Problem 1: Consider the data set:

x	1	2	3	4	5	6	7
y	1	2	1.5	2	3	2.5	4
y'	1	0	-0.1	0.4	0	-0.1	1.7

Determine an interpolant $p(x)$ such that $p(x_0) = y_0, p(x_1) = y_1, \dots, p(x_7) = y_7$.
(1) **(5 Points)** Find a polynomial interpolant as

$$\Pi_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 \qquad n \leq 6 \ .$$

Use **Lagrange polynomial** interpolant. Show your results in a plot and plot the fitted polynomial functions together with data points.

(2) **(5 Points)** Use **linear piecewise** interpolation. Show your results in a plot and plot the fitted polynomial functions together with data points.

(3) **(5 points)** Use **least squares** to fit a polynomial function $\Pi_n(x)$. Try both $n = 6$ and $n = 3$. Show your results in a plot and plot the fitted polynomial functions together with data points.

Problem 2: Approximate the function

$$f(x) = \frac{1}{1 + 36x^2}$$

in the interval $[-1, 1]$ through interpolation.
(1) Find a polynomial interpolant as

$$p(x) = \Pi_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 \ .$$

You can use Lagrange polynomial to construct the interpolant.

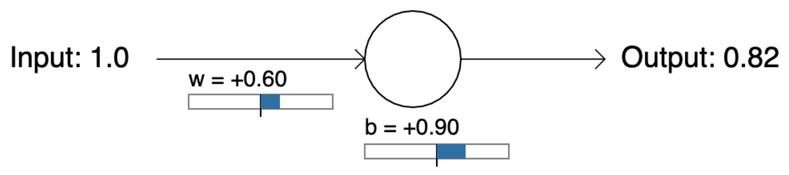
(1.a) Use equidistant data points; **(5 Points)**

(1.b) Read the note: "**Runge's phenomenon**" and use Chebyshev nodes. **(10 Points)**

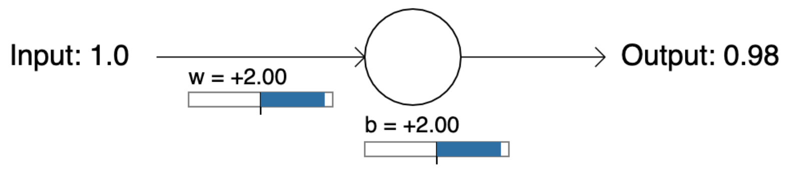
In both cases, when you increase n, does your interpolant approximate the true function better? Show your results in plots, and explain why.

(2) **(10 Points)** Use linear piecewise interpolation. When you increase n, does your interpolant approximate the true function better? Show your results in plots, and explain why.

Problem 3: **(20 Points)** Use **gradient descent** to learn the weight and bias for a sigmoid neuron such that the neuron takes the input 1 to the output 0.
1) Initialize the weight and bias as the following:



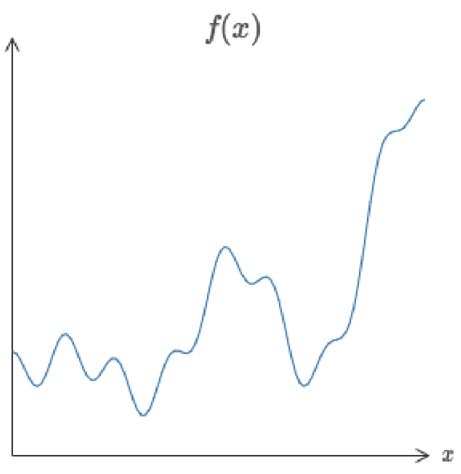
2) Initialize the weight and bias as the following:



For each case, learn up to 300 epochs. Plot the learning curve, i.e., the curve of cost vs. epoch. You need to determine an appropriate learning rate η.

Problem 4: **(40 Points)** Train a neural network so that for every possible input, $x \in [0, 1]$, the output from the network $g(x)$ is a close approximation for

$$f(x) = 0.2 + 0.4x^2 + 0.3x\sin(15x) + 0.05\cos(50x) \ .$$



You can use the python code (pytorch_regression.py) uploaded in Canvas as the starting point. And you need to install **PyTorch** from `https://pytorch.org/`.

Study the following aspects:

1. Determine an appropriate learning rate η;
2. Determine appropriate mini-batch size and number of epoch;
3. Compare different activation functions (sigmoid, Tanh, and ReLU);
4. Compare different network architectures, e.g., with the same total number of neurons but different number of hidden layers, or with fixed number of layers but different numbers of neurons.