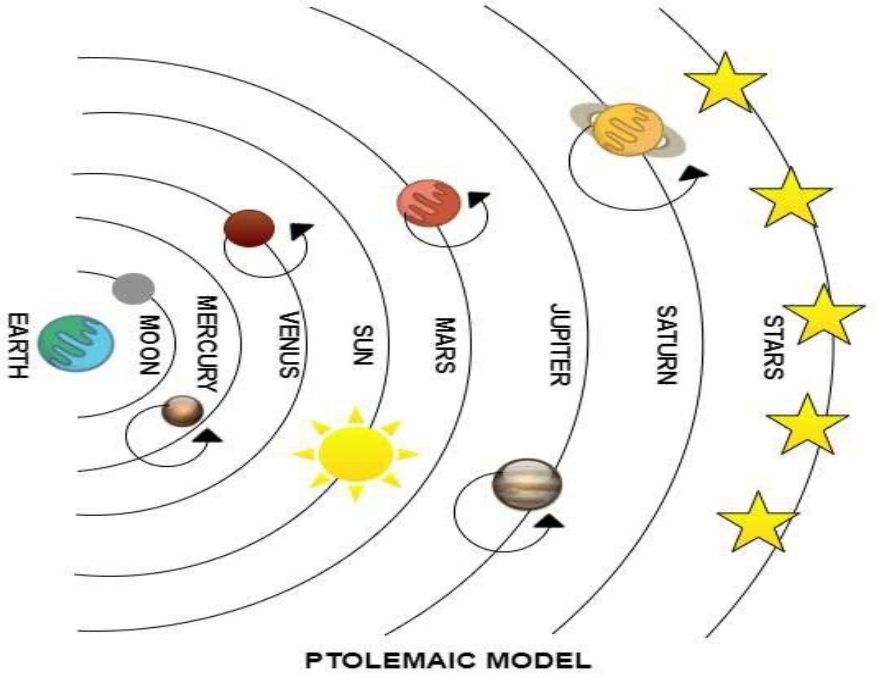
Lecture 5. Reconstructing dynamics

Chris Budd OBE and Aengus Roberts

Ai4Sci, December 2025



Knowledge Exchange Hub for Mathematical Sciences

UNIVERSITY OF BATH

Some papers/books to look at

- Chen et. al., *Neural Ordinary Differential Equations*
- Matlab and Simulink, *Dynamical System Modelling Using Neural ODE*
- Brunton, Kutz et. al, *Discovering governing equations from data by sparse identification of nonlinear dynamical systems*
- Ashwin et. al. *RNNS*

m4DL

UNIVERSITY OF BATH

Dynamical systems

Flows:

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t, \theta)$$

Maps:

$$\mathbf{x}_{n+1} = \mathbf{f}(\mathbf{x}_n, \theta)$$

$x(0), \quad x_0$ Given. Study the evolution from this state

m4DL

UNIVERSITY OF BATH

Dynamical systems link to neural networks in a number of ways

- Solving a dynamical system using a PINN .. See lecture 3
- Constructing a dynamical system operator using a Neural Operator .. See lecture 4
- Approximating the flow through a NN as a dynamical system (Neural ODE 1) [Chen et. al.] and train that dynamical system
- Using a NN to approximate/learn the RHS of a dynamical system by

$$\frac{du}{dt} = f(\mathrm{t}, \mathrm{u}, \theta)$$

where f is the output of a NN and learn the parameters θ from data (Neural ODE 2)

- Take f to be symbolic and learn it from data. SINDy [Brunton et. al.]
- Predict the dynamics of a dynamical system using a LSTM or similar

m4DL

UNIVERSITY OF BATH

Neural ODES 1 [Chen et. al] : The ResNET architecture

$$x_{n+1} = x_n + \Delta t \sum_i c_i^n \sigma(A_i^n x^n + b_i^n)$$

Train this to do a specific task

Input: x_0 Output: x_N

m4DL

UNIVERSITY OF BATH

Let $t = n\,\Delta t$

$$\frac{x_{n+1} - x_n}{\Delta t} = \sum_i c_i(t)\ \sigma(A_i(t)x^n + b_i(t))$$

Let $\Delta t \to 0$

$$\frac{dx}{dt} = \sum_i c_i(t)\ \sigma(A_i(t)x^n + b_i(t))$$

Neural ODE. Input: x(0). Output: x(T), $T = n\,\Delta t$

- Idea [Chen et. al.]:
- Replace the NN by solving the Neural ODE using an ODE solver eg. RK4, Adam
- Training now becomes a question of finding the functions A(t), b(t), c(t)
- Advantages in memory efficiency and back propagation
- This a problem in optimal control theory

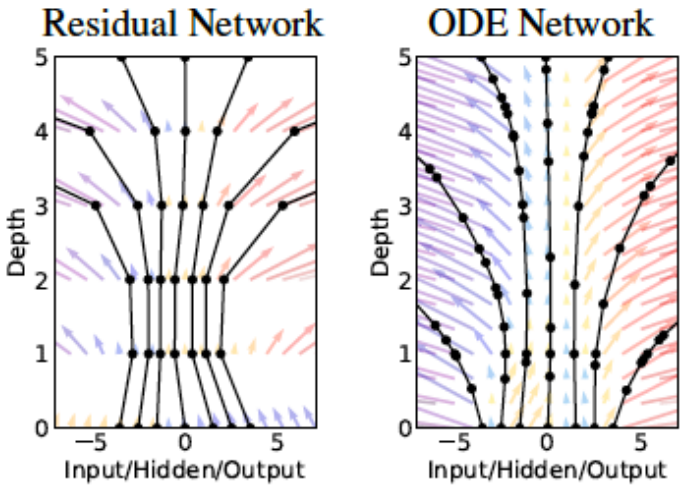


Figure 1: *Left:* A Residual network defines a discrete sequence of finite transformations. *Right:* A ODE network defines a vector field, which continuously transforms the state. *Both:* Circles represent evaluation locations.

Training such a ‘Neural ODE’. [Chen et. al 19], [Pontryagin 62]

Consider the parametrized Neural ODE

$$\frac{d\mathbf{h}}{dt} = f(\mathbf{h}(t), t, \theta), \quad \mathbf{h}(0) \rightarrow \mathbf{h}(T).$$

Compute solution using an ODE solver

Loss function L(.)

$$L(\mathbf{z}(T)) = L\left(\mathbf{z}(0) + \int_0^T f(\mathbf{z}(t), t, \theta)\, dt\right)$$

m4DL

UNIVERSITY OF BATH

To train the neural net to optimize L we need to find how it changes with θ

Results from classical control theory (and data assimilation) allow us to do this

Step 1

Define adjoint $\mathbf{a}(t) = \partial L / \partial \mathbf{z}(t)$ then

$$\frac{d\mathbf{a}}{dt} = -\mathbf{a}^T \, \partial f / \partial \mathbf{z}$$

Compute a(t) using an ODE solver 'backwards' starting from z(T)

m4DL

UNIVERSITY OF BATH

Step 2

Compute the gradient of L by solving a third ODE

$$\frac{dL}{d\theta} = -\int_T^0 \mathbf{a}(t)^T \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} \, dt$$

Can then train θ to optimize L using gradient descent

[Chen et. al.] implement this using an adaptive Adam solver.

Dynamical systems defined by Neural Nets and learning dynamics from data

Original DS: $$\frac{dx}{dt} = f(x) \qquad x \in R^n$$

Data points: $$y_i = H(x(t_i)) + \epsilon_i, \quad i = 1 \ldots N$$

Train: a NN to approximate this dynamics via:

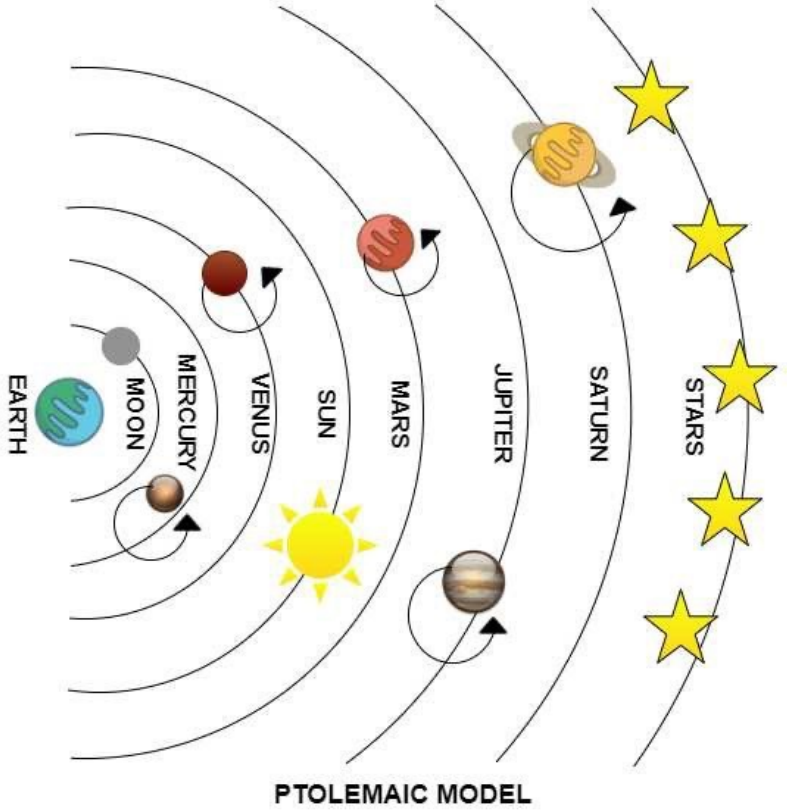
$$\frac{dz}{dt} = f(z, t, \theta) : \quad \text{f: Neural Net}$$

Also (and confusingly) called a Neural ODE eg. by Matlab

m4DL

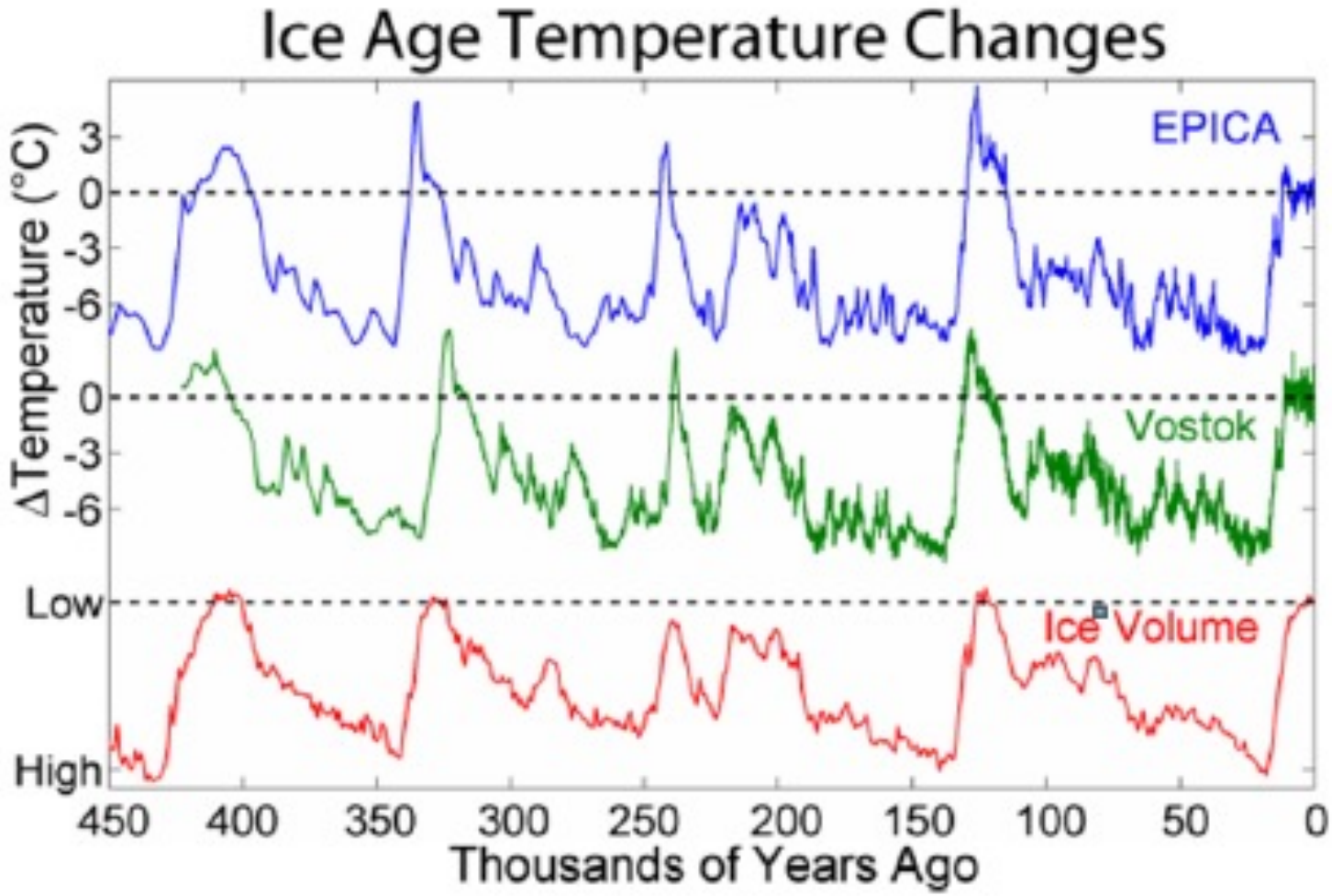
UNIVERSITY OF BATH

Big literature on these systems



- Ptolemaic model of the solar system
- Standard learning
- Echo State Networks, Reservoir networks
  Especially good for chaotic dynamics
- SINDy approach (Kutze) using a very careful choice of RHS (can even work from video data)

Q. What caused the ice ages?



Training such a ‘Neural ODE’. I [From MatLab website]

Example:

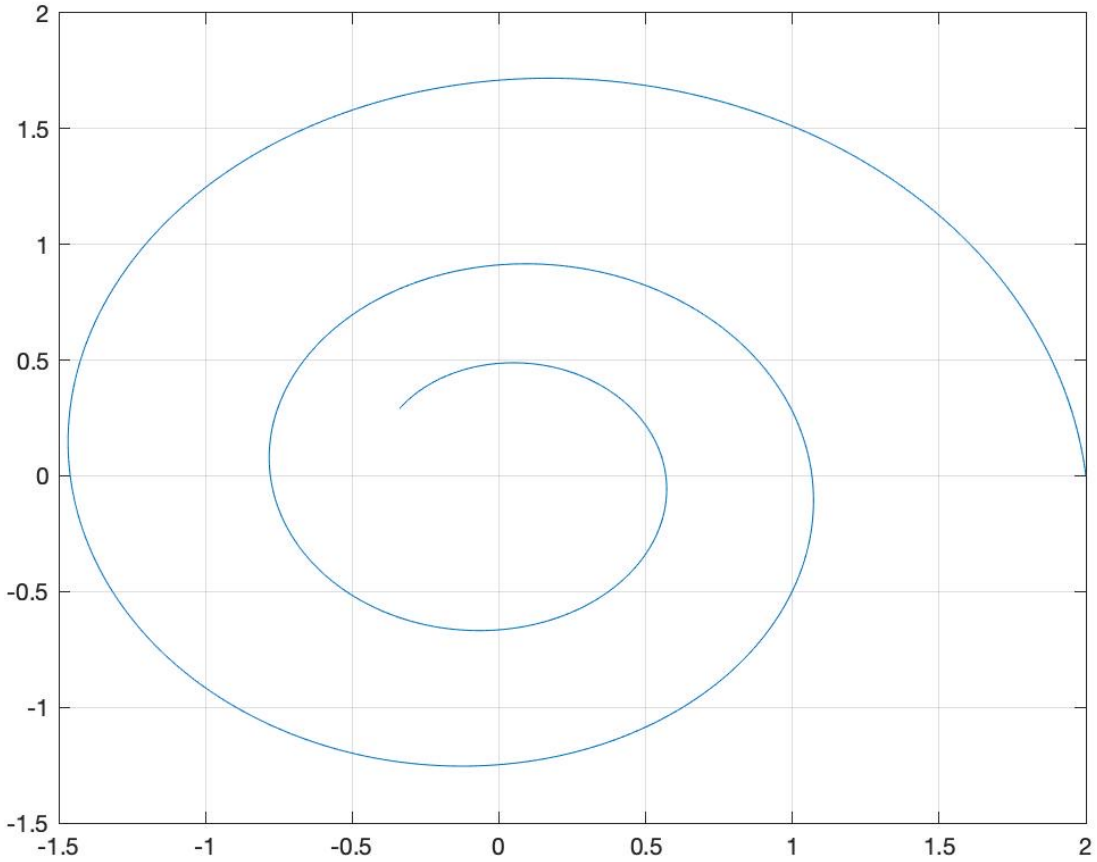
$$\frac{d\mathbf{x}}{dt} = A\mathbf{x}, \quad \mathbf{x} \in R^2$$

$$\mathbf{x}(t) = e^{At}\,\mathbf{x}(0) \equiv B\mathbf{x}(0).$$

Neural ODE: $\dfrac{dz}{dt} = f(z, t, \theta), \quad z(0) = x(0), \quad$ f: Neural Net

Solve: using RK45

Training trajectory starting from: x(0) = [2,0], t = [0,15], 2000 points



$$A = \begin{pmatrix} 0.1 & -1 \\ 1 & -0.1 \end{pmatrix}$$

Define loss: $L(\theta)$

$$L(\theta) = \sum_i |H(z(t_i)) - y_i|^2$$

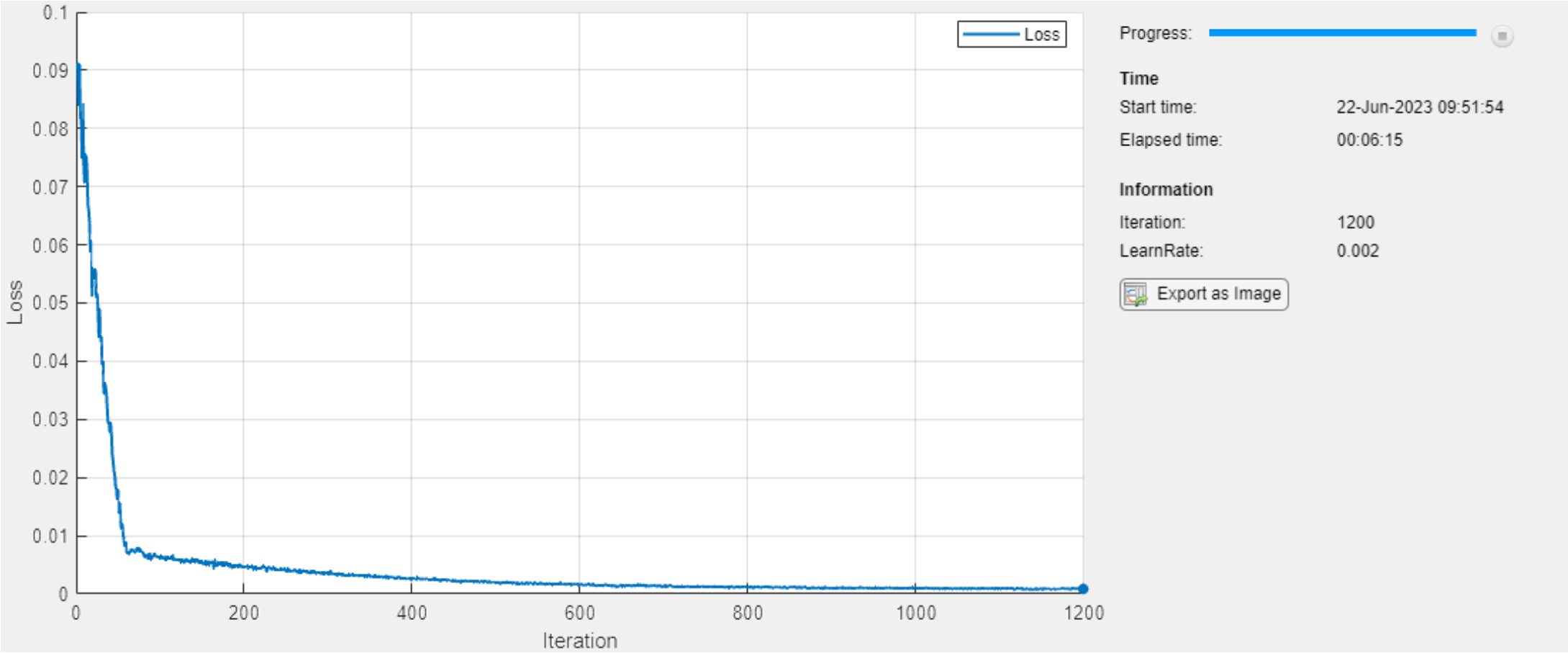
- Implement NN using 2 layers and tanh activation

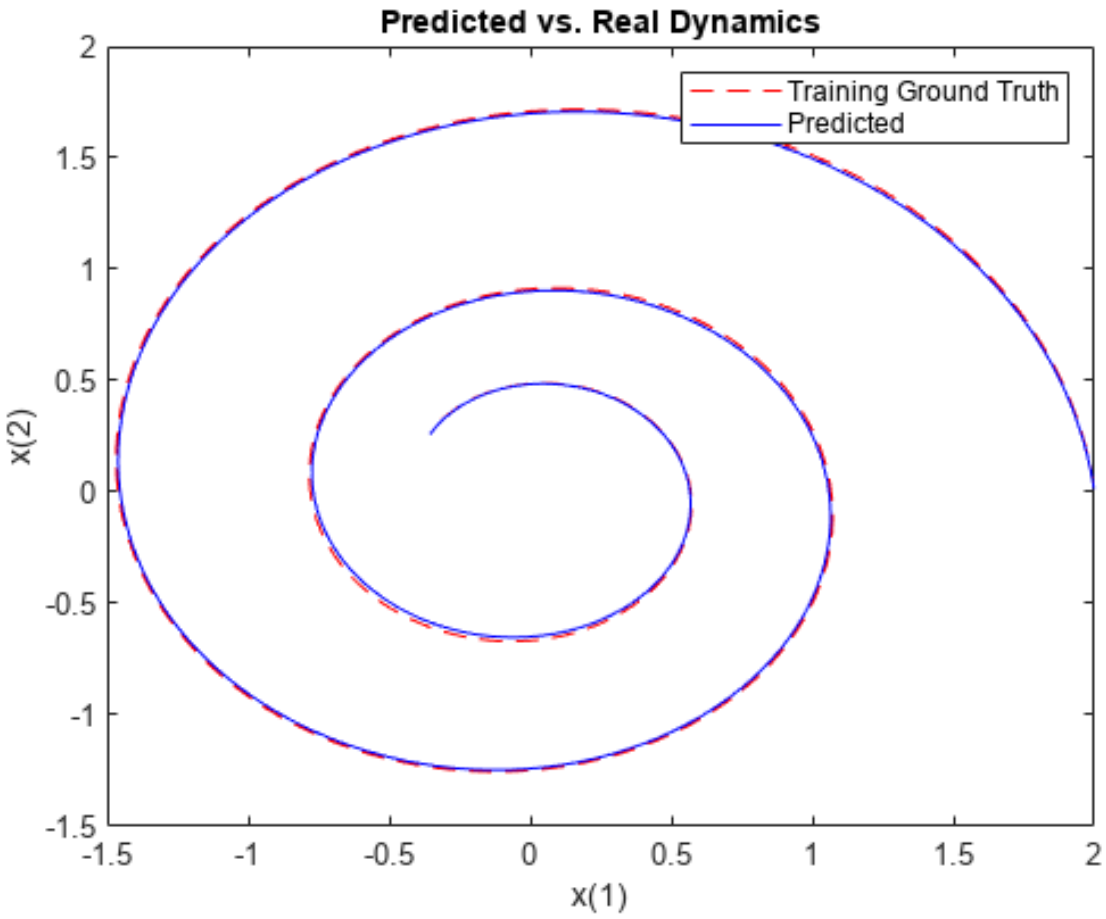
$$f(\mathbf{z}, \theta) = W_2 \ \tanh\left(W_1 \ \mathbf{z} + \mathbf{b}_1\right) + \mathbf{b}_2.$$

- Compute gradients directly from the ODE solver using the adjoint method as before
- Optimize using ADAM using first 400 training points

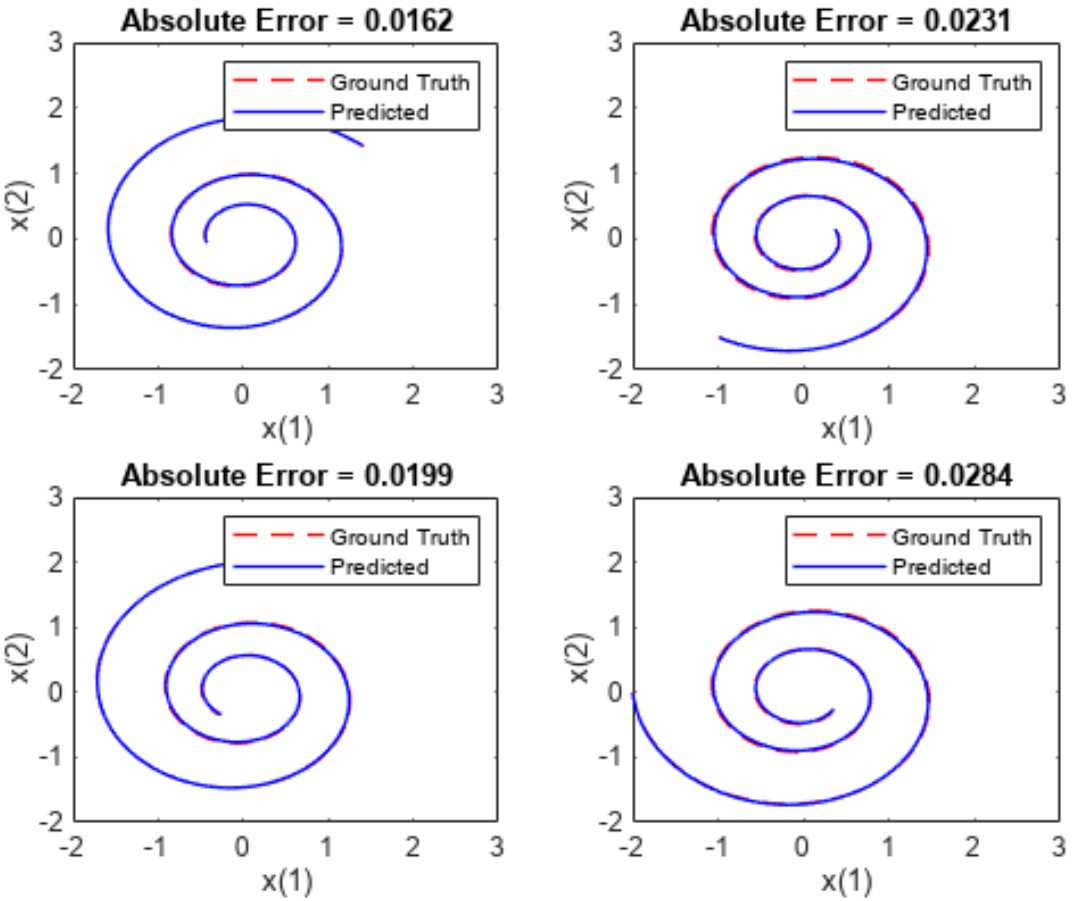
m4DL

UNIVERSITY OF BATH

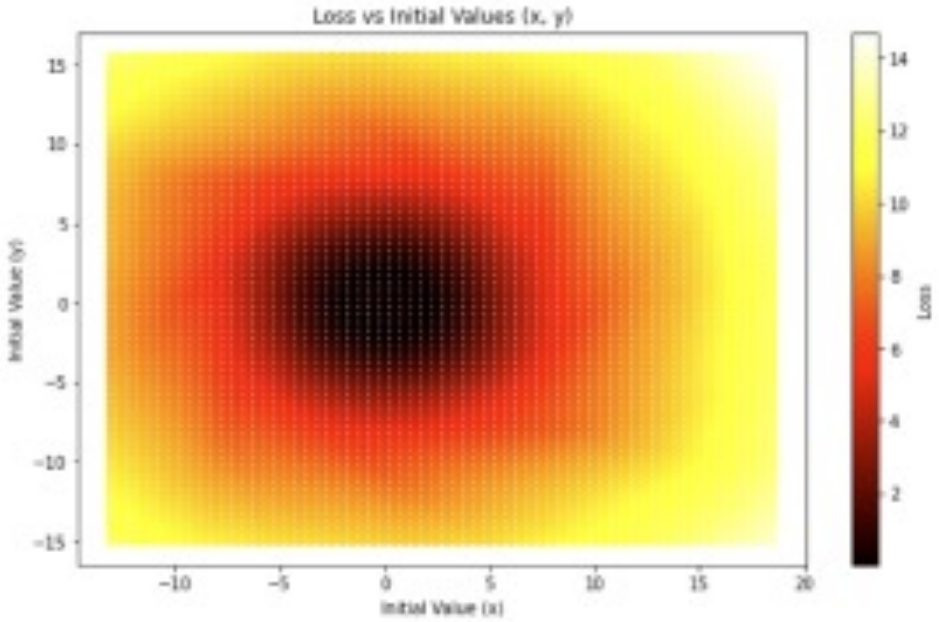

Loss
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
Loss
0
200
400
600
800
1000
1200
Iteration
Progress:
Time
Start time: 22-Jun-2023 09:51:54
Elapsed time: 00:06:15
Information
Iteration: 1200
LearnRate: 0.002
Export as Image

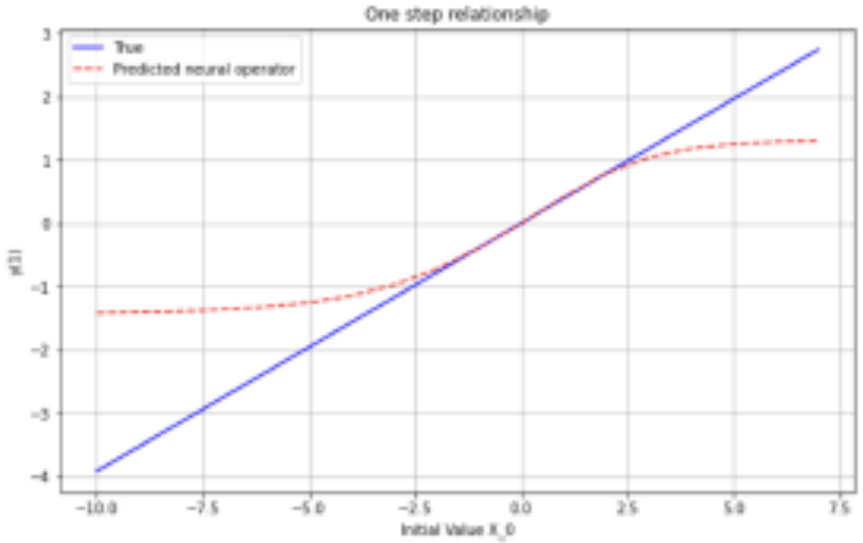

Predicted vs. Real Dynamics
Training Ground Truth
Predicted
x(1)
x(2)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-1.5
-1
-0.5
0
0.5
1
1.5
2

Prediction error is good for initial data close to the training trajectory



Loss increases as we move away from the training trajectory [+Libo Chen]





By way of contrast:

Assume that $$\frac{d\mathbf{z}}{dt} = B\,\mathbf{z}$$

Find matrix B

100 points: $$B = \begin{pmatrix} -0.1001 & -1.0005 \\ 1.0004 & -0.0993 \end{pmatrix}$$

400 points: $$B = \begin{pmatrix} -0.1000 & -1.0005 \\ 1.0007 & -0.1002 \end{pmatrix}$$

2000 points: $$B = \begin{pmatrix} -0.1000 & -1.0005 \\ 1.0005 & -0.1001 \end{pmatrix}$$

m4DL

UNIVERSITY OF BATH

SINDy [Brunton et. al.]

ODE: $$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x})$$

Data: $\mathbf{x}(t_i), \quad \dot{\mathbf{x}}(t_i)$ Plus noise: m measurements

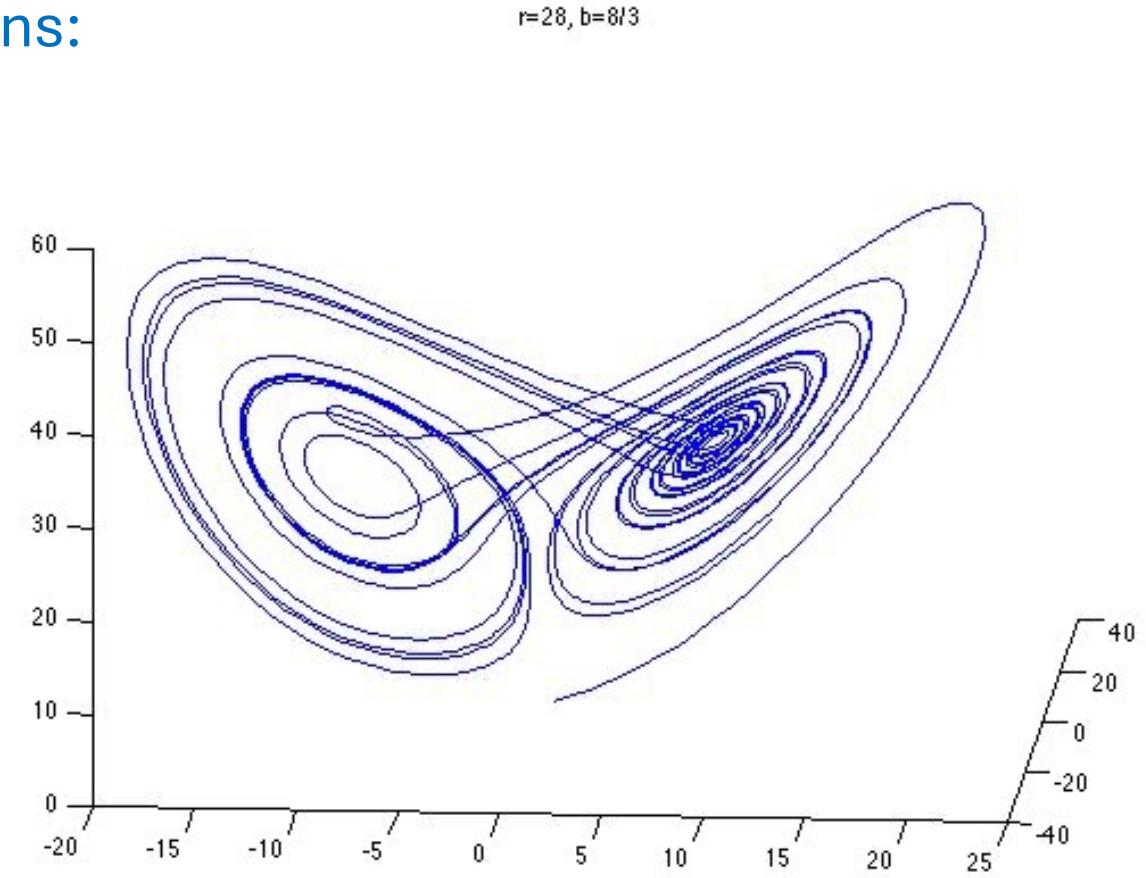
Assume: f(x) can be constructed from a library of known functions

m4DL

UNIVERSITY OF BATH

Eg. Lorenz Equations:

$$dx/dt = \sigma(y - x),$$
$$dy/dt = x(\rho - z) - y,$$
$$dz/dt = xy - \beta z.$$



Library: $\Theta(\mathbf{x}) = \left[\mathbf{1} \mid \mathbf{x} \mid \mathbf{x}^{P_2} \mid \mathbf{x}^{P_3} \mid \sin(\mathbf{x}) \mid \cos(\mathbf{x}) \mid \ldots\right]$

Size. m x p m >> p

Coefficients (sparse vectors): $\Xi = \left[\xi_1 \; \xi_2 \; \ldots \; \xi_n\right]$

Seek a SPARSE approximation of the form:

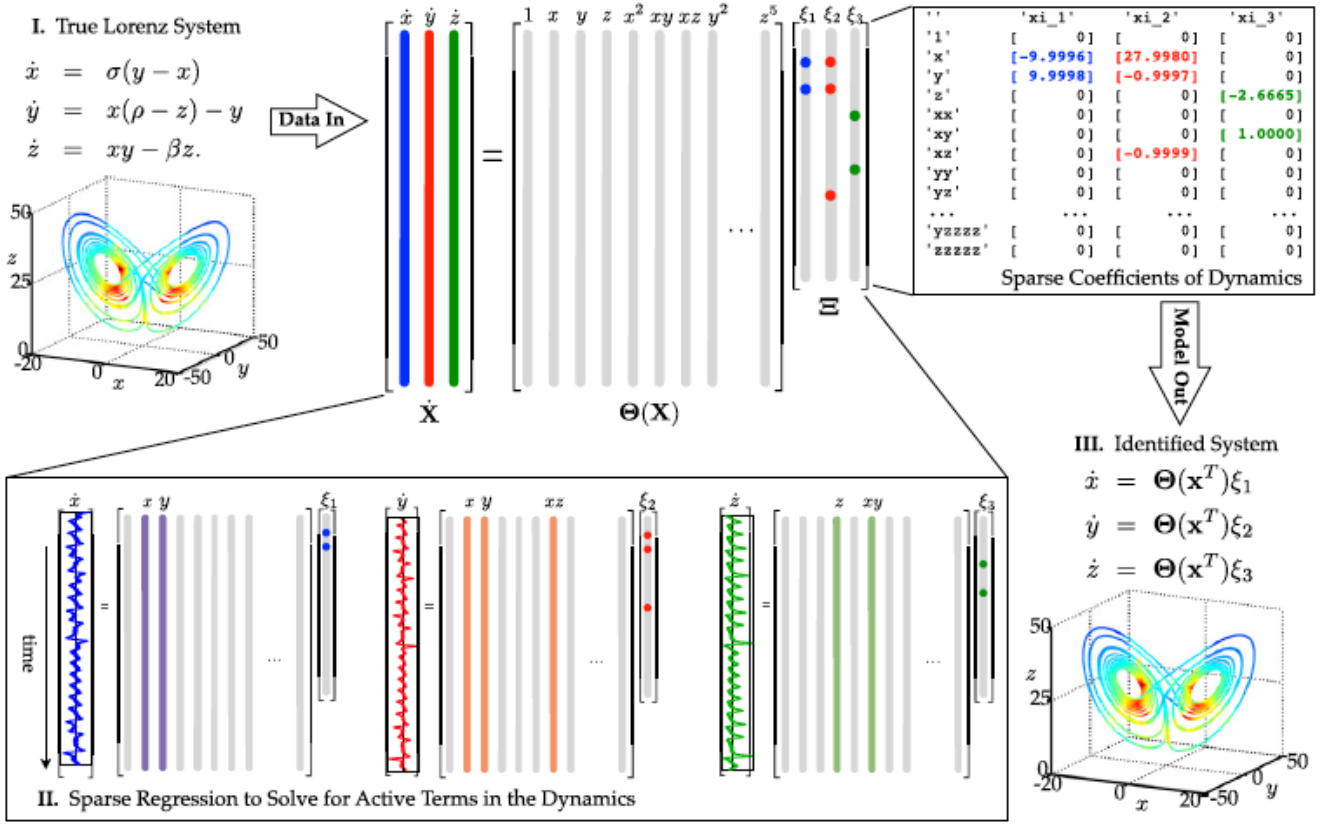
$$\dot{\mathbf{x}} = \Theta(\mathbf{x})\, \Xi$$

m4DL

UNIVERSITY OF BATH

Aim: Find best set Ξ to fit to (filtered, noisy) data

Possibly with numerically estimated derivatives



From Brunton et. el.

Conclusions

Neural ODEs are effective for

- Describing the flow through a NN
- Learning dynamics from data
- They inherently non smooth (or close to non smooth)
- They work best when informed by the physics