Modelling dynamical systems: Learning ODEs with no internal ODE resolution

Bowen Zhu[1] Fotis Kapotos[1] Sida-Bastien Li[1]
E. Goutierre[2,3] Johanne Cohen[2] **Hayg Guler**[3] Michèle Sebag[2]

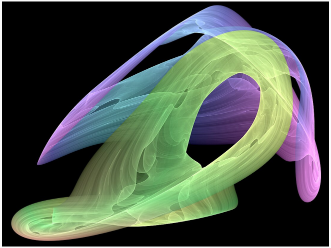
Friday 27th September 2024

[1]Centrale Supélec, Université Paris-Saclay, France

[2]LISN, Université Paris-Saclay, France,

[3]IJCLab, Université Paris-Saclay, France.

Dynamic system modeling



Goal:

Create a surrogate model for particle accelerators

- Predict beam trajectory from initial state and accelerator settings
- Accelerator trajectories could be anything from beam position, size, emittance along the beamline
- trajectory could be also seen as any times series data
- Model beam trajectory using an **ODE framework**

Problem Description: Supervised Learning for ODE Approximation

Objective: Learn an approximate trajectory of the particle beam based on time measurements and control parameters.

Key Components:

- **Time Measurements:**

  $$(T_i)_{i=1}^N \quad \text{with} \quad T_i \in [t_0, T]$$

  These are the discrete time points where the trajectory is observed.
- **Trajectory Function:**

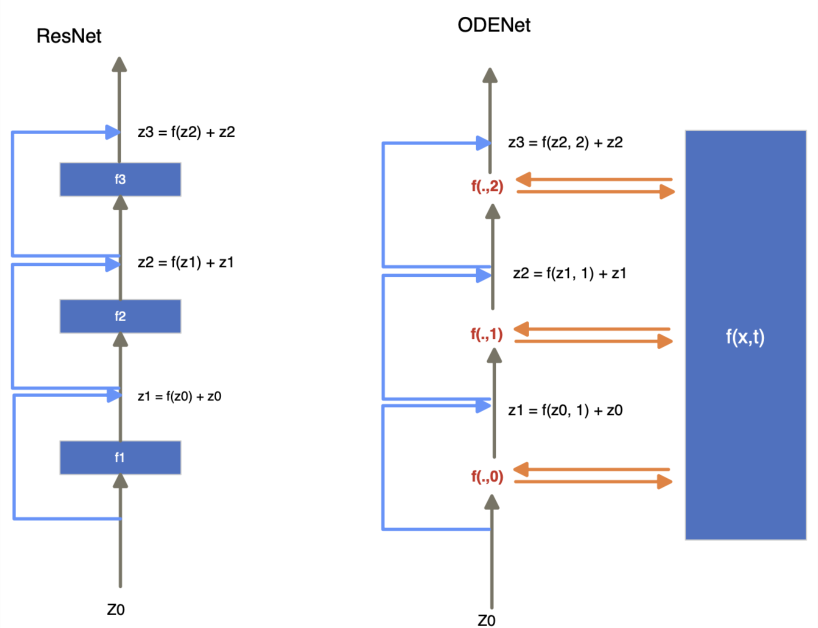
  $$\mathrm{u} : [t_0, T] \times \mathbb{R}^p \to \mathbb{R}^n$$

  The trajectory $\mathrm{u}(t, \mathbf{c})$ maps time t and the initial state/control parameters $\mathbf{c}$ to a position in the n-dimensional space.
- **Control Parameters:**

  $$\mathbf{c} \in \mathbb{R}^m$$

  Control parameters $\mathbf{c}$ influence the trajectory of the beam. These could represent accelerator settings or other system controls.
- **Goal:** Approximate the trajectory $\mathrm{u}(t, \mathbf{c})$ by learning from the initial

Background


ResNet
ODENet
z3 = f(z2) + z2
f3
z2 = f(z1) + z1
f2
z1 = f(z0) + z0
f1
Z0
z3 = f(z2, 2) + z2
f(.,2)
z2 = f(z1, 1) + z1
f(.,1)
z1 = f(z0, 1) + z0
f(.,0)
Z0
f(x,t)

- **Recurrent Neural Networks (RNNs):**

$$\mathrm{u}(t_{i+1}, \mathbf{c}) = \mathrm{u}(t_i, \mathbf{c}) + f_\theta(\mathrm{u}(t_i, \mathbf{c}))$$

  - f_θ is a neural network that captures the system's dynamics.
  - θ refers to the learned weights of the neural network.

- **Neural ODEs (NODEs) Chen-2018:**
  - Unlike RNNs, NODEs model the system's behavior in continuous time:

$$\frac{d\mathrm{u}(t, \mathbf{c})}{dt} = f(\mathrm{u}(t, \mathbf{c}), t, \mathbf{c})$$

  - Here, f_θ is a neural network that approximates the unknown dynamics f.
  - The trajectory $\hat{\mathrm{u}}(t, \mathbf{c})$ is computed by solving the ODE:

$$\frac{d\hat{\mathrm{u}}(t, \mathbf{c})}{dt} = f_\theta(\hat{\mathrm{u}}(t, \mathbf{c}), t, \mathbf{c})$$

  - This approach allows for a more flexible representation of time-evolving processes.

NODE integrates neural networks into the ODE framework by parameterizing the derivative of the state with respect to time using a neural network:

$$\frac{\mathrm{d}\mathrm{u}}{\mathrm{d}t}(t,\mathbf{c}) = f\left(\mathrm{u}(t,\mathbf{c}), t, \mathbf{c}\right) \quad \rightarrow \quad \frac{\mathrm{d}\widehat{\mathrm{u}}}{\mathrm{d}t}(t,\mathbf{c}) = f_\theta\left(\widehat{\mathrm{u}}(t,\mathbf{c}), t, \mathbf{c}\right)$$

NODE Algorithm:

1. Initialize the neural network f_θ with random weights.
2. For a control parameter $\mathbf{c}$ and trajectory $\mathrm{u}(t,\mathbf{c})$:
   - 2.1 Solve the ODE to get $\widehat{\mathrm{u}}(t,\mathbf{c})$.
   - 2.2 Compute the loss: $L(\widehat{\mathrm{u}}(T,\mathbf{c})) = ||\widehat{\mathrm{u}}(T,\mathbf{c}) - \mathrm{u}(T,\mathbf{c})||^2$.
   - 2.3 Calculate the gradient of the loss using the adjoint method.
   - 2.4 Update the network weights based on the gradient.

Resolution: The state estimate $\widehat{u}(t,\mathbf{c})$ at any time t is obtained by numerically solving the ODE.

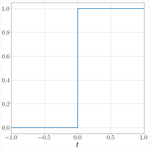
$$\widehat{u}(t,\mathbf{c}) = \mathrm{ODESolve}\left(\widehat{f_\theta}(\cdot,\mathbf{c}), \widehat{u}(t_0,\mathbf{c}), t_0, t, \mathbf{c}\right)$$

Limitations of NODE

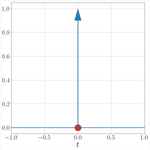
1. Computational Time: NODE relies on numerical ODE solvers to integrate the system's dynamics. This can become computationally intensive, especially for complex models and long-time series.

2. Modeling Discontinuities:

- NODE inherently assumes smooth dynamics governed by the ODEs ($\frac{\mathrm{d}u}{\mathrm{d}t}$ should be well defined everywhere).
- The smooth dynamics assumption makes it challenging to model time series with abrupt changes or discontinuities.



$$\mathrm{u}\left(t, \mathbf{c}\right) = H\left(t\right)$$



$$\frac{\mathrm{d}u}{\mathrm{d}t}\left(t, \mathbf{c}\right) = \delta\left(t\right)$$

Neural Integral Ordinary Differential Equations (NIODE)

Objective: Learn the trajectories u using a model inspired by NODEs.

Additional Information: Incorporate an **extra function** $\mathrm{v} = \mathcal{F}(u)$ that satisfies:

$$\frac{d\mathrm{v}}{dt}(t, \mathbf{c}) = g(\mathrm{u}(t, \mathbf{c}), \mathrm{v}(t, \mathbf{c}), t)$$

where g is called a driving function and must satisfy :

1. **Lipschitz Continuity in State Space:**

$$\|g(\mathrm{u}_2, \mathbf{x}_\mathbf{v}, t) - g(\mathrm{u}_1, \mathbf{x}_\mathbf{v}, t)\| \leq k_\mathrm{u} \|\mathrm{u}_2 - \mathrm{u}_1\|$$

2. **Lipschitz Continuity in Integral State Space:**

$$\|g(\mathrm{u}, \mathbf{x}_{\mathbf{v}2}, t) - g(\mathrm{u}, \mathbf{x}_{\mathbf{v}1}, t)\| \leq k_\mathrm{v} \|\mathbf{x}_{\mathbf{v}2} - \mathbf{x}_{\mathbf{v}1}\|$$

3. **Continuity Over Time:** g is continuous with respect to time.

Different Choices for v and Their Impact on g

1. **Case 1:** If $\mathrm{v}(t, \mathbf{c})$ is the integral of $u(t, \mathbf{c})$:

$$\mathrm{v}(t, \mathbf{c}) = \int_{t_0}^{t} u(s, \mathbf{c})\, ds$$

   Then, $g = u$ (identity with respect to u).

Different Choices for v and Their Impact on g

1. **Case 1:** If $\mathrm{v}(t, \mathbf{c})$ is the integral of $u(t, \mathbf{c})$:

$$\mathrm{v}(t, \mathbf{c}) = \int_{t_0}^{t} u(s, \mathbf{c})\, ds$$

Then, $g = u$ (identity with respect to u).

2. **Case 2:** If $\mathrm{v}(t, \mathbf{c})$ is the exponentially smoothed version of $u(t, \mathbf{c})$:

$$\mathrm{v}(t, \mathbf{c}) = u(t_0, \mathbf{c}) e^{-\lambda(t - t_0)} + \lambda \int_{t_0}^{t} e^{-\lambda(t-s)} u(s, \mathbf{c})\, ds$$

Then, $g = \lambda(u - \mathrm{v})$.

Framework:

We consider an **extra function** $\mathrm{v}(t, \mathbf{c})$ that evolves according to the differential equation, where the **driving function** g verifies certain properties :

$$\frac{\mathrm{d}\mathrm{v}(t,\mathbf{c})}{\mathrm{d}t} = g(\mathrm{u}(t,\mathbf{c}), \mathrm{v}(t,\mathbf{c}), t) \tag{1}$$

Framework:

We consider an **extra function** $\mathrm{v}(t, \mathbf{c})$ that evolves according to the differential equation, where the **driving function** g verifies certain properties :

$$\frac{\mathrm{d}\mathrm{v}(t,\mathbf{c})}{\mathrm{d}t} = g(\mathrm{u}(t,\mathbf{c}), \mathrm{v}(t,\mathbf{c}), t) \tag{1}$$

Training Framework:

- Compute $\mathrm{v}(t, \mathbf{c})$ from all observable trajectories $\mathrm{u}(t, \mathbf{c})$.

Framework:

We consider an **extra function** $\mathsf{v}(t, \mathbf{c})$ that evolves according to the differential equation, where the **driving function** g verifies certain properties :

$$\frac{\mathrm{d}\mathsf{v}(t, \mathbf{c})}{\mathrm{d}t} = g(\mathsf{u}(t, \mathbf{c}), \mathsf{v}(t, \mathbf{c}), t) \tag{1}$$

Training Framework:

- Compute $\mathsf{v}(t, \mathbf{c})$ from all observable trajectories $\mathsf{u}(t, \mathbf{c})$.
- Train a neural network f_θ to learn $\mathsf{u}(t, \mathbf{c})$ from $\mathsf{v}(t, \mathbf{c})$:

$$\mathsf{u}(t, \mathbf{c}) = f_\theta(\mathsf{v}(t, \mathbf{c}), t, \mathbf{c})$$

Framework:

We consider an **extra function** $\mathrm{v}(t, \mathbf{c})$ that evolves according to the differential equation, where the **driving function** g verifies certain properties :

$$\frac{\mathrm{dv}(t, \mathbf{c})}{\mathrm{d}t} = g(\mathrm{u}(t, \mathbf{c}), \mathrm{v}(t, \mathbf{c}), t) \tag{1}$$

Training Framework:

- Compute $\mathrm{v}(t, \mathbf{c})$ from all observable trajectories $\mathrm{u}(t, \mathbf{c})$.
- Train a neural network f_θ to learn $\mathrm{u}(t, \mathbf{c})$ from $\mathrm{v}(t, \mathbf{c})$:

$$\mathrm{u}(t, \mathbf{c}) = f_\theta(\mathrm{v}(t, \mathbf{c}), t, \mathbf{c})$$

- This defines a new ODE:

$$\frac{\mathrm{dv}(t, \mathbf{c})}{\mathrm{d}t} = g(f_\theta(\mathrm{v}(t, \mathbf{c}), t, \mathbf{c}), \mathrm{v}(t, \mathbf{c}), t) = g_\theta(\mathrm{v}(t, \mathbf{c}), t, \mathbf{c})$$

Step 1: Operator preprocessing:

- Compute $\mathrm{v} = \mathcal{F}(\mathrm{u})$ from training data.

Step 2: Learning the ODE:

- Train neural network f_θ to minimize the discrepancy between observed trajectories $\mathrm{u}(t, \mathbf{c})$ and predicted trajectories $f_\theta(\mathrm{v}(t, \mathbf{c}))$:

$$\min_{f_\theta} \|\mathrm{u}(t, \mathbf{c}) - f_\theta(\mathrm{v}(t, \mathbf{c}))\|$$

  This is a classic regression problem

Step 3: Evaluation:

- Solve the ODE defined by g_θ to compute the predicted auxiliary function $\widehat{\mathrm{v}}(t, \mathbf{c})$.
- Estimate the trajectory $\widehat{\mathrm{u}}(t, \mathbf{c})$ as $\widehat{\mathrm{u}}(t, \mathbf{c}) = f_\theta(\widehat{\mathrm{v}}(t, \mathbf{c}))$.

Preliminary Experiment Results

- $u(z, c)$ is the evolution of a beam in a linear particle accelerator
- z is the longitudinal position of the beam (equivalent of$t$ in the equations above)
- $c \in \mathbb{R}^{108}$ is a control parameter of the simulation

Goal:

- Learn $c \to u(\cdot, c)$

Method:

- Learn $\widehat{f}$ s.t. $u(z, c) = \widehat{f}\left(\int_{s=0}^{z} u(s, c)\, \mathrm{d}s, z, c, \theta\right)$

Dataset

- Linac dataset **ThomX-2024** contains 4000 simulations generated by Astra.
- Control settings dimension ($\mathbf{c}$) is 36 **Purwar-2023**.
- Each trajectory consists of 4000 points: $(t_j, \mathrm{u}(t_j, \mathbf{c}_j))$, $t_j \in [0, 9.393]$.

Experiment Set-up

- Compare three variants against baselines (LSTM, NODE).

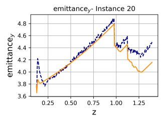
Performance Measurement

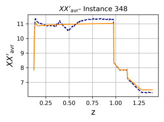
- Use the coefficient of determination R^2 to evaluate model performance:

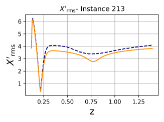
$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Preliminary Results

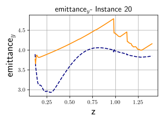
LSTM

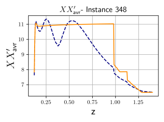


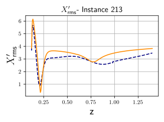




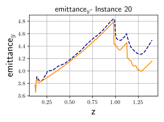
NODE

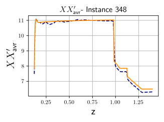


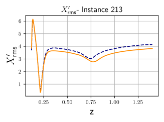




INODE







Emittance over time | XX'_{avr} over time | X'_{ms} over time

(The orange curve represents the ground truth trajectory)

Conclusion

INode Framework:

- Extends Neural ODEs to handle systems with discontinuous behavior.
- Efficient for data-driven ODEs, avoiding the need to directly solve complex ODEs during training.
- Uses integral operators to process input data, addressing challenges of discontinuities. **Other families of operators could be explored !**

Key Results:

- Theoretical guarantees
- Improved computational efficiency
- Provides a foundational approach that can be extended for future work

Questions ?

Thanks you for your attention

Bibliography

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As there was some unprocessed data that should have been added to the final page this extra page has been added to receive it.

If you rerun the document (without altering it) this surplus page will go away, because LaTeX now knows how many pages to expect for this document.