# APMA 2070 & ENGN 2912V Deep Learning for Scientists and Engineers
# Homework 05
**Due Date: 04-05-2024, 11:59 pm (E.T.)**

## Lecture 1.5

The **Weierstrass theorem** states that *any continuous function defined in a finite range can be approximated to any degree of accuracy by polynomial powers.* This theorem does not state what is the appropriate type of polynomial approximation. The question which still remains to answer is whether equidistant data always produce stable and convergent polynomial approximations. O. Runge in 1901 demonstrated the non convergent polynomial approximation using equidistant points for the following function

$$f(x) = \frac{1}{1+25x^2}, \qquad x \in [-1,1] \tag{1}$$

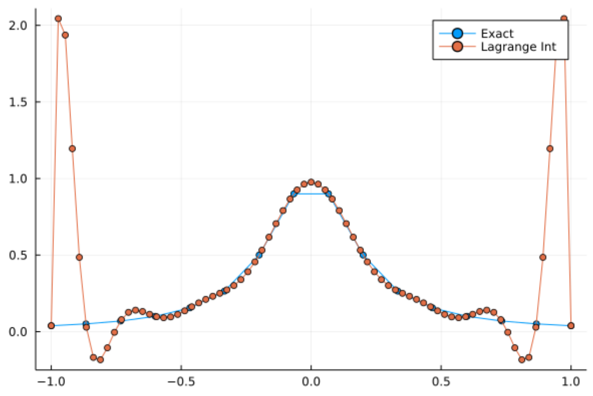

Figure 1: Interpolation of function in (1) using equidistant grid.

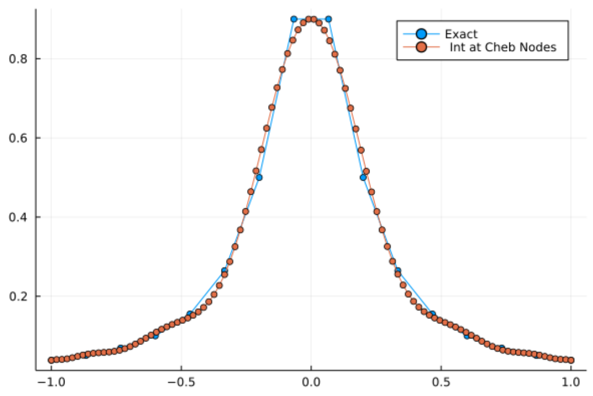

Figure 2: Interpolation of function in (1) using Chebyshev gridpoints.

## The Barycentric Lagrange Interpolation in 1D

Given grid $x_j$, we need to obtain polynomial $p(x_j)$ such that

$$p(x_j) = f(x_j) \qquad j = 0 \dots n$$

Solution to the above problem is expressed as Lagrange formulation

$$p(x_j) = \sum_{i=0}^{n} f(x_i) l_i(x),\ \ l_i(x) = \frac{\prod_{k=0,k\neq j}^{n}(x - x_k)}{\prod_{k=0,k\neq j}^{n}(x_i - x_k)} \tag{2}$$

This can be written simply by using the Barycentric Lagrange form

$$p(x_j) = \frac{\sum_{j=0}^{n} \frac{w_j}{x-x_j} f(x_j)}{\sum_{j=0}^{n} \frac{w_j}{x-x_j}}, \tag{3}$$

where the barycentric weights are

$$w_j = \frac{1}{\prod_{k\neq j}(x_j - x_k)}$$

For equidistant grid

$$w_j = (-1)^j \binom{n}{j}$$

Interpolation of (1) using equidistant grid is given in Figure 1. Interpolation of (1) using Chebyshev grids is given in Figure 2. The Chebyshev grid points are expressed as

$$x_j = \cos(j\pi/N).$$

## Function approximation using Fourier Series and Deep Neural Network

A periodic function $f(x)$ can be written as linear combination of cosine and sine of different frequency, which reads

$$f(x) = \frac{A_0}{2} + \sum_{k=1}^{\infty} A_k \cos(kx) + B_k \sin(kx), \tag{4}$$

where

$$A_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(kx)dx = \frac{1}{||\cos(kx)||^2}\langle f(x), \cos(kx)\rangle$$

$$B_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(kx)dx = \frac{1}{||\sin(kx)||^2}\langle f(x), \sin(kx)\rangle$$

Now, we will express (4) for any finite domain between $[0, L]$

$$f(x) = \sum_{k=0}^{\infty}\left(A_k \cos\left(\frac{2\pi kx}{L}\right) + B_k \sin\left(\frac{2\pi kx}{L}\right)\right) \tag{5}$$

where

$$A_k = \left\langle f(x), \cos\left(\frac{2\pi kx}{L}\right)\right\rangle$$

$$B_k = \left\langle f(x), \sin\left(\frac{2\pi kx}{L}\right)\right\rangle$$

**Example and implementation of function approximation using Fourier series**

Consider a hat function expressed as

$$f(x) = \begin{cases} 1 - |x|, & |x| < \frac{\pi}{2} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

The plot of (6) is shown in Figure 3. The approximation of (6) using (5) is shown in Figure 4.

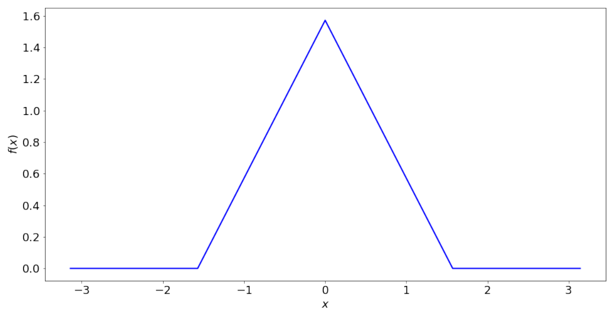

Figure 3: Hat function

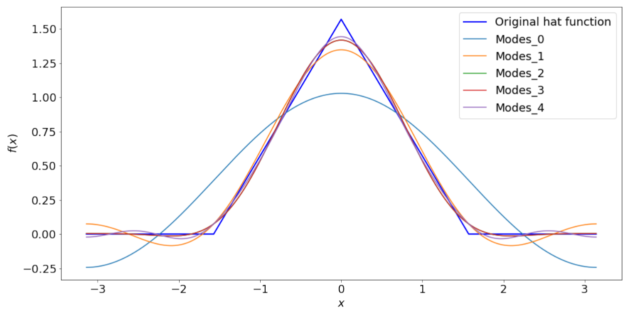

Figure 4: Approximation of (6) using 5 modes.

1. Approximate the function in (1) using a deep neural network.
2. Obtain results by sampling $x$ randomly, equidistant or Chebyshev distribution.
3. Implement your algorithms either in TensorFlow/PyTorch or JAX.
4. Approximate the function in (6) using a neural network.
5. Observe the accuracy of approximation of (6) in Figure 4 by increasing the number of Fourier modes.
6. Observe the accuracy of the approximation by varying the number of neurons and layers.
7. Succinctly provide your observations by establishing similarity and differences between Fourier series approximation and neural network. *Hint: Linear combination of basis functions.*
8. Comparing performance of Optimizers: Use various first and second order optimizers along with different combinations to test their performance for function approximation using neural networks. Use the following functions for testing.

   (i)

$$\begin{aligned} y &= 5 + \sum_{k=1}^{6} \sin(kx), \quad x < 0 \\ y &= \cos(10x), \ \ x \geq 0 \end{aligned} \tag{7}$$

   (ii)

$$y = \begin{cases} 0 & \text{if } t < 1 \\ 1 & \text{if } t \geq 1 \end{cases} \tag{8}$$

Hint: Sample code provided

- PyTorch based code for a simple function approximation *runge_ approx.py*.
- Python code *fourier_ approx.py* containing the code for Figure 4.
- Boiler plate code for function approximation using a DNN.
- Boiler plate code for plotting loss landscape has been provided *loss_ landscape.py* .