Implementation of Deep Galerkin Method (DGM) for 1D Heat Equation

Shadab Anwar, Shaikh

PhD Candidate
University of North Carolina at Charlotte

Bi-Weekly Research Meeting, December 2021

Table of Contents

1D Heat Equation

$$\frac{\partial u(t,x)}{\partial t} = \alpha \frac{\partial^2 u(t,x)}{\partial x^2}$$

where $\quad (t,x) \in [0,T] \times [0,L]$

$$u(0,x) = \sin(\frac{\pi x}{L}),$$

$$u(t,0) = u(t,L) = 0$$

where, $\alpha = \dfrac{K}{\rho C_p}$

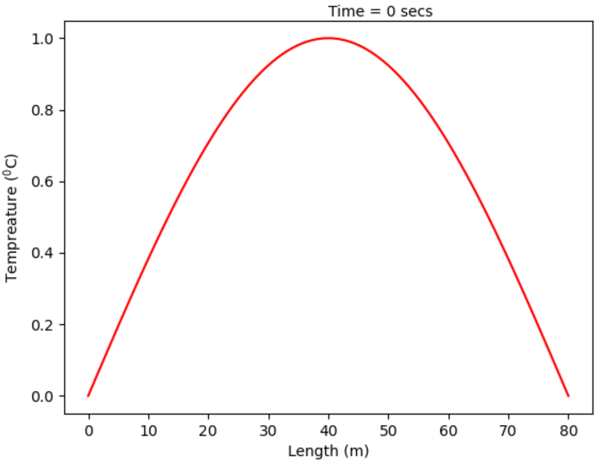
Analytical Solution

$$u(x,t) = \sin(\frac{\pi x}{L})\ e^{-0.001785t}$$

which was obtained by taking

$$\rho = 8.92 \text{ gram/cm}^3, C_p = 0.092 \text{ cal/g°C}, K = 0.95 \text{ cal/cm°C}$$

where $\quad (t,x) \in [0, 5 \text{ sec}] \times [0, 80 \text{ cm}]$


Time = 0 secs
1.0
0.8
0.6
0.4
0.2
0.0
Tempreature (0C)
0
10
20
30
40
50
60
70
80
Length (m)

DGM Algorithm

- Generate random points (t_n, x_n) , (t_i, x_i), (t_0, x_0) and (t_L, x_L) from $[0, T] \times [0, L]$, $[0, 0] \times [0, L]$, $[0, T] \times [0, 0]$ and $[0, T] \times [L, L]$ respectively.
- Calculate the squared error $G(\theta_n, s_n)$ at the randomly sampled points $s_n = \{(t_n, x_n), (t_i, x_i), (t_0, x_0), (t_L, x_L)\}$ where,

$$G(\theta_n, s_n) = \left(\frac{\partial f(t_n, x_n; \theta_n)}{\partial t} - \alpha \frac{\partial^2 f(t_n, x_n; \theta_n)}{\partial x^2} \right)^2 +$$

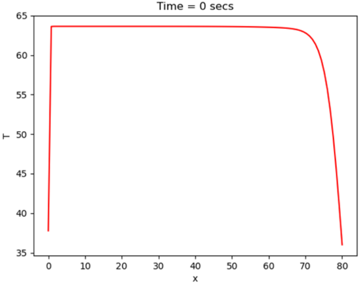
$$(f(t_0, x_0; \theta_n) - u(t_0, x_0))^2 + (f(t_L, x_L; \theta_n) - u(t_L, x_L))^2 +$$
$$(f(t_i, x_i; \theta_n) - u(t_i, x_i))^2$$

- Take a descent step at random point s_n

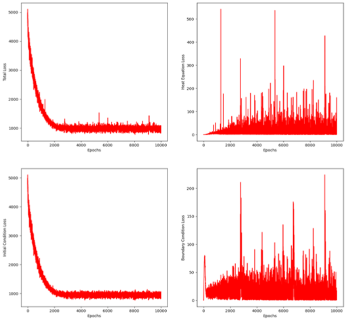
$$\theta_{n+1} = \theta_n - \alpha_n \nabla_\theta \, G(\theta_n, s_n)$$

- Repeat until convergence criteria is satisfied.

Results from last meeting



(a) Prediction



(b) Losses

Figure: 1

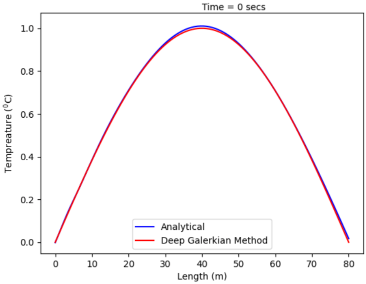
Neural Network Parameters

- Number of trainable parameters : 921
- Batch Size : 500
- Learning Rate : 0.0001 (Adam Optimizer)
- Total Epochs : 10000
- Shape of Neural Network : [2,20,20,20,1]
- Activation: tanh
- Training time: 4 - 6 hrs (approx.)

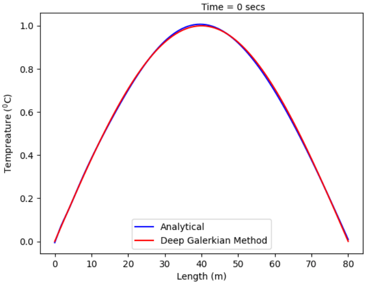
Important

I apologize for the typos in plots. It should be "Galerkin" instead of "Galerkian" and legends should be flipped.

- Selecting random points from



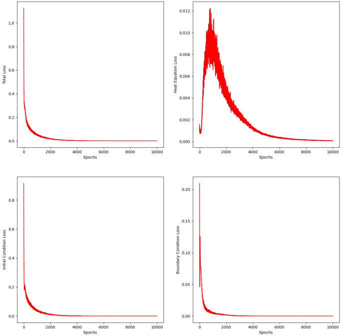
(a) Normal Distribution



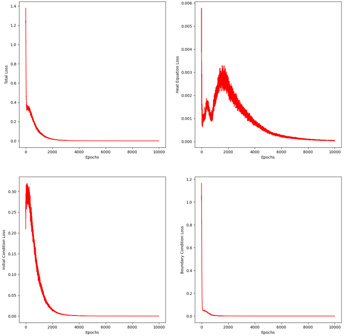
(b) Uniform Distribution

Figure: 2

- Corresponding losses



(a) Normal Distribution

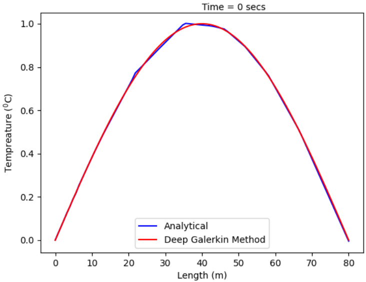


(b) Uniform Distribution

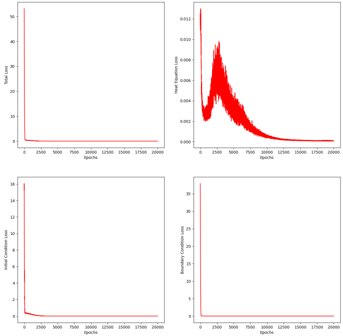
Figure: 3

Experiment 02 (Finding the activation)

- ReLU



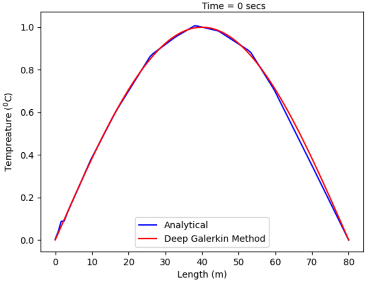
(a) Solution



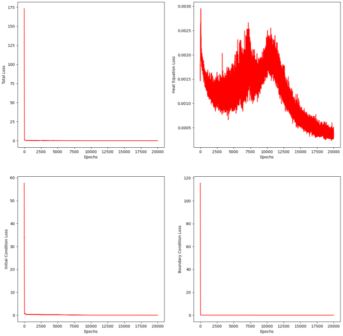
(b) Losses

Figure: 4

- Leaky ReLU



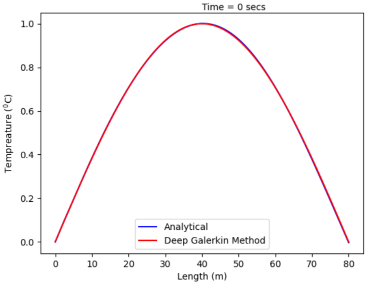
(a) Solution



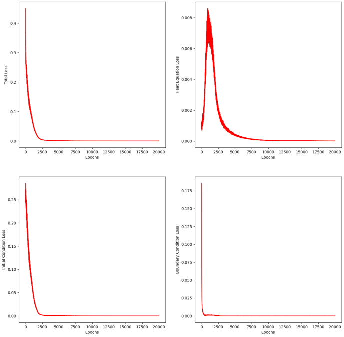
(b) Losses

Figure: 5

- tanh (Extended Domain, Increased epochs)



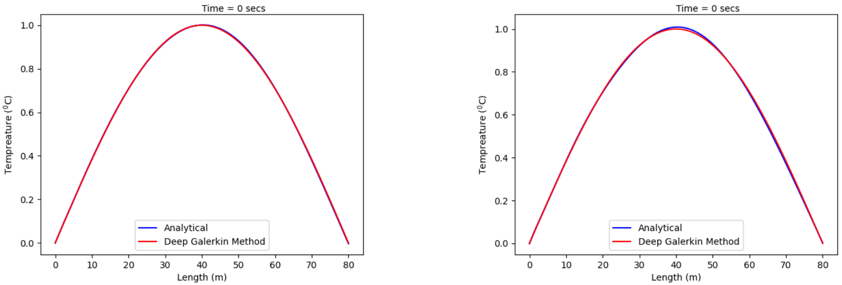
(a) Solution



(b) Losses

Figure: 6

Experiment 03 (Finding the good architecture)



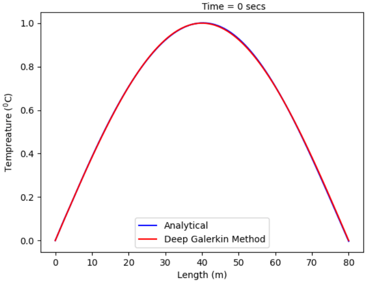
(a) S:(2,20,20,1), P:501

(b) S:(2,20,50,20,1), P:2151

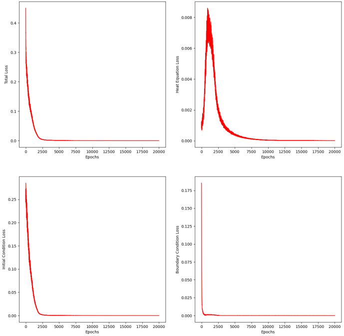
Figure: 7

Best Results

- Number of trainable parameters : 921
- Batch Size : 500
- Learning Rate : 0.0001 (Adam Optimizer)
- Total Epochs : 20000
- Shape of Neural Network : [2,20,20,20,1]



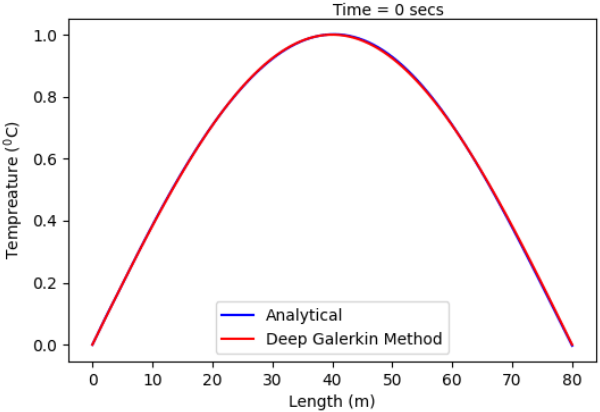
(a) Solution



(b) Losses

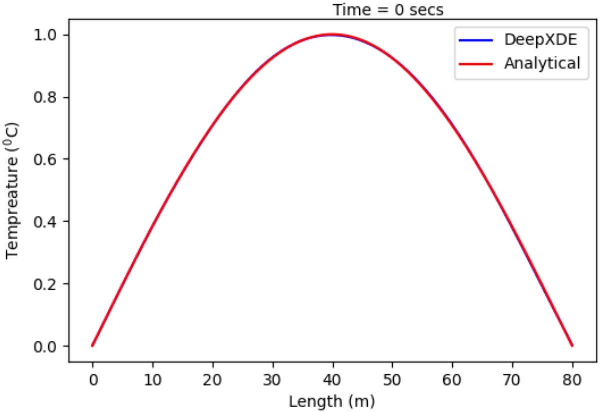
Figure: 8

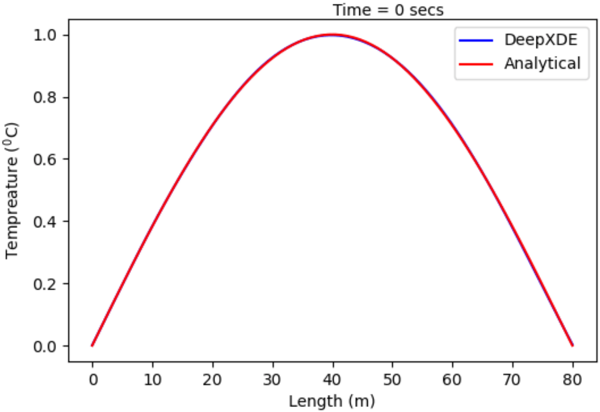
But not good in generalization, and slow in training



Results using DeepXDE

- Activation: tanh
- Learning Rate : 0.001 (Adam Optimizer)
- Total Epochs : 20000
- Shape of Neural Network : [2,20,20,20,1]


Time = 0 secs
DeepXDE
Analytical
Tempreature (^{0}C)
Length (m)
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
70
80



Time = 0 secs
DeepXDE
Analytical
Trempreature (^{0}C)
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
70
80
Length (m)

Sirignano, Justin and Spiliopoulos, Konstantinos
DGM: A deep learning algorithm for solving partial differential equations
Journal of computational physics, 375, 1339–1364, 2018.

Al-Aradi, Ali and Correia, Adolfo and Naiff, Danilo and Jardim, Gabriel and Saporito, Yuri
Solving nonlinear and high-dimensional partial differential equations via deep learning
preprint arXiv:1811.08782, 2018.

https://github.com/alialaradi/DeepGalerkinMethod
.

https://github.com/pooyasf/DGM
.