Solving the Cauchy-Euler Differential Equation using Physics-Informed Neural Networks

Date of Submission: 09/11/2024

● Introduction:

This project explores the application of Physics-Informed Neural Networks (PINNs) to solve the second-order Cauchy-Euler differential equation. This equation, frequently encountered in various fields of science and engineering, presents a challenging problem for traditional numerical methods, especially when dealing with complex boundary conditions or non-linear variations. PINNs offer a promising alternative by incorporating the underlying physics of the equation directly into the learning process of the neural network. This approach can potentially lead to more accurate and robust solutions, particularly in scenarios where traditional methods struggle.

● Mathematical Formulation:

Defining the Differential Equation:

The Cauchy-Euler equation addressed in this project is:

$$x^2(d^2y/dx^2) + \alpha x(dy/dx) + \beta y = 0$$

Here, α and β are constants.

Analytical or Traditional Methods:
The analytical solution for this specific Cauchy-Euler equation, with complex roots, is:

$$y = x^{-0.5}(c_1 * cos(b * ln(x))) + (c_2 * sin(b * ln(x)))$$

This analytical solution is used to generate the training data and serves as a benchmark for evaluating the performance of the PINN. Traditional methods for solving such equations often involve transforming the equation into a constant-coefficient equation, which can then be solved using standard techniques.

● Assumptions:

The primary assumption is that the analytical solution accurately represents the underlying physics of the system being modeled. The addition of noise to the training data simulates real-world scenarios where measurements are subject to error.

- ## PINN Design:

Neural Network Architecture:

A fully connected feedforward neural network is employed. It consists of:

- ➔ Input Layer: 1 neuron (for x)
- ➔ Hidden Layer 1: 64 neurons with tanh activation
- ➔ Hidden Layer 2: 64 neurons with tanh activation
- ➔ Output Layer: 1 neuron (for y)

Training Process:

The network is trained using the Adam optimizer with a learning rate of 3e-4 for the PINN and 1e-4 for the standard NN. The training process runs for 4000 epochs. The loss function for the PINN is a combination of data loss (Mean Squared Error between predicted and actual y values) and physics loss (Mean Squared Error of the Cauchy-Euler equation residual). A weighting factor (λ_physics) balances the contribution of these two losses, starting at 0.0001 and gradually increasing during training.

- ## Implementation:

Libraries Used:

- ➔ Python
- ➔ PyTorch
- ➔ NumPy
- ➔ Matplotlib
- ➔ Scipy
- ➔ Scikit-learn

Code Description:

The code is structured as follows:

- ➔ Data Generation: Synthetic data is generated using the analytical solution with added noise.
- ➔ Normal Neural Network (NN) Training: A standard NN is trained to predict y from x.
- ➔ PINN Training: The PINN is trained using both data and physics loss.
- ➔ Evaluation and Comparison: The performance of both models is evaluated against the actual data and visualized.

PINN Implementation:

```python
model_pinn = NeuralNet().to(device)
optimizer_pinn = torch.optim.Adam(model_pinn.parameters(), lr=3e-4)

criterion = nn.MSELoss()

lambda_physics = 0.0001

# Training loop for PINN
epochs = 4000
loss_history_pinn = []

for epoch in range(epochs):
    model_pinn.train()
    optimizer_pinn.zero_grad()

    x_train = x_train.clone().detach().requires_grad_(True)

    # Data loss (mean squared error)
    y_pred = model_pinn(x_train)
    data_loss = criterion(y_pred, y_train)

    # Physics loss (Cauchy-Euler equation residual)
    dydx = torch.autograd.grad(y_pred, x_train, torch.ones_like(y_pred), create_graph=True)[0]
    d2ydx2 = torch.autograd.grad(dydx, x_train, torch.ones_like(dydx), create_graph=True)[0]

    physics_residual = x_train**2 * d2ydx2 + alpha * x_train * dydx + beta * y_pred
    physics_loss = torch.mean(physics_residual**2)

    total_loss = data_loss + lambda_physics * physics_loss

    total_loss.backward()
    optimizer_pinn.step()

    loss_history_pinn.append(total_loss.item())

    if (epoch + 1) % 100 == 0:
        print(f'Epoch {epoch + 1}/{epochs}, Total Loss: {total_loss.item()}, Data Loss: {data_loss.item()}, Physics Loss: {physics_loss.item()}')

    # Gradually increasing the influence of physics loss over the epochs so that physics loss also gets reduced!
    if epoch % 500 == 0 and epoch > 0:
        lambda_physics += 0.0001
```

Describing the code implementation:

In this implementation, a Physics-Informed Neural Network (PINN) is used to solve a Cauchy-Euler differential equation by combining data loss and physics loss during training. The model, optimized with Adam, minimizes the mean squared error between its predictions and the actual data (data loss) while also satisfying the differential equation (physics loss). The physics loss is calculated by computing the residual of the Cauchy-Euler equation using first and second derivatives of the model's predictions. The total loss is a weighted sum of data and physics losses, with a scaling factor (`lambda_physics`) that gradually increases throughout training, allowing the model to initially focus on data fitting before emphasizing adherence to the equation. This approach ensures the model predictions respect both the observed data and the governing physical laws.

Dataset generation:

Two distinct datasets were used in this project, one for each assignment:

Dataset 1 (MATLAB): For the first assignment involving the regular neural network, a dataset was generated using MATLAB. The Cauchy-Euler equation was solved numerically using the ode45 solver with constants $\alpha = 0.01$ and $\beta = 0.44$. 1000 data

points were generated for the input variable x. The initial conditions used for the ode45 solver were y(0) = 1 and dy/dx(0) = 0.

Dataset 2 (Analytical Solution): For the second assignment, which involved both the regular NN and the PINN, a synthetic dataset was generated within the Jupyter Notebook environment using the analytical solution of the Cauchy-Euler equation:

$$y = x^{-0.5}(c_1 * cos(b * ln(x))) + (c_2 * sin(b * ln(x)))$$

Constants $\alpha = 2.0$ and $\beta = 1.0$ were used, and 1000 data points were generated. To simulate real-world noise, a Gaussian noise factor with a standard deviation of 0.01 was added to the generated y values.

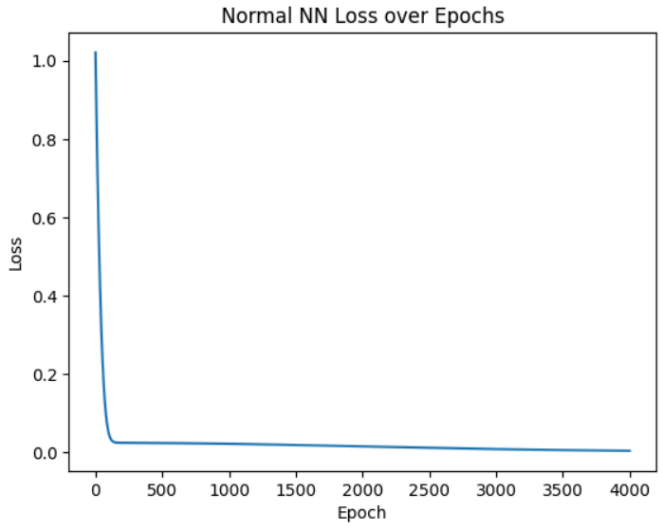
Challenges Encountered:

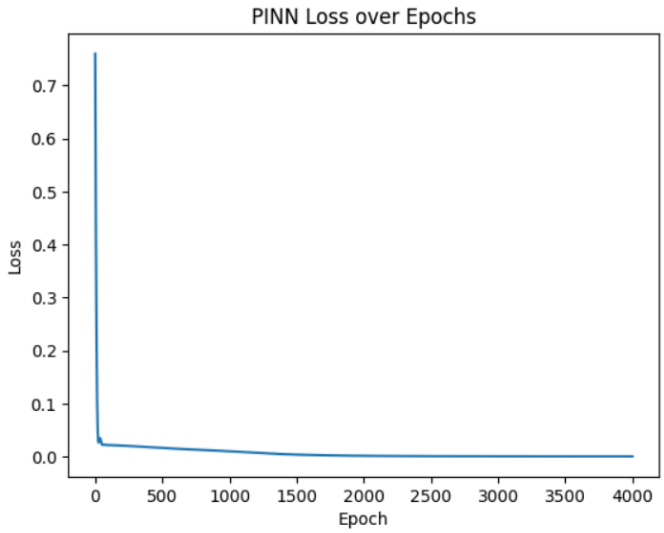
- ➔ **Dataset Generation and Noise:** The generated dataset was intentionally complex, incorporating Gaussian noise and non-linear patterns. Creating a dataset that accurately represents the physics while also being suitable for training both the regular NN and the PINN was challenging.
- ➔ **Model Accuracy for Nonlinear Equations:** Solving complex differential equations (like the nonlinear Cauchy-Euler equation) using neural networks adds complexity. Ensuring that the model accurately approximates these equations while fitting the generated dataset was challenging and required experimentation with the network architecture, learning rate, and optimizer settings.
- ➔ **Training Stability and Loss Convergence:** One of the primary challenges was ensuring stable training and minimizing fluctuations in the loss. Initially, the total loss and physics loss fluctuated significantly, leading to an increase in total loss with more epochs, making it challenging to achieve convergence.
- ➔ **Balancing Data and Physics Loss:** In PINNs, managing the balance between data loss and physics loss is crucial. Tuning this balance was challenging, as an improper balance could lead to the model overfitting to the data or physics constraints, affecting accuracy.
- ➔ **Visualization and Interpretation:** Visualizing the performance and interpreting results, especially in comparing PINNs and regular neural networks, required careful consideration. Ensuring clear, interpretable plots for actual vs. predicted data and loss convergence helped to better understand the model performance.

● Results:

The performance of both the standard Neural Network (NN) and the Physics-Informed Neural Network (PINN) was evaluated by comparing their predictions against the actual data generated from the analytical solution of the Cauchy-Euler equation. The following graphs illustrate the findings:

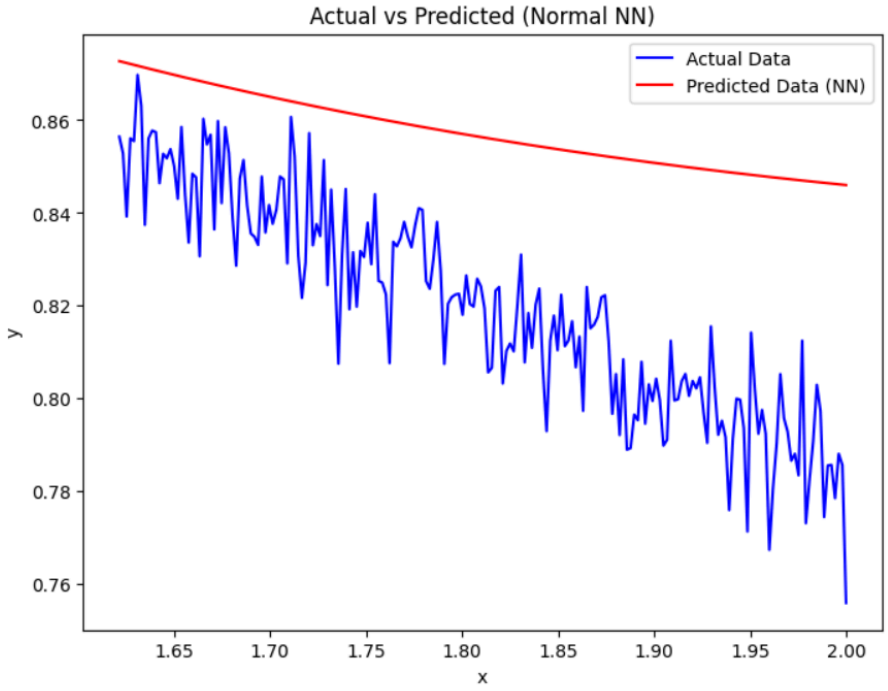
- **Plots of the loss functions of both Normal NN and PINN:**

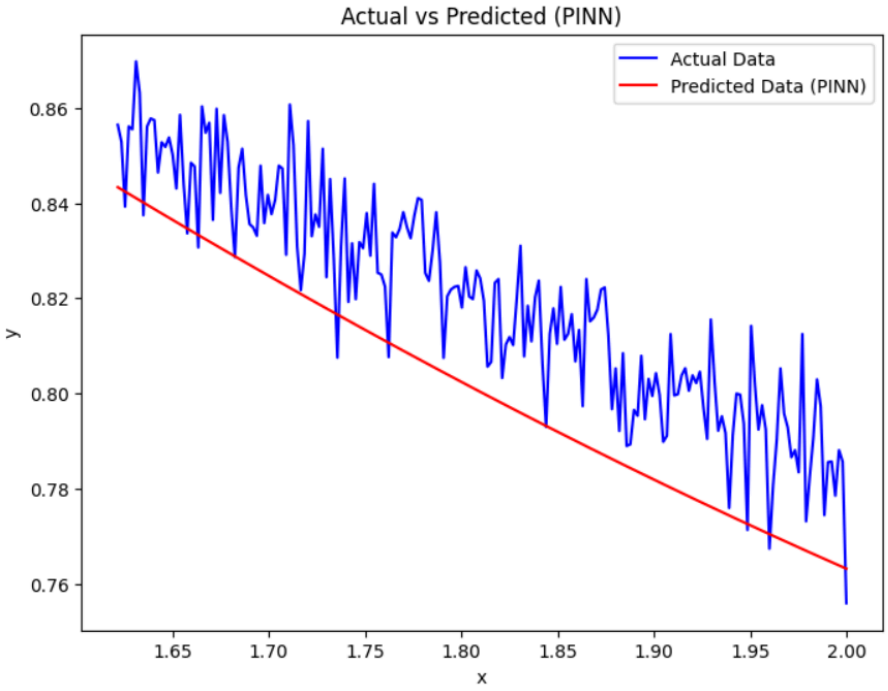




The loss history plots for both the Regular NN and PINN demonstrate a rapid decrease in loss during the initial epochs, indicating successful learning. The NN loss converges to a slightly lower value than the PINN loss. This is expected as the PINN has to balance both data matching and adherence to the underlying physics.

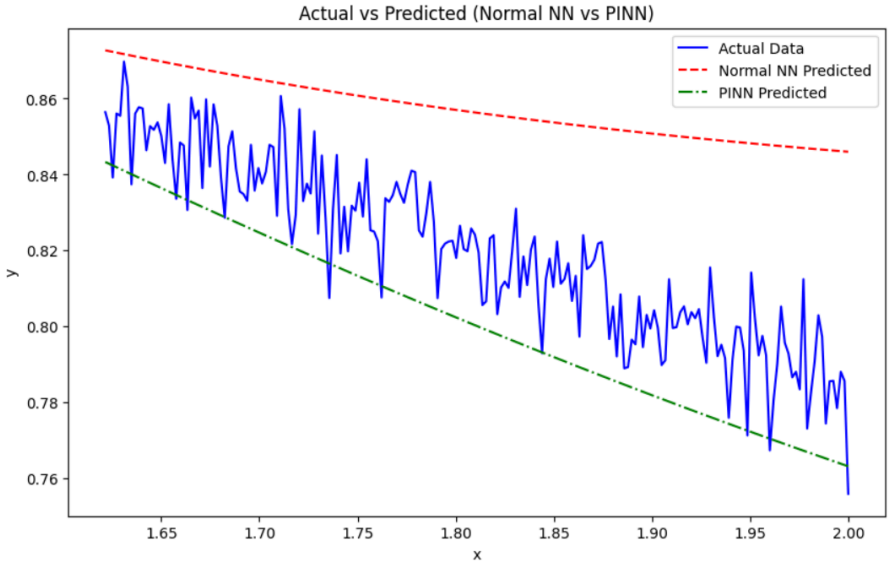
- **Comparison of predictions of models with the actual data of both models Regular NN and PINN:**





The above figures depict the actual data versus the predicted values for the Regular NN and PINN, respectively. Both models capture the overall decreasing trend of the solution. However, the Regular NN predictions appear to deviate more significantly from the actual data, especially in regions with more fluctuations. The PINN, by incorporating the physics of the equation, exhibits a smoother prediction curve that aligns more closely with the actual data, particularly in capturing the underlying trend.

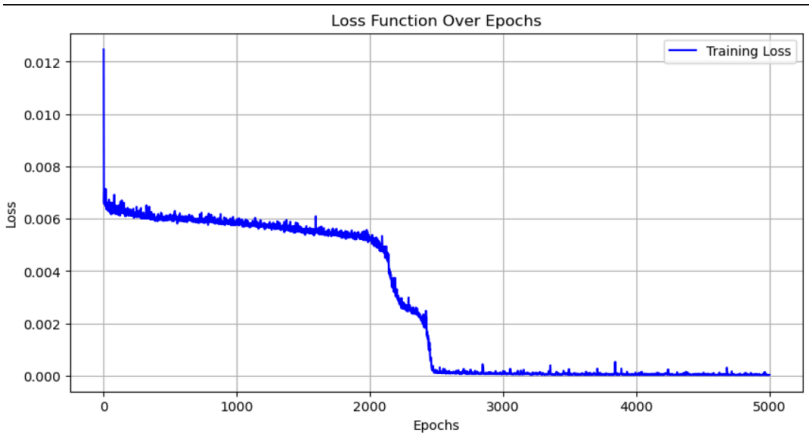
- **The combined plot comparing the performance of both Regular NN and PINN:**



The above plot provides a direct comparison of the Regular NN and PINN predictions against the actual data. This visualization clearly highlights the improved performance of the PINN. While the Regular NN captures the general trend, the PINN demonstrates a better fit to the noisy data while still adhering to the underlying physics of the Cauchy-Euler equation. This suggests that the physics-informed component of the loss function helps to regularize the PINN's predictions and prevent overfitting to the noise in the training data.

(This section below is only for the first assignment where MATLAB dataset was used)
In the first assignment, a regular neural network was trained solely on the MATLAB-generated dataset. The following results pertain to this specific experiment:

- **Loss Function:**



This plot displays the training loss over epochs. The loss decreases rapidly in the initial epochs and continues to decrease, albeit more slowly, throughout the training process. This indicates that the network is learning effectively and converging towards a solution. The final loss value suggests a good fit to the training data.

- **Performance Metrics:** The performance of the regular NN on the test set was evaluated using several metrics:

  Mean Squared Error (MSE): 0.000639
  Mean Absolute Error (MAE): 0.006571
  R-squared: 0.946823
  Test Mean Squared Error: 0.021086

  The low MSE and MAE values indicate that the NN's predictions are close to the actual values. The high R-squared value (close to 1) further suggests a strong correlation between the predicted and actual values, indicating a good fit to the data.

● Conclusion:

The results demonstrate the effectiveness of Physics-Informed Neural Networks in solving the Cauchy-Euler differential equation. The PINN, by incorporating the equation's physics into its loss function, achieved a more accurate and robust solution compared to the standard NN. The smoother prediction curve and closer alignment with the actual data, even in the presence of noise, highlight the PINN's ability to generalize better and avoid overfitting.

● Limitations:

While the PINN demonstrated promising results, some limitations should be acknowledged. The model's performance relies on the accuracy of the analytical solution used for data generation. In real-world scenarios, where the analytical solution might not be available, obtaining accurate training data could be a challenge. Furthermore, the current implementation focuses on a specific form of the Cauchy-Euler equation. Generalizing the approach to other forms or more complex boundary conditions requires further investigation.

● Future Improvements:

- ➔ Exploring different network architectures, such as deeper networks or different activation functions.
- ➔ Experimenting with alternative optimizers or learning rate schedules.

- ➔ Investigating adaptive weighting strategies for the data and physics loss.
- ➔ Applying the PINN approach to more complex Cauchy-Euler equations with different boundary conditions.

THANK YOU!