Charged Particle in an Electromagnetic Field

by:Ahmad AliAhmad

TOC

Project Overview

Given the trajectory data of a charged particle in a electromagnetic field, we would like to predict its future states using:

- SympNets
- PNNs
- PINNs

$$m\ddot{x} = q(E + \dot{x} \times B),$$

where m is the mass, $x \in \mathbb{R}^3$ denotes the particle's position, q is the electric charge, $B = \nabla \times A$ denotes the magnetic field, and $E = -\nabla\varphi$ is the electric field with A, φ being the potentials. Let $\dot{x} = v$ be the velocity of the charged particle, then the governing equations of the particle's motion can be expressed as

$$\begin{aligned} \begin{pmatrix} \dot{v} \\ \dot{x} \end{pmatrix} &= \begin{pmatrix} -\frac{q}{m^2}\hat{B}(x) & -\frac{1}{m}I \\ \frac{1}{m}I & 0 \end{pmatrix} \nabla H(v, x), \\ H(v, x) &= \frac{1}{2} m v^T v + q\varphi(x), \end{aligned} \tag{1}$$

$$\hat{B}(x) = \begin{pmatrix} 0 & -B_3(x) & B_2(x) \\ B_3(x) & 0 & -B_1(x) \\ -B_2(x) & B_1(x) & 0 \end{pmatrix}$$

for $B(x) = (B_1(x), B_2(x), B_3(x))$. Here we test the dynamics with $m = 1$, $q = 1$, and

$$A(x) = \frac{1}{3}\sqrt{x_1^2 + x_2^2} \cdot (-x_2, x_1, 0), \quad \varphi(x) = \frac{1}{100\sqrt{x_1^2 + x_2^2}}$$

for $x = (x_1, x_2, x_3)^T$. Then

$$\begin{aligned} B(x) &= (\nabla \times A)(x) = (0, 0, \sqrt{x_1^2 + x_2^2}), \\ E(x) &= -(\nabla\varphi)(x) = \frac{(x_1, x_2, 0)}{100(x_1^2 + x_2^2)^{\frac{3}{2}}}. \end{aligned}$$

SympNet Architecture

01 LA-SympNet

- **Linear modules.**

$$\mathcal{L}_n \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} I & 0/S_n \\ S_n/0 & I \end{pmatrix} \cdots \begin{pmatrix} I & 0 \\ S_2 & I \end{pmatrix} \begin{pmatrix} I & S_1 \\ 0 & I \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} + b, \quad p, q \in \mathbb{R}^d,$$

where $S_i \in \mathbb{R}^{d \times d}$ are symmetric, $b \in \mathbb{R}^{2d}$ is the bias, while the unit upper triangular symplectic matrices and the unit lower triangular symplectic matrices appear alternately. In this module, S_i (represented by $A_i + A_i^T$ in practice) and b are parameters to learn. In fact, $\mathcal{L}_n$ can represent any linear symplectic map [37].

- **Activation modules.**

$$\mathcal{N}_{up} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} p + \tilde{\sigma}_a(q) \\ q \end{pmatrix},$$

$$\mathcal{N}_{low} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} p \\ \tilde{\sigma}_a(p) + q \end{pmatrix}, \quad p, q \in \mathbb{R}^d,$$

where $\tilde{\sigma}_a(x) := a \odot \sigma(x)$ for $x \in \mathbb{R}^d$. Here $\odot$ is the element-wise product, σ is the activation function, and $a \in \mathbb{R}^d$ is the parameter to learn.

02 G-SympNet

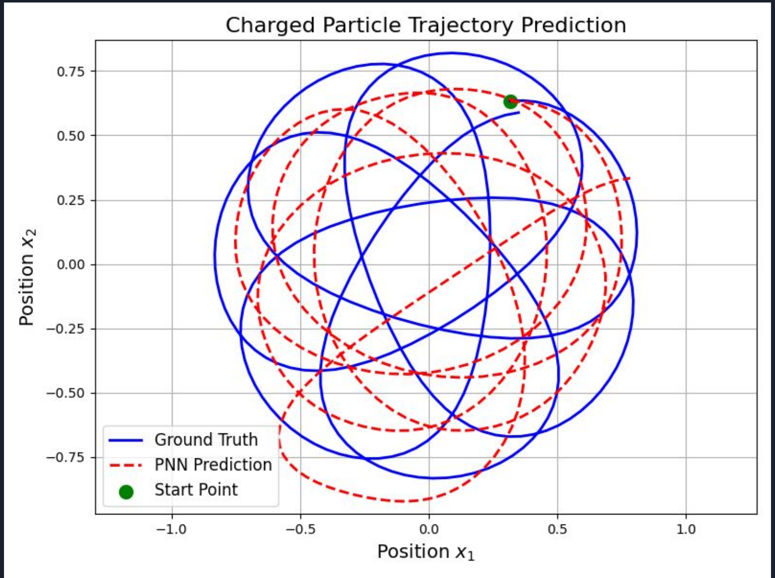
- **Gradient modules.**

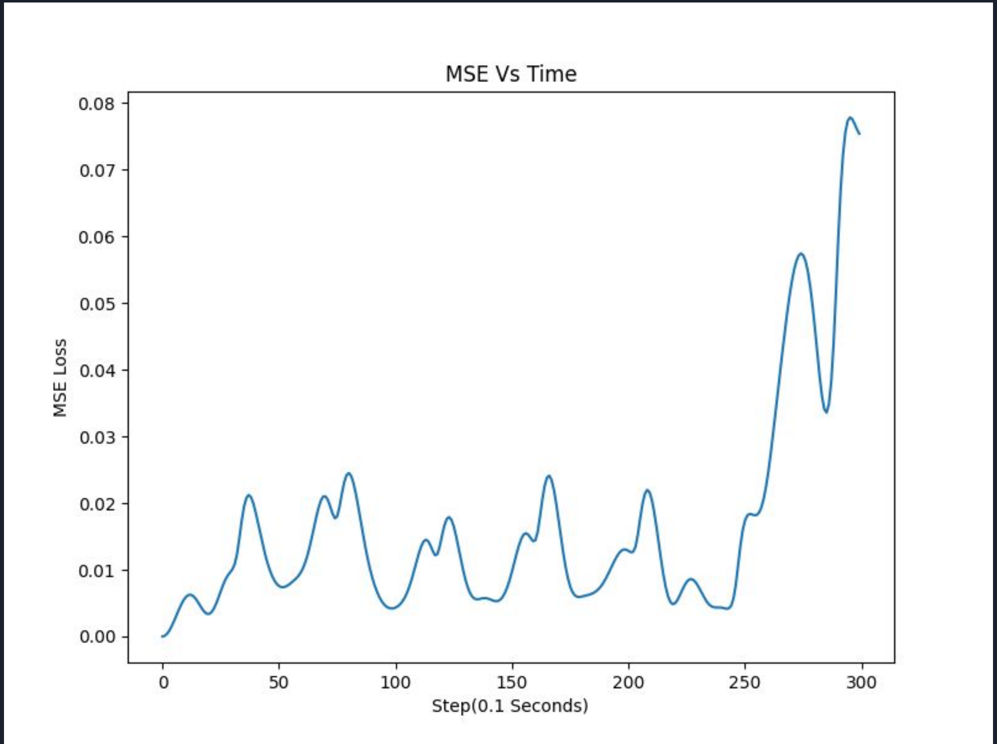
$$\mathcal{G}_{up} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} p + \hat{\sigma}_{K,a,b}(q) \\ q \end{pmatrix},$$

$$\mathcal{G}_{low} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} p \\ \hat{\sigma}_{K,a,b}(p) + q \end{pmatrix}, \quad p, q \in \mathbb{R}^d,$$

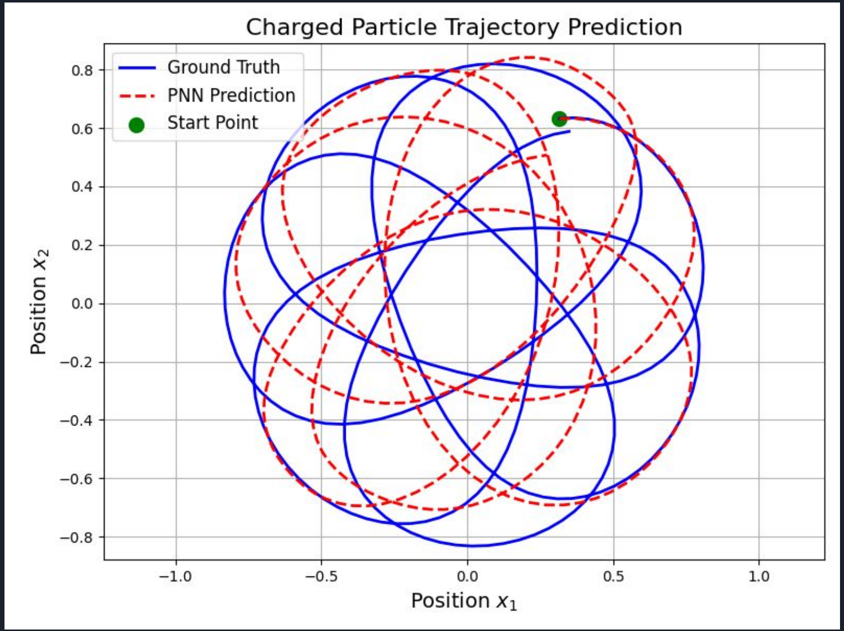
where $\hat{\sigma}_{K,a,b}(x) := K^T(a \odot \sigma(Kx + b))$ for $x \in \mathbb{R}^d$. Here $a, b \in \mathbb{R}^l$, $K \in \mathbb{R}^{l \times d}$ are the parameters to learn, and l is a positive integer regarded as the width of the module.

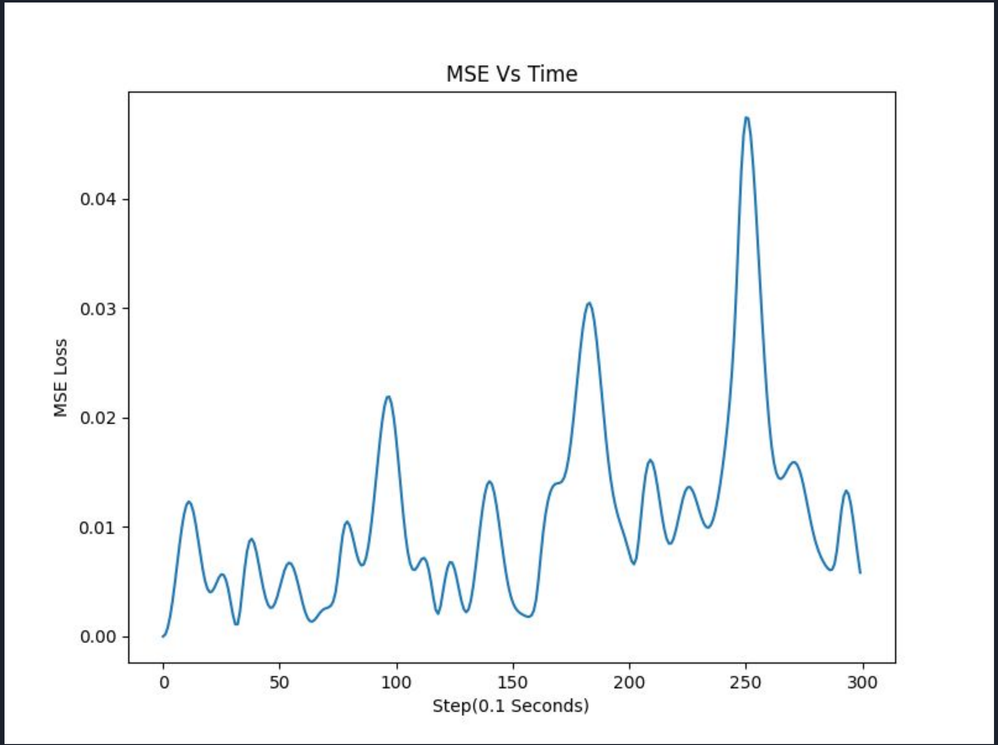
LA-SympNet Results





G-SympNet Results





PNN Architecture

01 NICE

- **Volume-preserving modules.**

$$\mathcal{V}_{up}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 + m_1(x_2) \\ x_2 \end{pmatrix},$$
$$\mathcal{V}_{low}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ m_2(x_1) + x_2 \end{pmatrix},$$
$$x_1 \in \mathbb{R}^d, x_2 \in \mathbb{R}^{n-d},$$

where $m_1 : \mathbb{R}^{n-d} \to \mathbb{R}^d$ and $m_2 : \mathbb{R}^d \to \mathbb{R}^{n-d}$ are modeled as fully-connected neural networks.

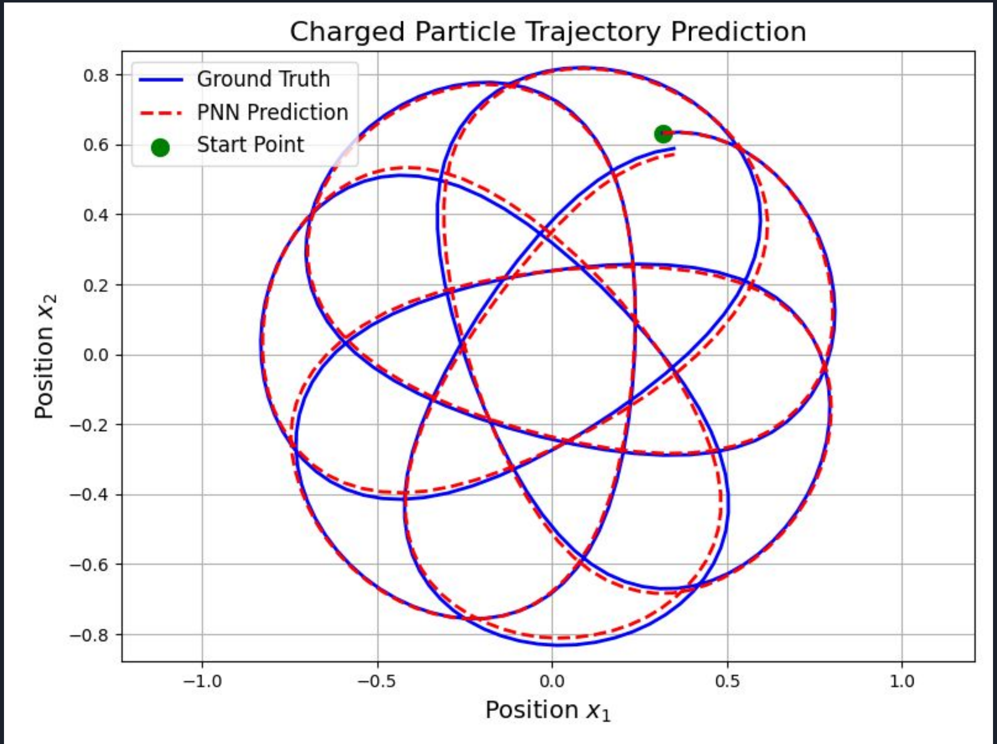
02 E-SympNet

- **Extended modules.**

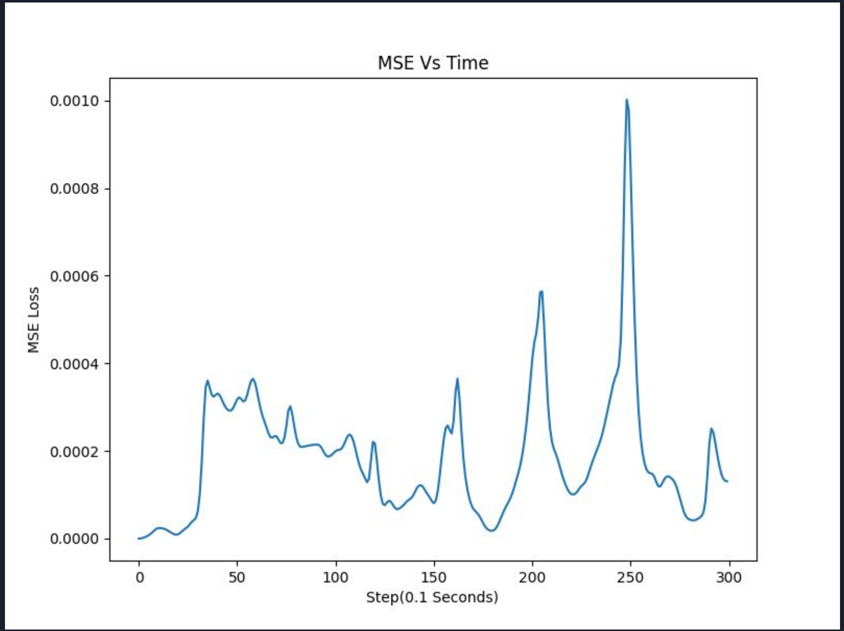
$$\mathcal{E}_{up}\begin{pmatrix} p \\ q \\ c \end{pmatrix} = \begin{pmatrix} p + \hat{\sigma}_{K_1,K_2,a,b}(q,c) \\ q \\ c \end{pmatrix},$$
$$\mathcal{E}_{low}\begin{pmatrix} p \\ q \\ c \end{pmatrix} = \begin{pmatrix} p \\ \hat{\sigma}_{K_1,K_2,a,b}(p.c) + q \\ c \end{pmatrix},$$
$$p, q \in \mathbb{R}^d,\ c \in \mathbb{R}^{n-2d},$$

where $\hat{\sigma}_{K_1,K_2,a,b}(x,c) := K_1^T(a \odot \sigma(K_1 x + K_2 c + b))$ for $x \in \mathbb{R}^d$, $c \in \mathbb{R}^{n-2d}$. Here $a, b \in \mathbb{R}^l$, $K_1 \in \mathbb{R}^{l\times d}$, $K_2 \in \mathbb{R}^{l\times(n-2d)}$ are the parameters to learn, and l is a positive integer regarded as the width of the module.

PNN Results



Charged Particle Trajectory Prediction
Ground Truth
PNN Prediction
Start Point
Position x_2
Position x_1
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−0.5
0.0
0.5
1.0



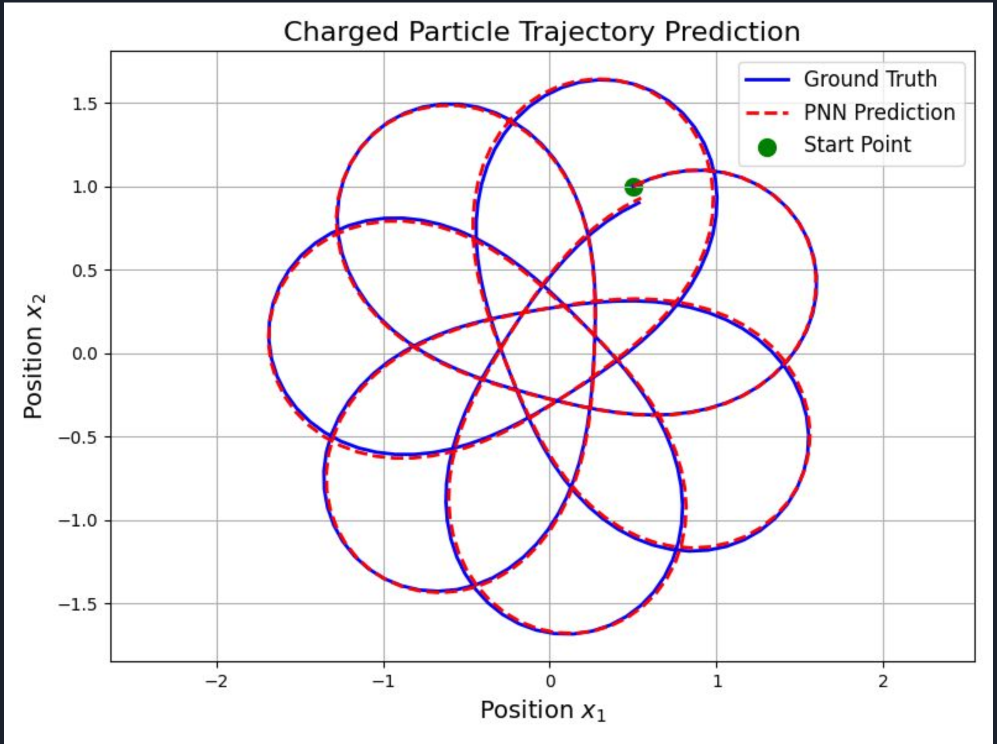
MSE Vs Time
MSE Loss
Step(0.1 Seconds)
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
0
50
100
150
200
250
300

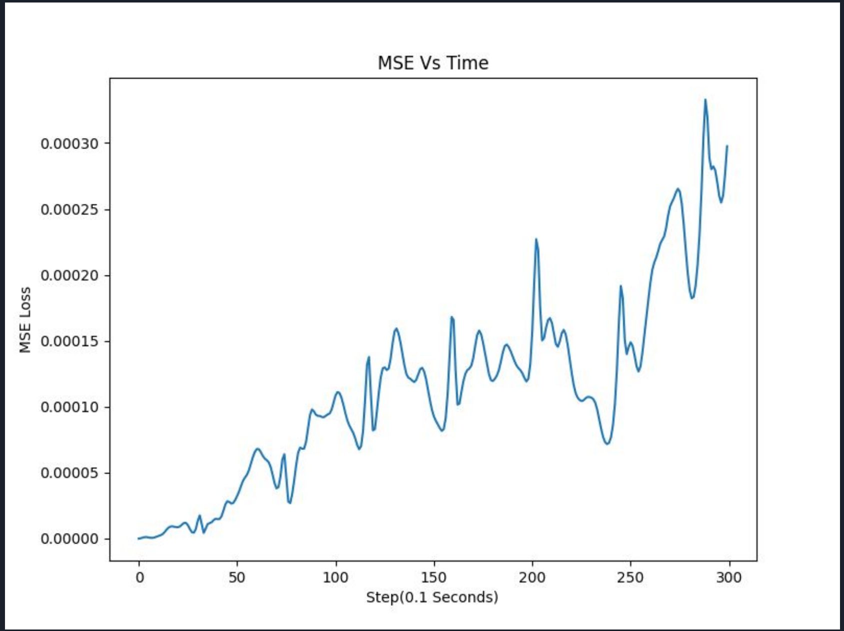
PINN Architecture

```python
# Get risidual physics loss
def risidualLoss(self,X,Y,dt):
    # Step forward and back
    XPred = self.Rrk4StepForward(X,dt)
    YPred = self.Rrk4StepBackwards(Y,dt)
    # split the position from velocity for better accuracy
    Xp = XPred[:,:2]
    Xq = XPred[:,2:]
    Yp = YPred[:,:2]
    Yq = YPred[:,2:]
    # Get all the losses
    XPredLoss = F.mse_loss(Xp,Y[:,:2]) + F.mse_loss(Xq,Y[:,2:])
    YPredLoss = F.mse_loss(Yp,X[:,:2]) + F.mse_loss(Yq,X[:,2:])
    return YPredLoss + XPredLoss
```

```python
# Used for the risidual loss
def dynamicMatrix(self, X):
    batch_size = X.shape[0]
    x3, x4 = X[:, 2], X[:, 3]
    a = (x3 ** 2 + x4 ** 2) ** (1/2)  # shape: (batch_size,)
    b = self.lambda1 # This coresponds to 1/m
    c = self.lambda2 * b # This corespons to q/m * 1/m

    zeros = torch.zeros(batch_size, device=X.device)

    row0 = torch.stack([zeros, c * a, -b.expand_as(a), zeros], dim=1)
    row1 = torch.stack([-c * a, zeros, zeros, -b.expand_as(a)], dim=1)
    row2 = torch.stack([b.expand_as(a), zeros, zeros, zeros], dim=1)
    row3 = torch.stack([zeros, b.expand_as(a), zeros, zeros], dim=1)

    dynamic = torch.stack([row0, row1, row2, row3], dim=1)  # shape: (batch_size, 4, 4)
    return dynamic
```

PINN Results





Thank You!