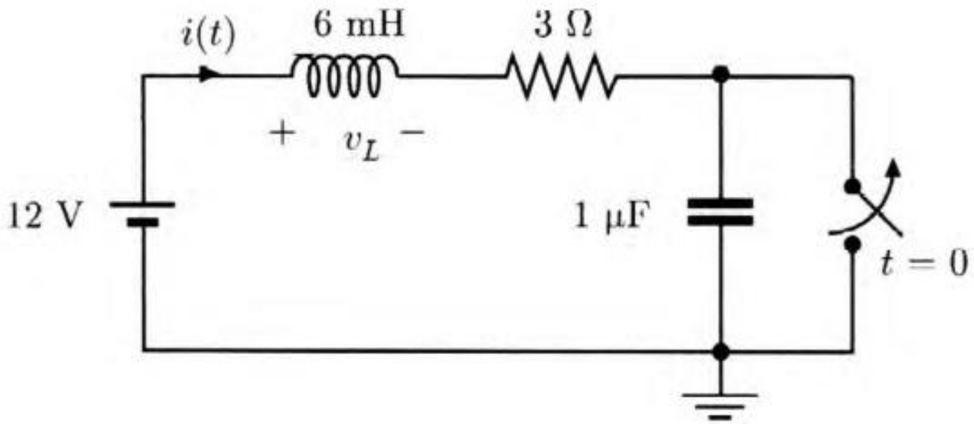


APPLICATION

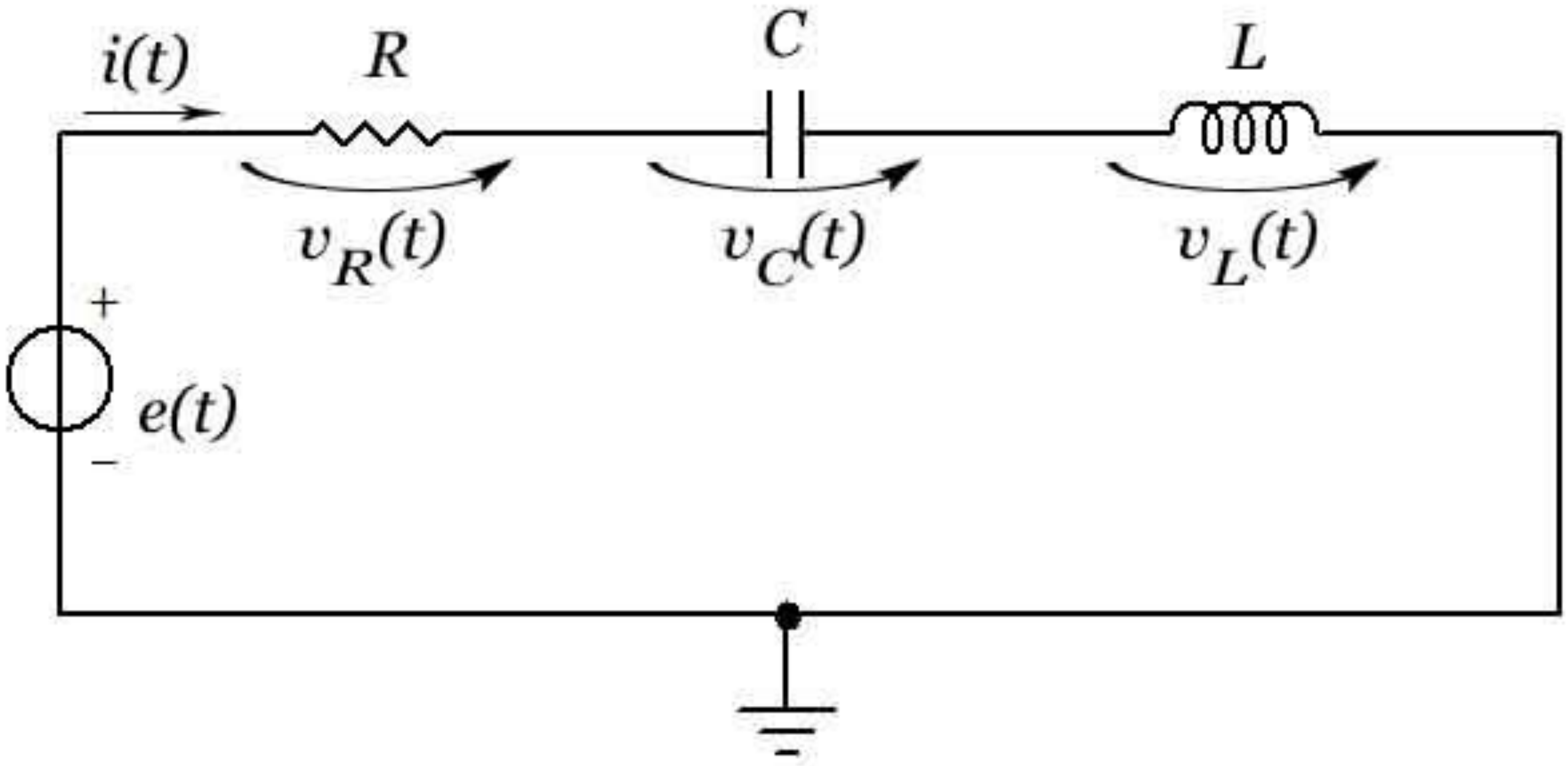
IGNITION CIRCUIT FOR A GASOLINE-POWERED ENGINE



→



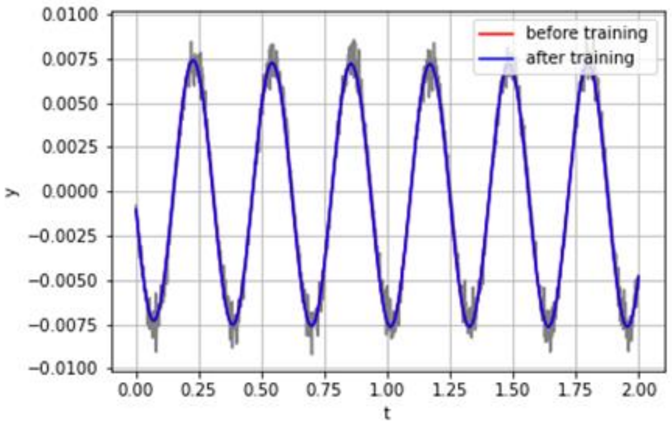
RLC circuit, sinusoidal input

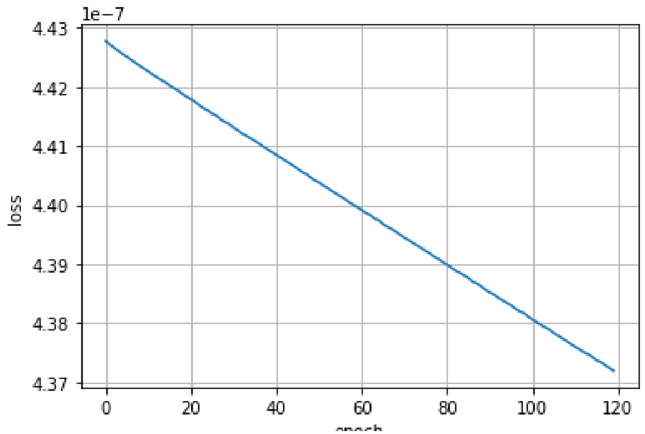


Neural Network parameters

Learning rate= 0.01

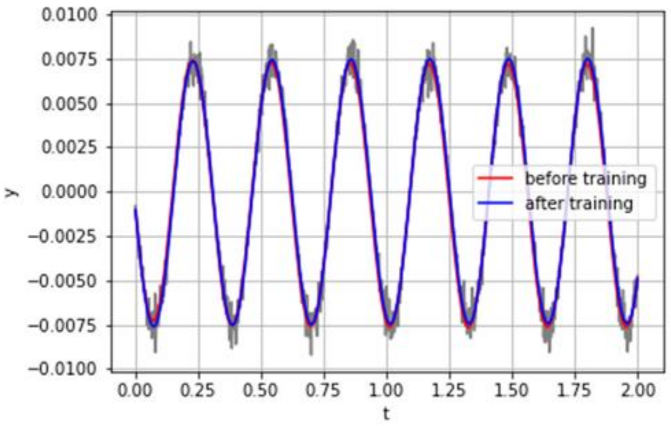
Epochs= 120

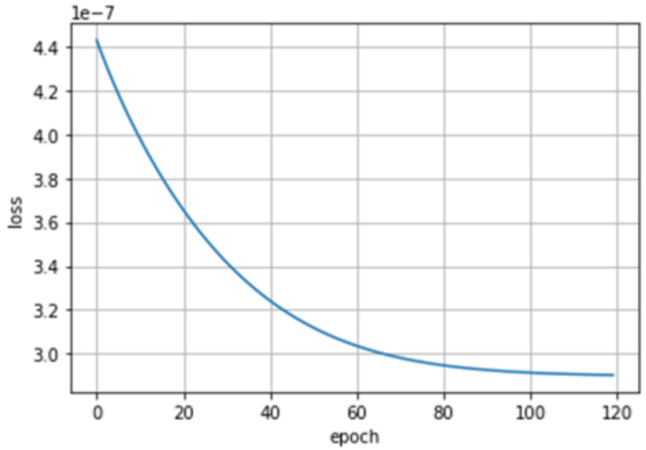




Learning rate= 1

Epochs= 120



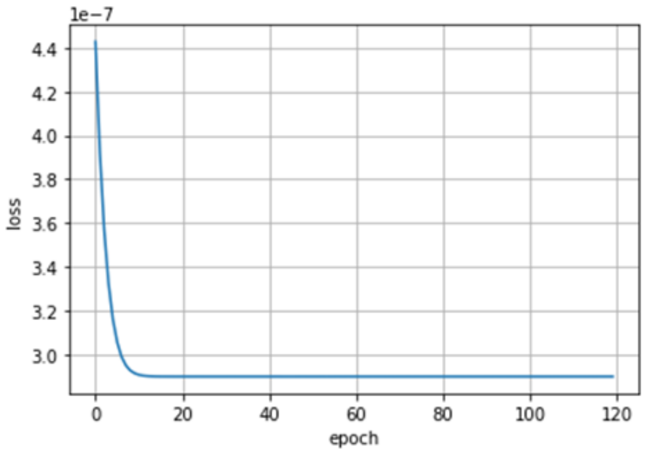


Neural Network parameters

Learning rate= 10

Epochs= 120

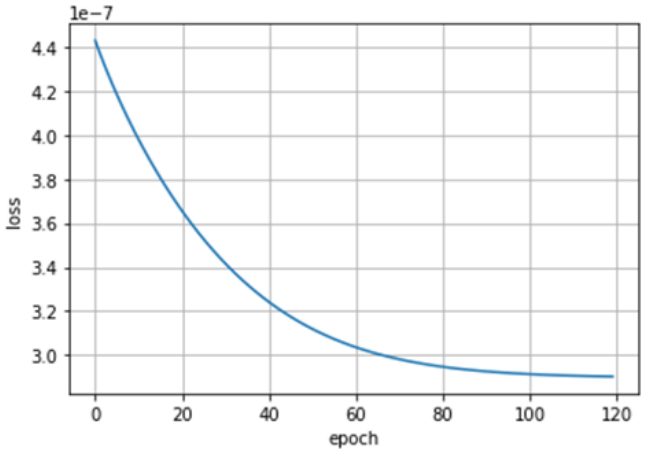
Output=[18.725863]



Learning rate= 1

Epochs= 120

Output=[18.769863]

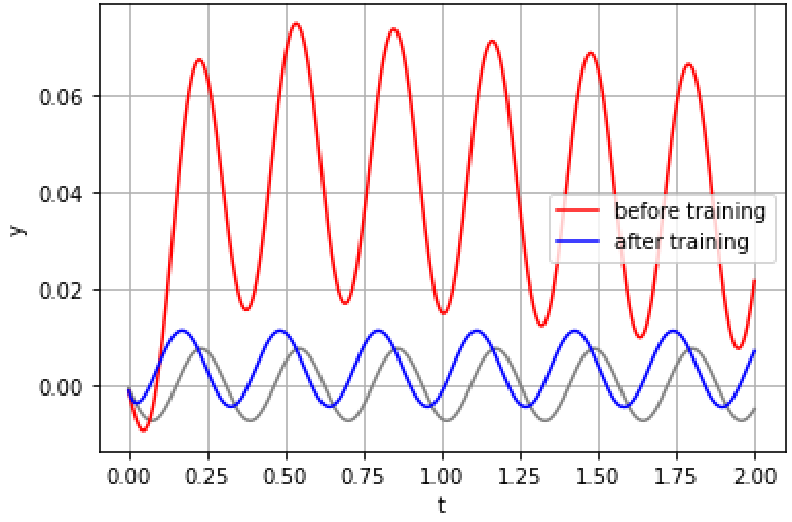


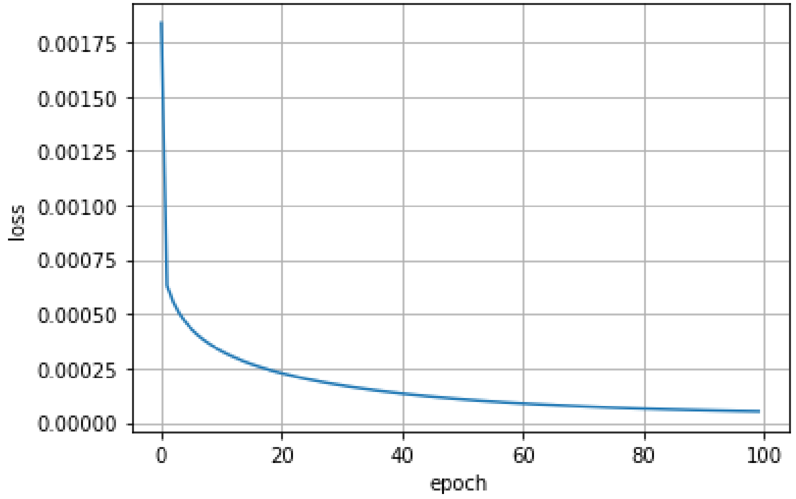
Perfect example of trade off:
We choosed as learning rate 10
Because we prefered using less epoch (100) over a more sensitive output

From now on we will use (10,100)

Different time step of RK

h = 0.004





The RK method does not converge for a bigger time step

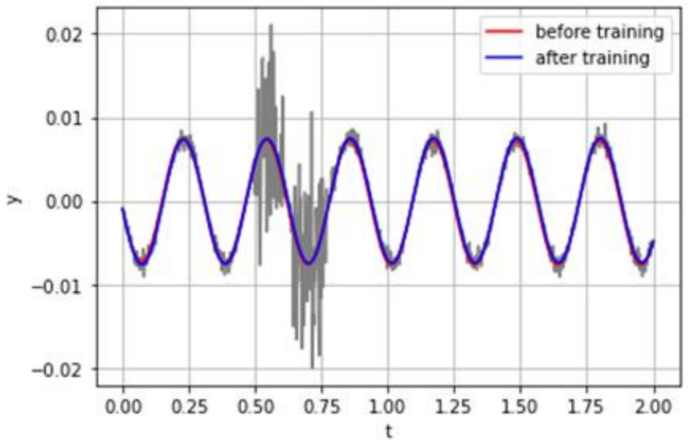
Also in this case to find the biggest time step, we used the data without noise

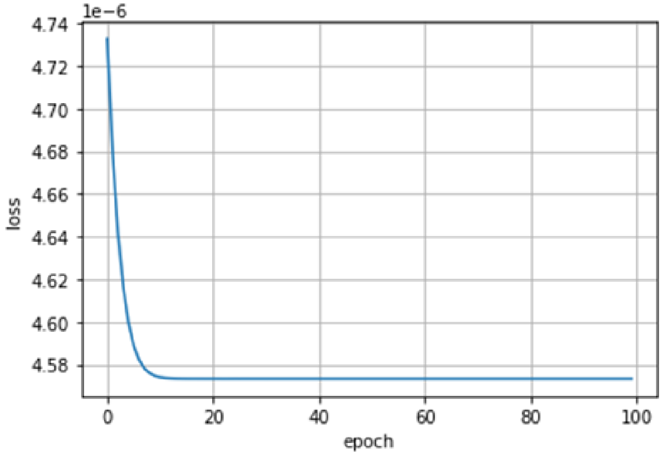
Gaussian Noise

Time interval=[0.5,0.8]

Std=100

Output=[18.589947]

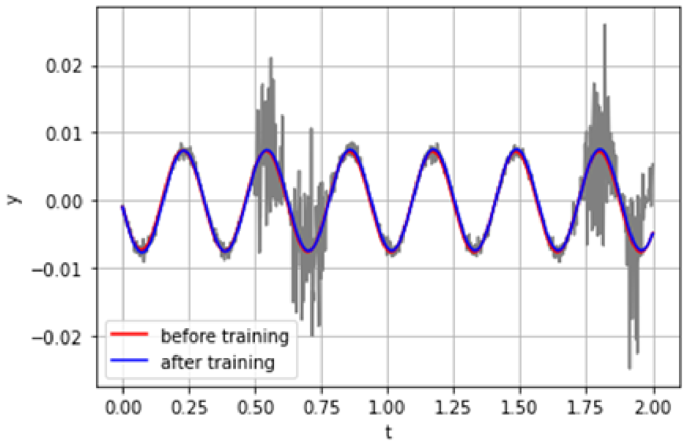


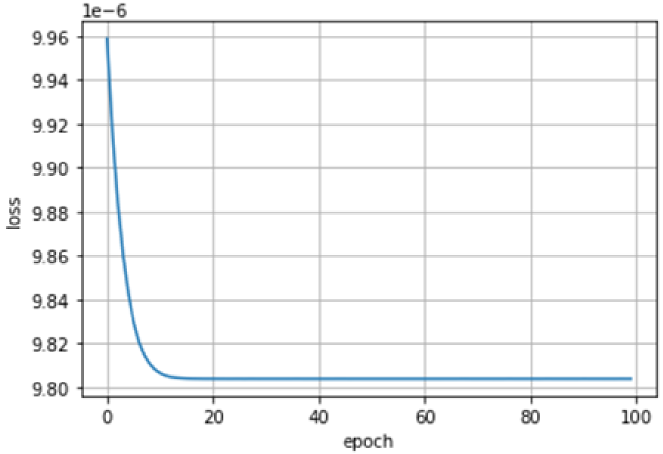


Time interval=[0.5,0.8]U[1.7,1.9]

Std=100

Output=[17.540703]

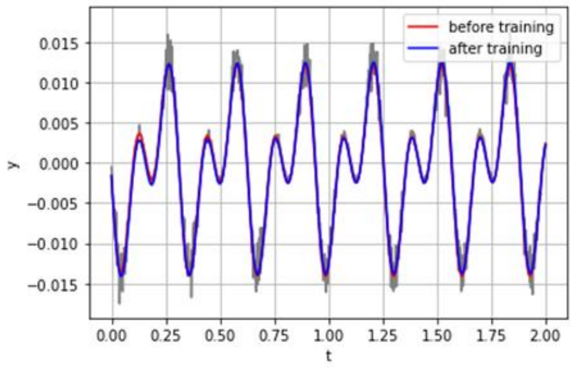


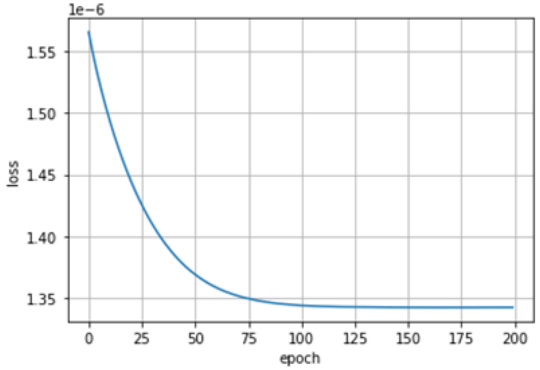


RLC sinusoidal with more input

We tried to see what happen if we use as an input the sum of some sine waves

2 inputs:
Learning rate=1
Epochs=200
Output=[17.065903]

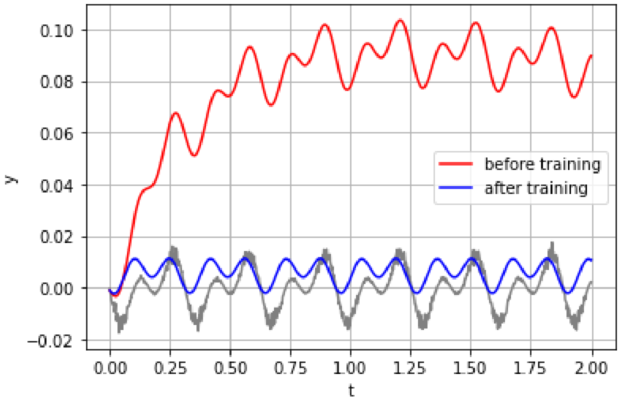


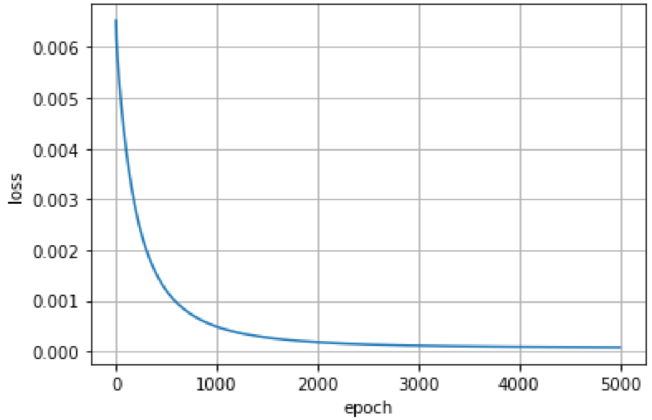


This is the best output that we found, and it's quite acceptable

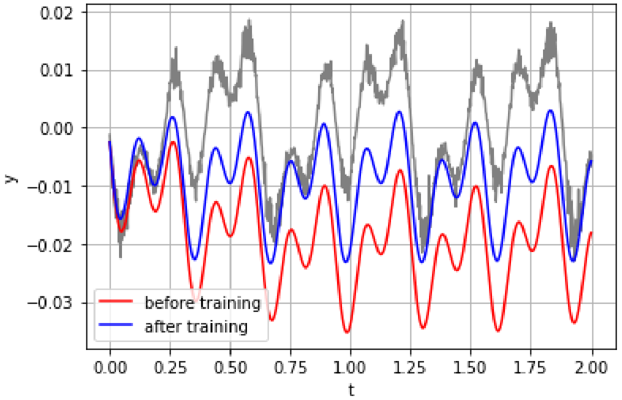
RLC sinusoidal with 3 inputs

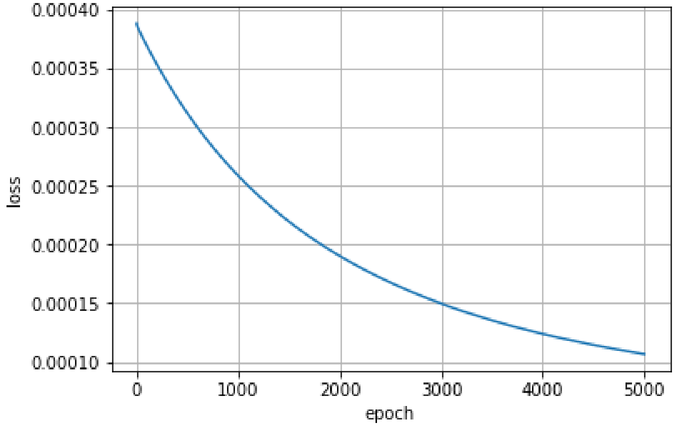
Learning rate= 0.01
Epochs=5000





Learning rate=0.1
Epochs=5000





Conclusion

- Data with a big noise are not a problem.

- The step for RK isn't flexible.
- The model doesn't work well for irregular input