SfS^2 (SECURE for Student Success)
First Annual Undergraduate Research Symposium

SfS^2
SECURE for Student Success

Orchard Conference Center
September 11, 2024

SfS2
SECURE for Student Success

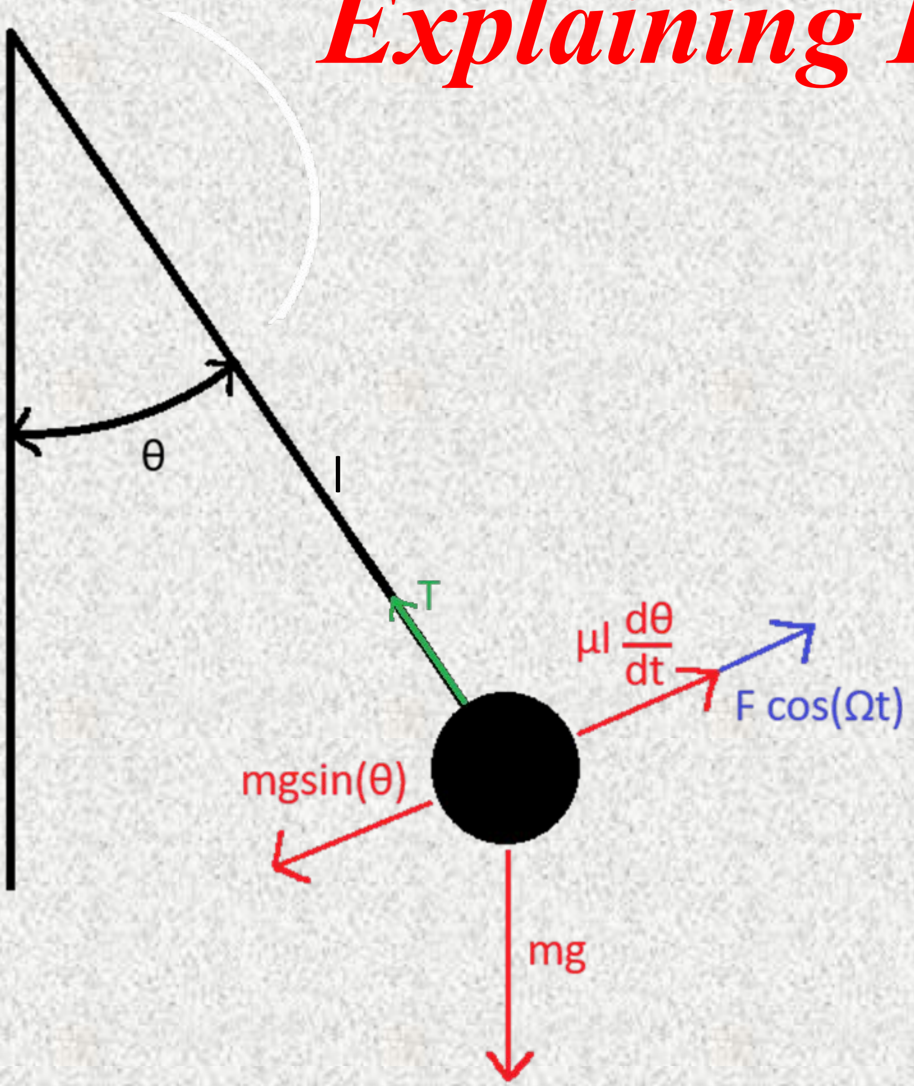
Solving Physics Equations with Neural Networks

Brandon Ismalej[1]
Davit Gasparyan[1]
Stepan Karapetyan[1]
Faculty Advisor: Dr. Yang Peng[2]

Department of Computer Science[1]
Department of Physics & Astronomy[2]

SfS[2]
SECURE for Student Success

Explaining Physics of the Pendulum



$$\frac{d^2\theta}{dt^2} + \lambda\frac{d\theta}{dt} + \omega^2\sin\theta - f\cos\Omega t = 0$$

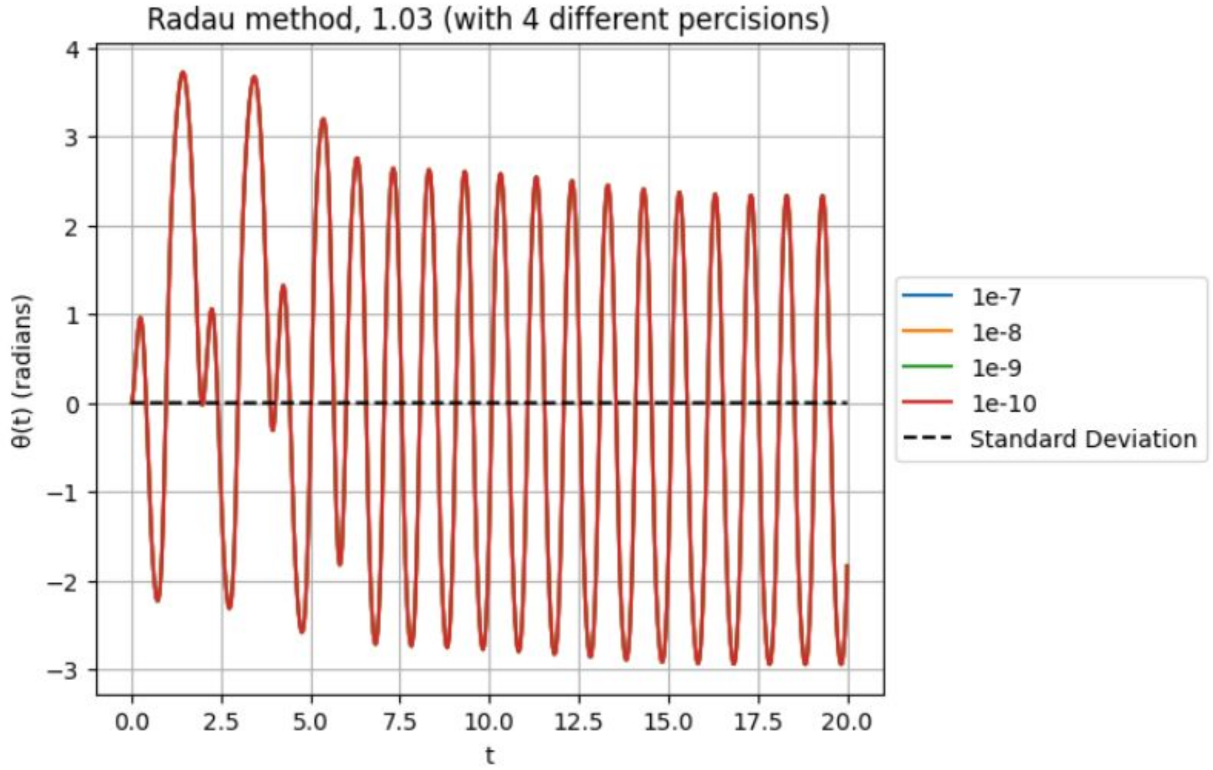
$$\text{where } \lambda = \mu/m,\ f = \frac{F}{ml} = \gamma\omega^2, \text{ and } \omega^2 = g/l.$$

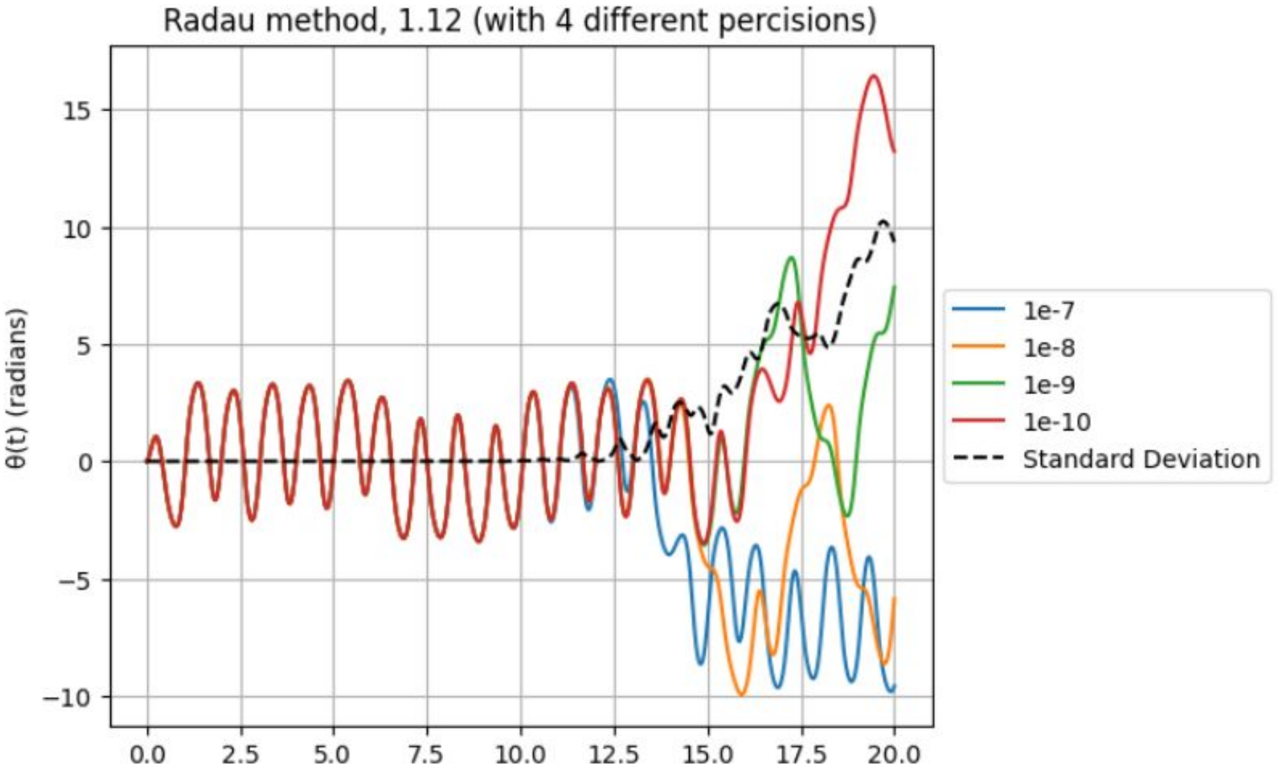
Equation 1: Damped Driven Pendulum

γ is what introduces chaotic behavior

Gamma Values

- Gamma values between 0 - 1.045 behavior are predictable
- Between 1.046 - 1.258, behavior is chaotic (some are predictable)





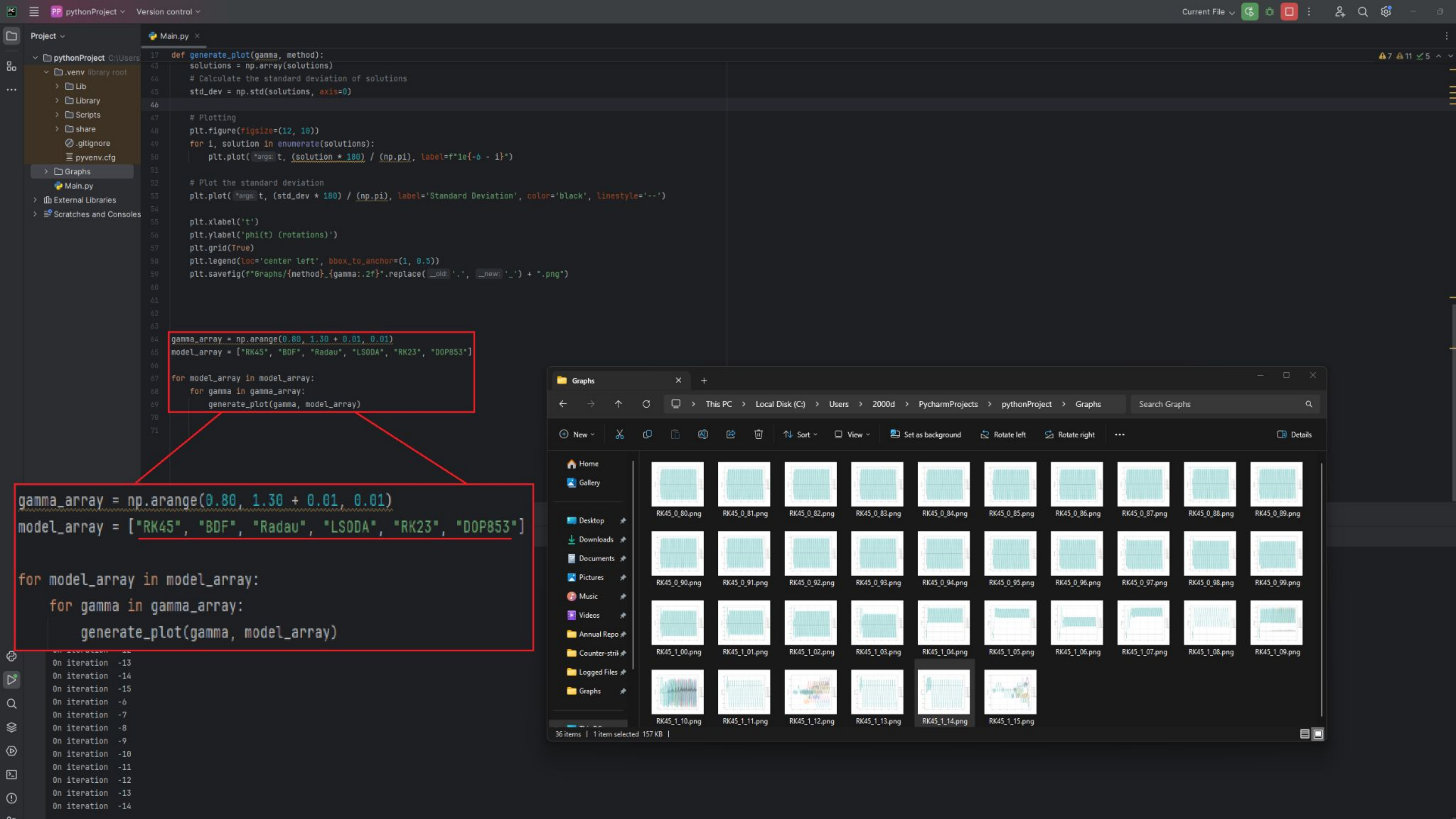
Objectives

To approximate solution using:

- Numerical Solver
- Multilayer Perceptron
- Kolmogorov-Arnold Network
- Comparison

Numerical Solution

- **Numerical Solution:** Approximation using step-by-step calculations of Eq. 1.
- **Methods:** Euler's method, Runge-Kutta, etc., that estimate the solution at each step.
- **Goal:** Get close to the true solution using small, manageable calculations.


pythonProject
Version control
Current File
Project
Main.py
pythonProject C:\Users
.venv library root
Lib
Library
Scripts
share
.gitignore
pyvenv.cfg
Graphs
Main.py
External Libraries
Scratches and Consoles
def generate_plot(gamma, method):
solutions = np.array(solutions)
# Calculate the standard deviation of solutions
std_dev = np.std(solutions, axis=0)
# Plotting
plt.figure(figsize=(12, 10))
for i, solution in enumerate(solutions):
plt.plot(*args: t, (solution * 180) / (np.pi), label=f"1e{-6 - i}")
# Plot the standard deviation
plt.plot(*args: t, (std_dev * 180) / (np.pi), label='Standard Deviation', color='black', linestyle='--')
plt.xlabel('t')
plt.ylabel('phi(t) (rotations)')
plt.grid(True)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.savefig(f"Graphs/{method}_{gamma:.2f}".replace(_old: '.', _new: '_') + ".png")
gamma_array = np.arange(0.80, 1.30 + 0.01, 0.01)
model_array = ["RK45", "BDF", "Radau", "LSODA", "RK23", "DOP853"]
for model_array in model_array:
for gamma in gamma_array:
generate_plot(gamma, model_array)
On iteration -13
On iteration -14
On iteration -15
On iteration -6
On iteration -7
On iteration -8
On iteration -9
On iteration -10
On iteration -11
On iteration -12
On iteration -13
On iteration -14
Graphs
This PC > Local Disk (C:) > Users > 2000d > PycharmProjects > pythonProject > Graphs
Search Graphs
New
Sort
View
Set as background
Rotate left
Rotate right
Details
Home
Gallery
Desktop
Downloads
Documents
Pictures
Music
Videos
Annual Repo
Counter-stri
Logged Files
Graphs
RK45_0_80.png
RK45_0_81.png
RK45_0_82.png
RK45_0_83.png
RK45_0_84.png
RK45_0_85.png
RK45_0_86.png
RK45_0_87.png
RK45_0_88.png
RK45_0_89.png
RK45_0_90.png
RK45_0_91.png
RK45_0_92.png
RK45_0_93.png
RK45_0_94.png
RK45_0_95.png
RK45_0_96.png
RK45_0_97.png
RK45_0_98.png
RK45_0_99.png
RK45_1_00.png
RK45_1_01.png
RK45_1_02.png
RK45_1_03.png
RK45_1_04.png
RK45_1_05.png
RK45_1_06.png
RK45_1_07.png
RK45_1_08.png
RK45_1_09.png
RK45_1_10.png
RK45_1_11.png
RK45_1_12.png
RK45_1_13.png
RK45_1_14.png
RK45_1_15.png
36 items | 1 item selected 157 KB

Numerical Solution's Method

- **Radau Method:** Type of implicit Runge-Kutta method specifically designed for stiff ODEs.

- **Why Radau?**
  - High Accuracy: Provides very accurate solutions, especially for challenging problems.
  - Long-Term Stability: Maintains accuracy over long periods, crucial for complex or sensitive systems.

SfS²
SECURE for Student Success

Objectives

To approximate solution using:

- Numerical Solver
- Multilayer Perceptron
- Kolmogorov-Arnold Network
- Comparison

One Neuron

A Neuron Contains:

- Input
- Weights
- Bias
- Activation Function
- Output

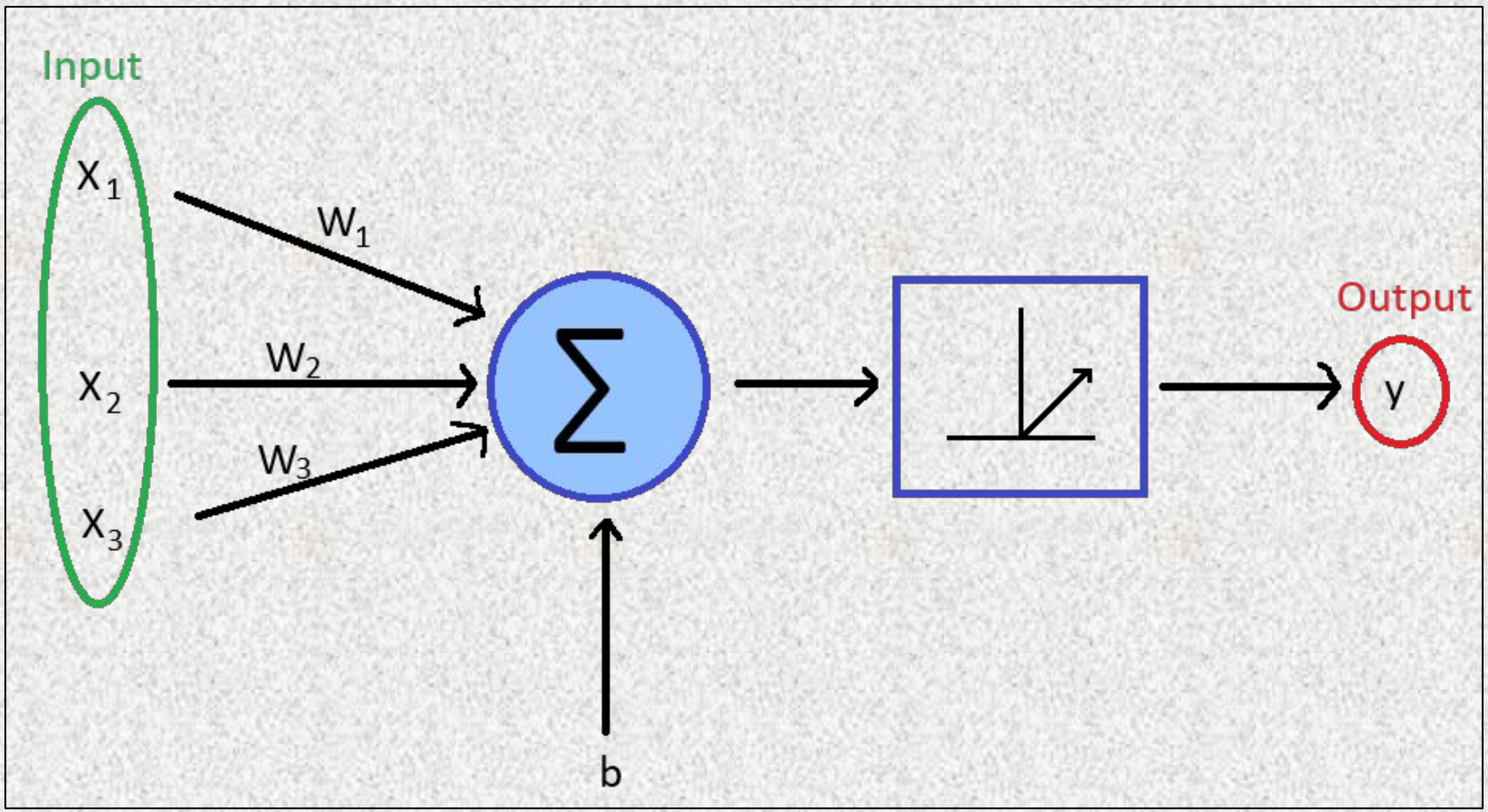


Figure: Image of Single Neuron

SfS2
SECURE for Student Success

Multilayer Perceptron (MLP)

- **Universal Approximation Theorem:**
  - A single hidden-layer MLP can approximate any continuous function on a limited domain to any desired accuracy with enough neurons
- **Architecture:**
  - Input Layer
  - Hidden Layers
  - Output Layer
- **Why MLP?**
  - Simple and extensively studied

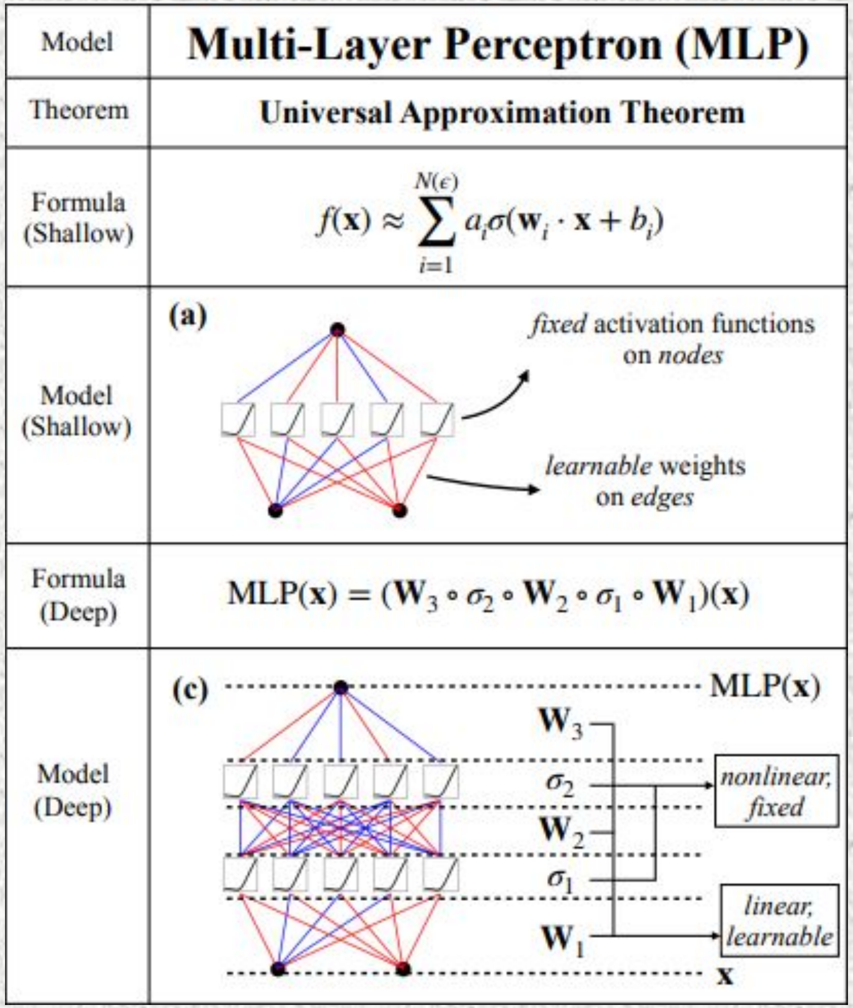
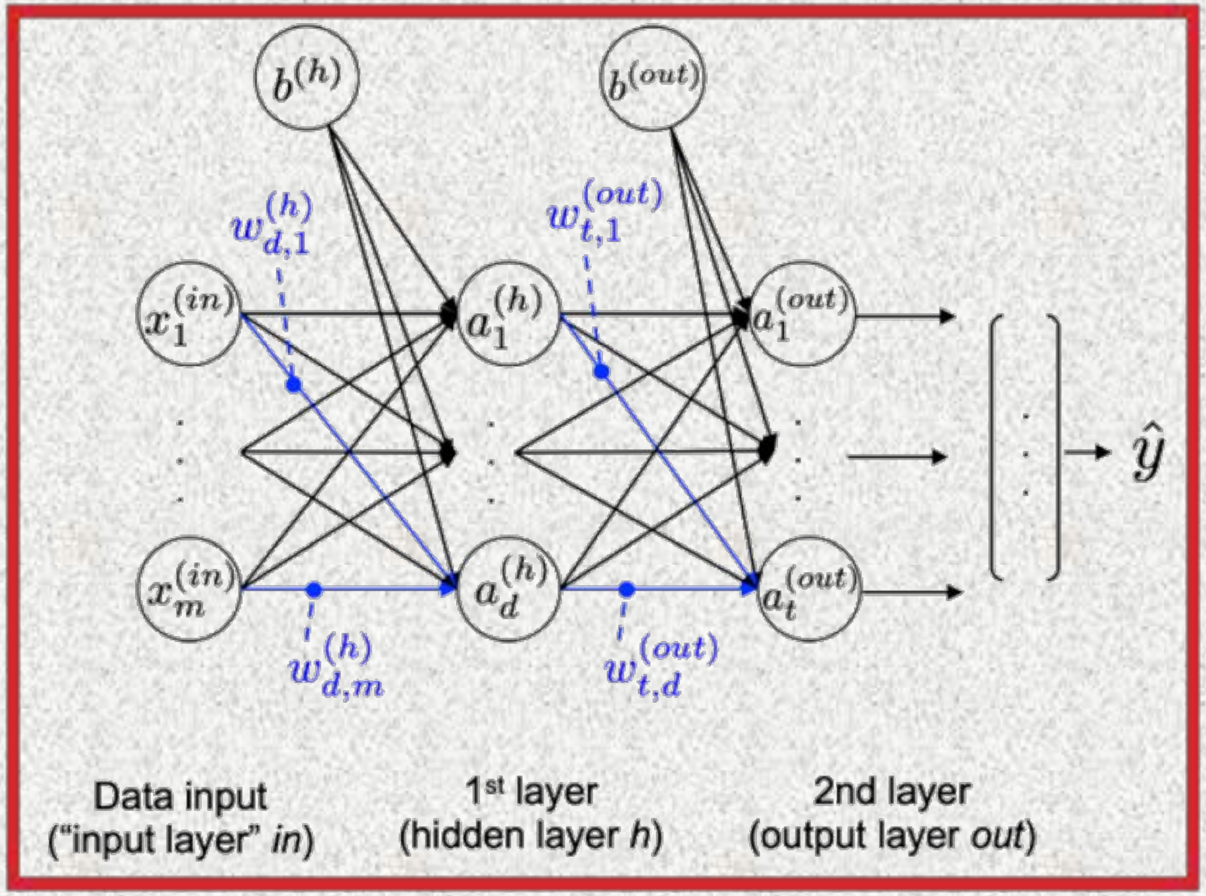


Figure: MLP Architecture [1]

Training MLP

F_{nn}



(input) $X = \boxed{\begin{matrix} X_1 \\ X_2 \\ X_3 \\ \vdots \\ X_n \end{matrix}}$ $\quad F_{nn}(X) = Y$ (output)

Loss function:

$L(Y) = \text{mistake}$

Loss tells use how much of a mistake MLP made, so we can correct it

Then we use math to correct the mistake

Objectives

To approximate solution using:

- Numerical Solver
- Multilayer Perceptron
- Kolmogorov-Arnold Network
- Comparison

SfS2 SECURE for Student Success

Kolmogorov Arnold Network (KAN)

- **Kolmogorov-Arnold Representation Theorem**:
  - Theoretical foundation for representing multivariate continuous functions as a sum of univariate functions
- **Architecture**:
  - Multiple layers, each modeling a univariate func.
  - Specific activation functions to capture non-linearities
  - Provides structured representation, enhancing interpretability compared to traditional NNs

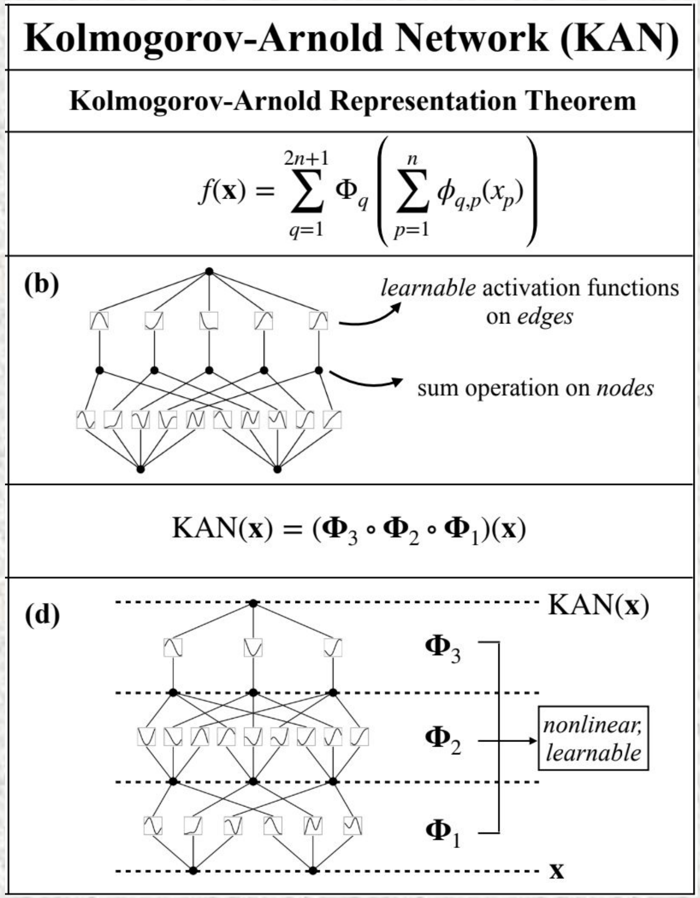


Figure: KAN Architecture [1]

Kolmogorov Arnold Network (KAN)

- **Why use KAN as a physics-informed neural network (PINN)?**
  - Liu et al. claims that KAN’s are promising alternatives to MLPs [1]:
    - KANs posses faster neural scaling laws that MLPs
    - Smaller KANs can achieve comparable accuracy than larger MLPs
    - KANs are more accurate and interpretable than MLPs

SfS²
SECURE for
Student Success

Objectives

To approximate solution using:

- Numerical solver
- Multilayer Perceptron
- Kolmogorov-Arnold Network
- Comparison

Comparing MLP and KAN

- **Small gamma (0.8)**, simulation time: 0-6s (No chaotic behavior)
- Data points: 1200

- **For MLP we used:**

layers: [1, 50, 50, 1]

num epochs: 30,000

- **For KAN we used:**

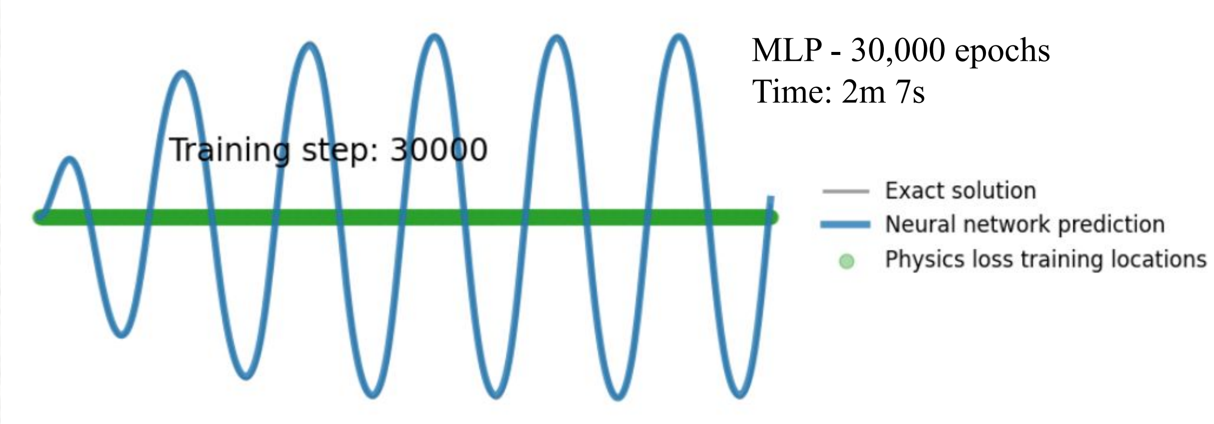
layers: [1,12,12,1]

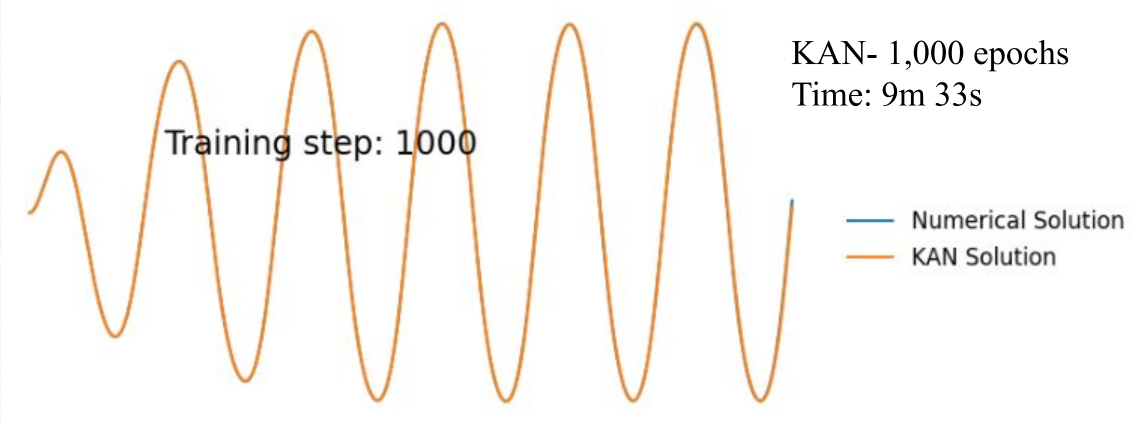
- doesn’t work

layers: [1, 8, 8, 8, 1]

- works, much slower than MLP

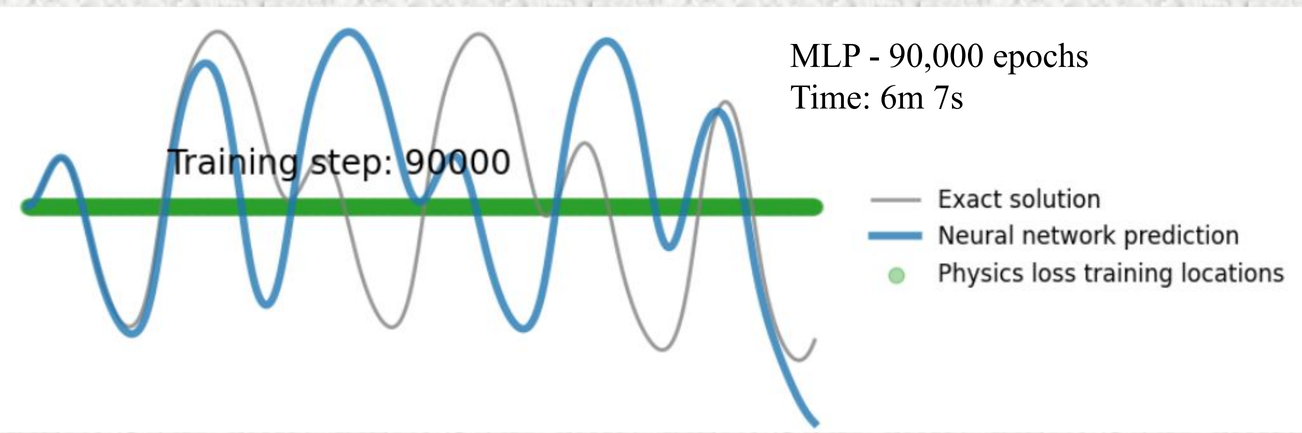
num epochs: 1,000





Comparing MLP and KAN

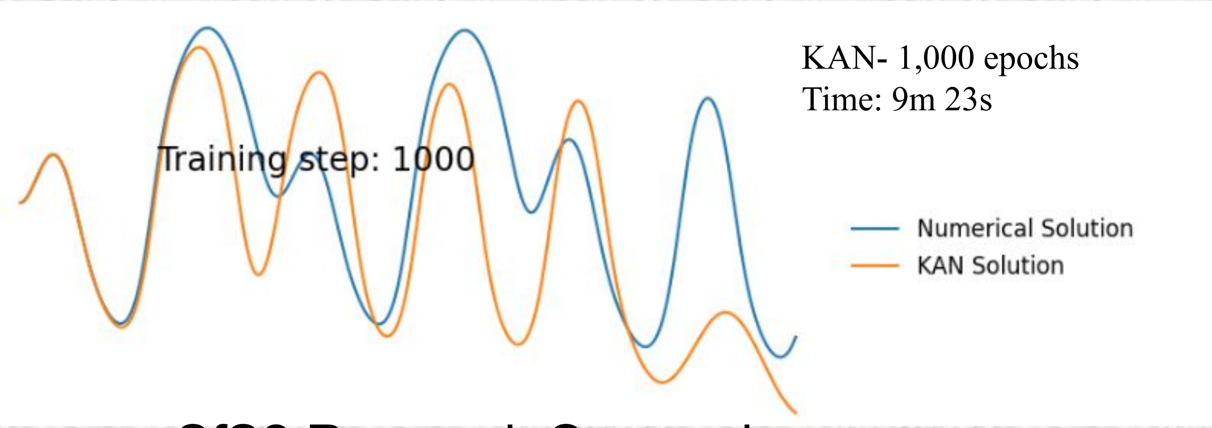
- **Larger gamma (1.1)**, simulation time: 0-6s (chaotic behavior)
- Data points: 1200



- **For MLP we used:**

layers: [1, 200, 200, 1]

num epochs: 90,000



- **For KAN we used:**

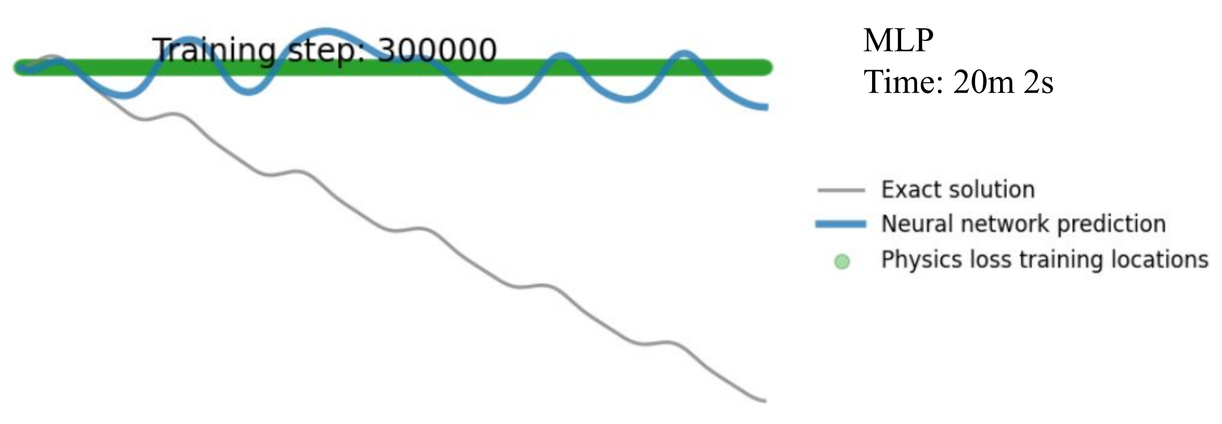
layers: [1, 8, 8, 8, 1]

- works, much slower than MLP

num epochs: 1,000

Comparing MLP and KAN

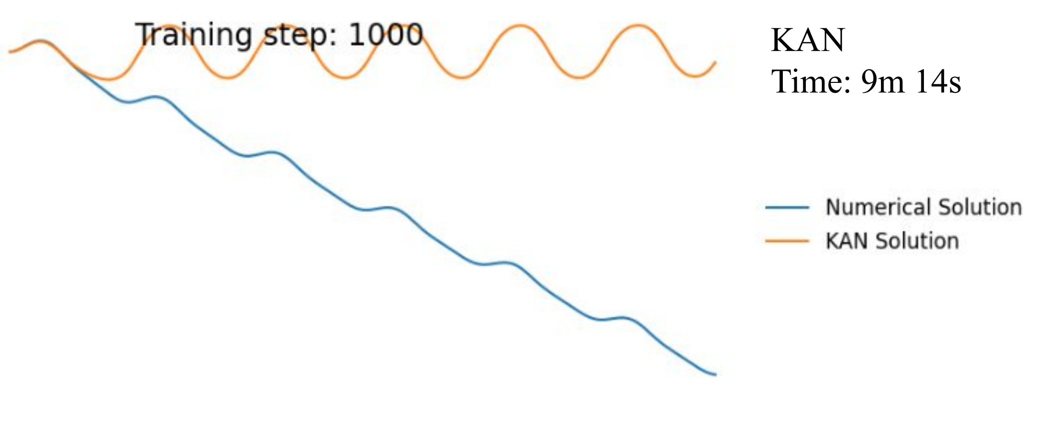
- **Larger gamma (1.3)**, simulation time: 0-6s (chaotic behavior)
- Data points: 1200



- **For MLP we used:**

layers: [1, 512, 512, 1]

num epochs: 300,000



- **For KAN we used:**

layers: [1, 8, 8, 8, 1]

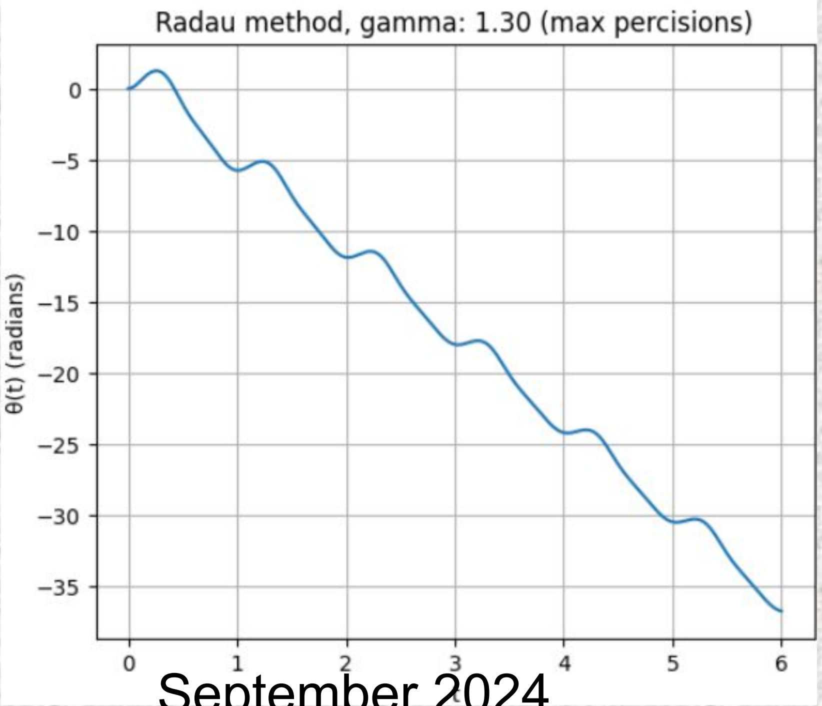
- works, much slower than MLP

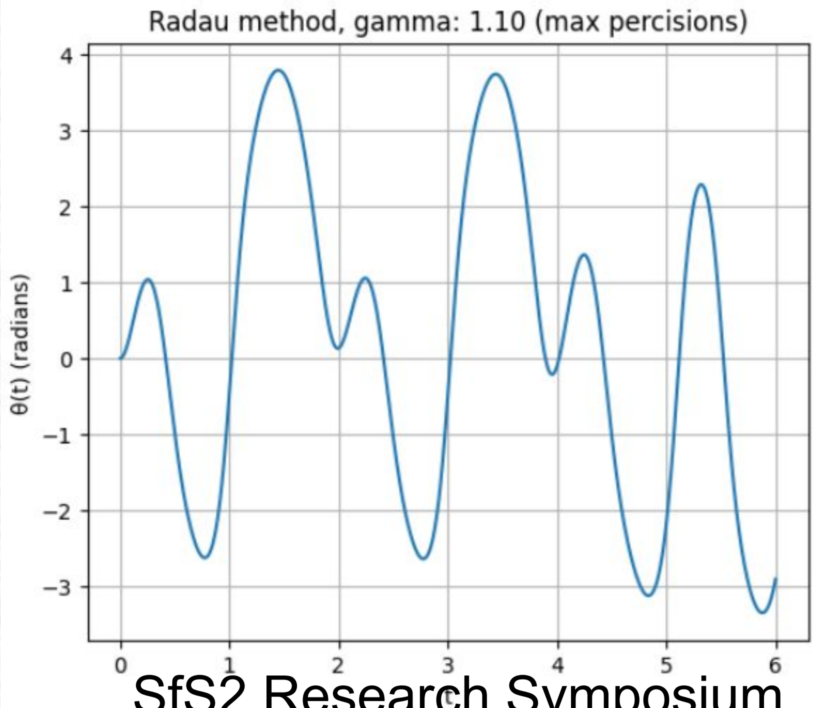
num epochs: 1,000

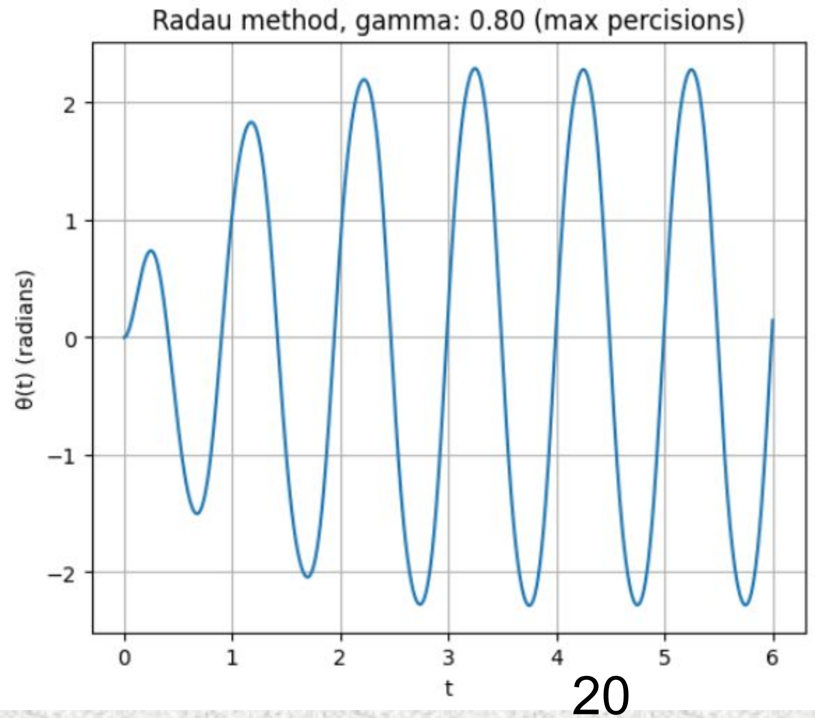
SECURE for Student Success

Comparing Numerical Sol. and NN

- All neural network benchmarks were done using numerical solution
- It is much faster than MLP and KAN, even with max precision we could do, it took max of 32s to accurately approximate solution from time 0-6s for any gamma.







Conclusion

- We did not find advantages of using Neural Network methods to solve nonlinear differential equations especially in chaotic regime.

- We didn’t find that KAN has better performance than MLP, as claimed in [1].

- Newly developed MultKAN could improve performance [2].

SfS2
SECURE for Student Success

References

- [1] Z. Liu *et al.*, “KAN: Kolmogorov-Arnold Networks,” Apr. 30, 2024, *arXiv*: arXiv:2404.19756. doi: 10.48550/arXiv.2404.19756.

- [2] Z. Liu *et al.*, “KAN 2.0: Kolmogorov-Arnold Networks Meet Science,” Aug. 19, 2024, *arXiv*: arXiv:2408.10205. doi: 10.48550/arXiv.2408.10205.

SfS2 SECURE for Student Success

Questions?