Neural Ordinary Differential Equations With Trainable Solvers

Said Ouala[1], Laurent Debreu, Bertrand Chapron,

Fabrice Collard, Lucile Gaultier, Ronan Fable

1) IMT Atlantique, Lab-STICC, Brest/ INRIA team ODYSSEY, France;

Outline

- Introduction to Neural ODEs
- Training Neural ODEs, example on learning dynamical systems
- Trainable solvers for (N)ODEs
- Applications
- Conclusion and perspectives

Introduction to Neural ODEs

Introduction to Neural ODEs

- Neural Ordinary Differential equations are differential equations where the vector field is a neural network

$$\dot{\mathbf{z}}_t = f_\theta(t, \mathbf{z}_t)$$

Introduction to Neural ODEs

- Neural Ordinary Differential equations are differential equations where the vector field is a neural network

$$\dot{\mathbf{z}}_t = f_{\boldsymbol{\theta}}(t, \mathbf{z}_t)$$

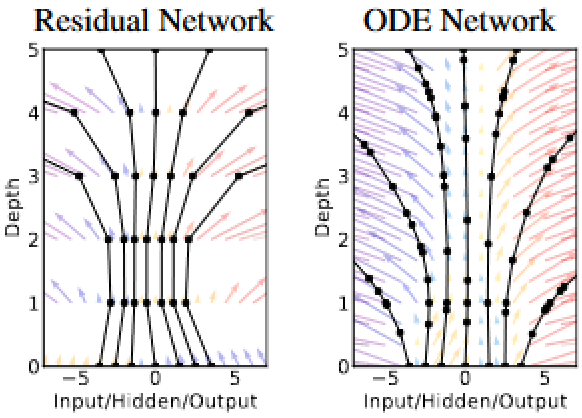
- They offer a system theoretic viewpoint on the study and design of neural networks in various applications including

Introduction to Neural ODEs

- Neural Ordinary Differential equations are differential equations where the vector field is a neural network

$$\dot{\mathbf{z}}_t = f_\theta(t, \mathbf{z}_t)$$

- They offer a system theoretic viewpoint on the study and design of neural networks in various applications including



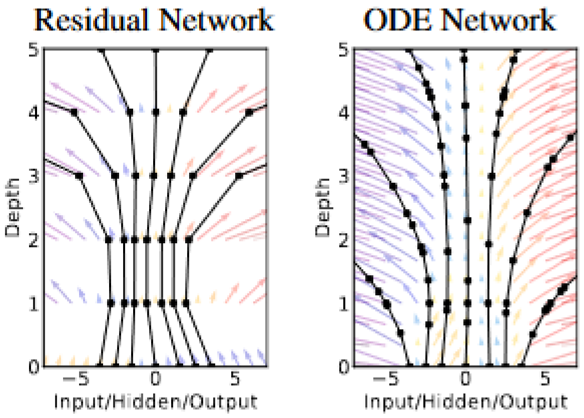
Residual networks with adaptive depth

Introduction to Neural ODEs

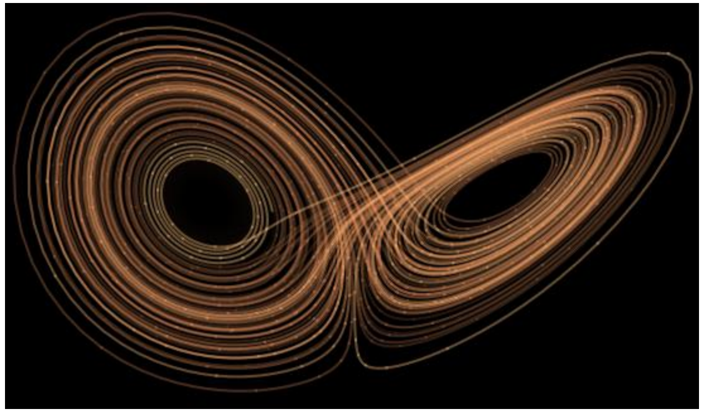
- Neural Ordinary Differential equations are differential equations where the vector field is a neural network

$$\dot{\mathbf{z}}_t = f_\theta(t, \mathbf{z}_t)$$

- They offer a system theoretic viewpoint on the study and design of neural networks in various applications including



Residual networks with adaptive depth



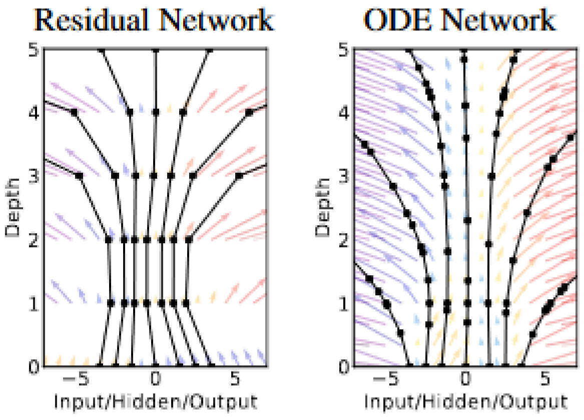
Modeling of time series and dynamical systems

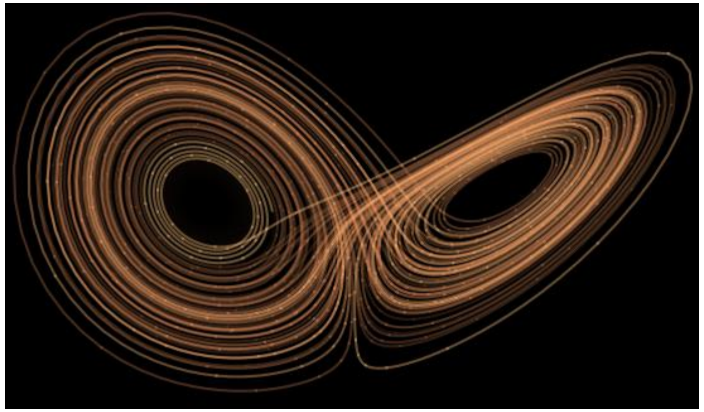
Introduction to Neural ODEs

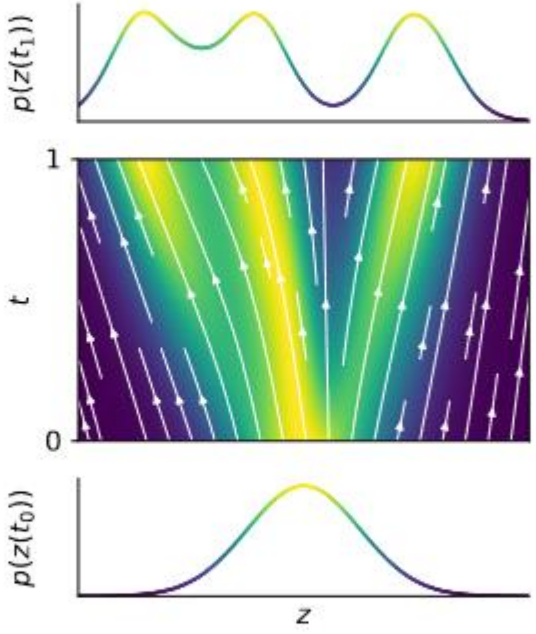
- Neural Ordinary Differential equations are differential equations where the vector field is a neural network

$$\dot{\mathbf{z}}_t = f_\theta(t, \mathbf{z}_t)$$

- They offer a system theoretic viewpoint on the study and design of neural tions including







Residual networks with adaptive depth

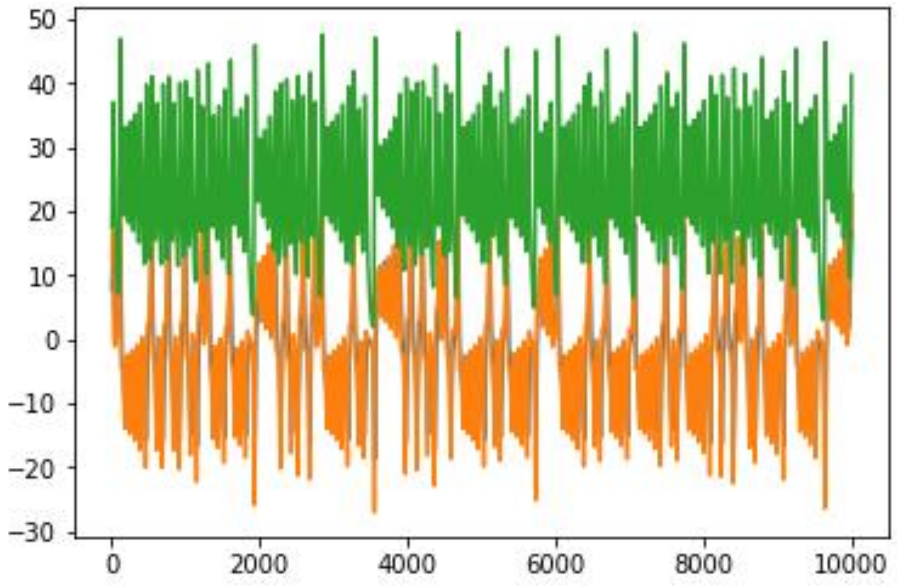
Modeling of time series and dynamical systems

Density estimation and generative modeling

Training Neural ODEs, example on learning dynamical systems

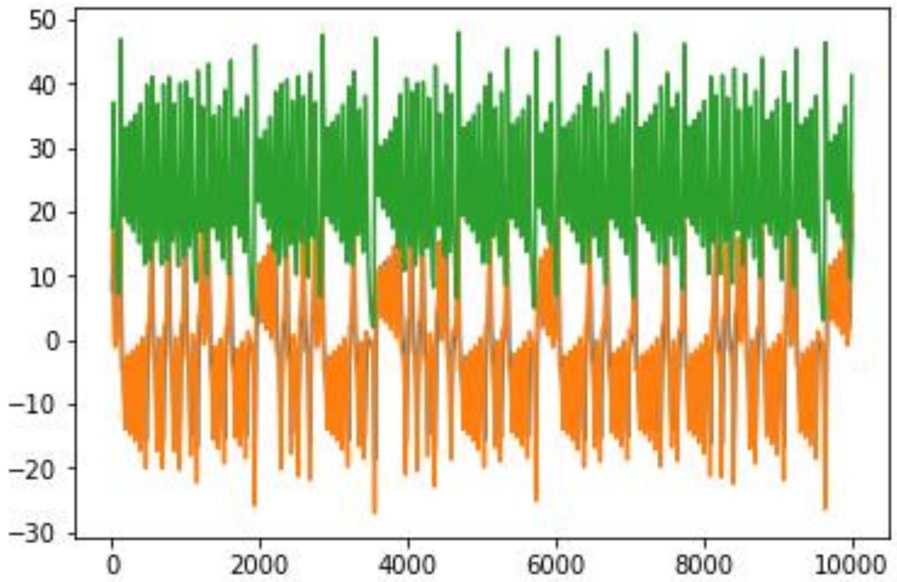
Training Neural ODEs, example on learning dynamical systems

- Let us assume that we are given measurements of a time varying dynamical system



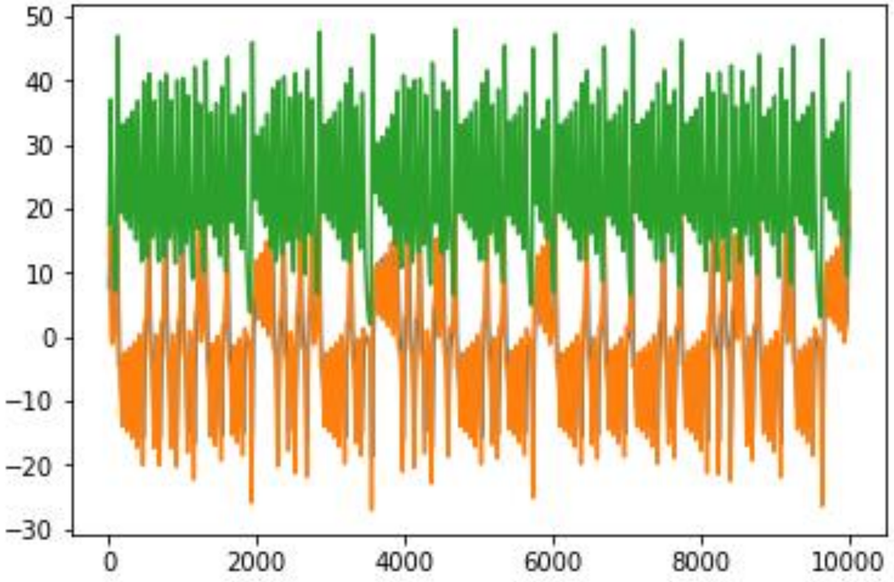
Training Neural ODEs, example on learning dynamical systems

- Let us assume that we are given measurements of a time varying dynamical system
- Assuming a parameterization for the data-driven model $\dot{\mathbf{z}}_t = f_{\boldsymbol{\theta}}(t, \mathbf{z}_t)$



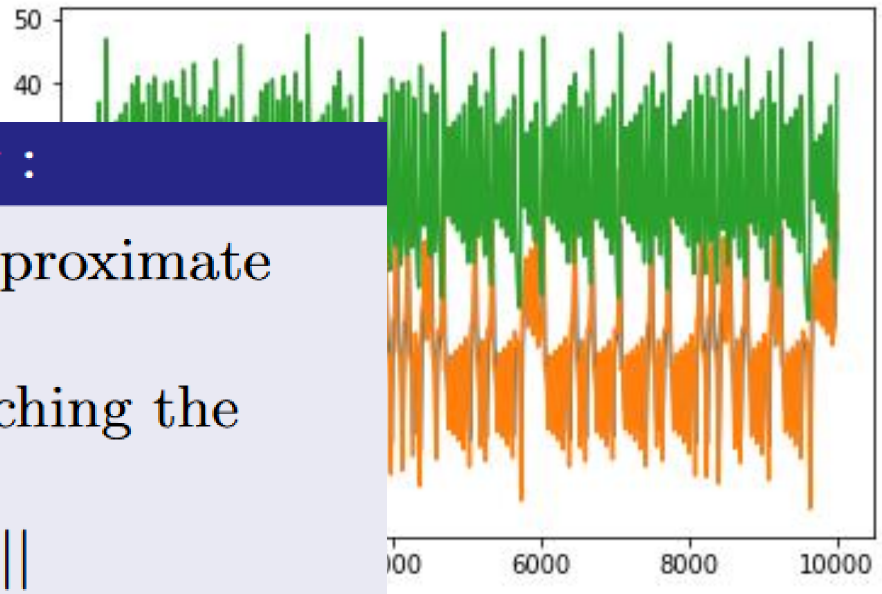
Training Neural ODEs, example on learning dynamical systems

- Let us assume that we are given measurements of a time varying dynamical system
- Assuming a parameterization for the data-driven model $\dot{\mathbf{z}}_t = f_{\boldsymbol{\theta}}(t, \mathbf{z}_t)$
- How to compute the parameters θ?



Training Neural ODEs, example on learning dynamical systems

- Let us assume that we are given measurements of a time varying dynamical system
- Assuming a parameterization for the data-driven model $\dot{\mathbf{z}}_t = f_{\boldsymbol{\theta}}(t, \mathbf{z}_t)$
- How to compute the parameters θ?

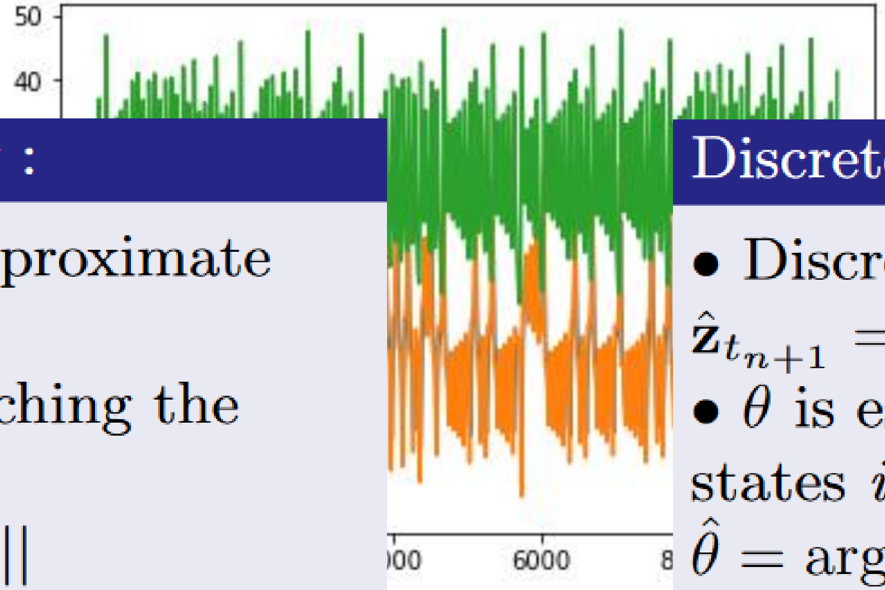


Continuous time setting :

- Direct work on the approximate ODE : $\dot{\mathbf{z}}_t = f_\theta(\mathbf{z}_t)$
- θ is estimated by matching the derivatives *i.e.* $\hat{\theta} = \arg\min_\theta ||f_\theta(\mathbf{z}_t) - \hat{\dot{\mathbf{z}}}_t||$
- Issue : How to estimate $\hat{\dot{\mathbf{z}}}_t$?

Training Neural ODEs, example on learning dynamical systems

- Let us assume that we are given measurements of a time varying dynamical system
- Assuming a parameterization for the data-driven model $\dot{\mathbf{z}}_t = f_{\boldsymbol{\theta}}(t, \mathbf{z}_t)$
- How to compute the parameters θ?



Continuous time setting :

- Direct work on the approximate ODE : $\dot{\mathbf{z}}_t = f_\theta(\mathbf{z}_t)$
- θ is estimated by matching the derivatives *i.e.* $\hat{\theta} = \arg\min_\theta ||f_\theta(\mathbf{z}_t) - \hat{\dot{\mathbf{z}}}_t||$
- Issue : How to estimate $\hat{\dot{\mathbf{z}}}_t$?

Discrete time setting :

- Discretized version : $\hat{\mathbf{z}}_{t_{n+1}} = \Phi_{\mathcal{D},t_{n+1}}(\mathbf{z}_{t_n})$
- θ is estimated by matching the states *i.e.* $\hat{\theta} = \arg\min_\theta ||\mathbf{z}_t - \Phi_{\mathcal{D},t_{n+1}}(\mathbf{z}_{t_n})||$
- Issue : which $\Phi_{\mathcal{D},t_{n+1}}$?

Training Neural ODEs, example on learning dynamical systems

- Choosing an integration scheme is an issue even in ODE integration applications.

Training Neural ODEs, example on learning dynamical systems

- Choosing an integration scheme is an issue even in ODE integration applications.

- Need to make sure that the integration is stable with respect to the ODE and the time step.

Training Neural ODEs, example on learning dynamical systems

- Choosing an integration scheme is an issue even in ODE integration applications.

- Need to make sure that the integration is stable with respect to the ODE and the time step.

- Solution : black box adaptive step-size solvers ?

Training Neural ODEs, example on learning dynamical systems

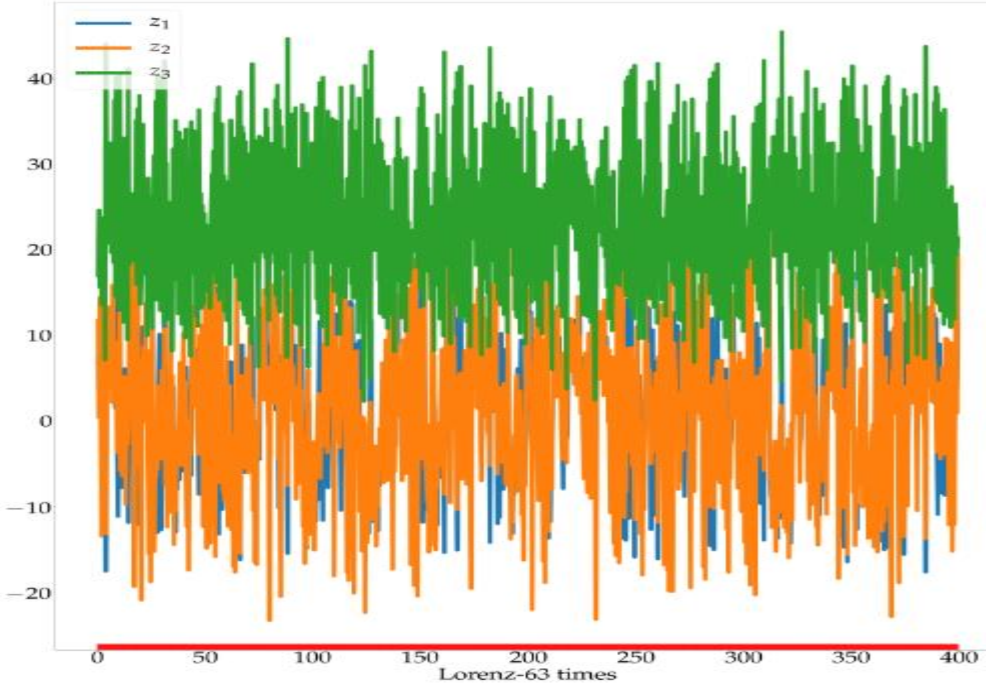
- Choosing an integration scheme is an issue even in ODE integration applications.
- Need to make sure that the integration is stable with respect to the ODE and the time step.
- Solution : black box adaptive step-size solvers ?
- Adaptive step-size solvers in identification applications (chen et al. 2018) can be subject to memory/instability issues.

Training Neural ODEs, example on learning dynamical systems

Training Neural ODEs, example on learning dynamical systems

- Example: model measurements of the Lorenz 63 dynamical system, the data are sampled at a (sparse) sampling rate of dt = 0.4

$$\left\{\begin{array}{lll} \frac{dz_{t,1}}{dt} & = & \sigma\left(z_{t,2} - z_{t,2}\right) \\ \frac{dz_{t,2}}{dt} & = & \rho z_{t,1} - z_{t,2} - z_{t,1} z_{t,3} \\ \frac{dz_{t,3}}{dt} & = & z_{t,1} z_{t,2} - \beta z_{t,3} \end{array}\right.$$



Training Neural ODEs, example on learning dynamical systems

- Example: model measurements of the Lorenz 63 dynamical system, the data are sampled at a (sparse) sampling rate of dt = 0.4

Training Neural ODEs, example on learning dynamical systems

- Example: model measurements of the Lorenz 63 dynamical system, the data are sampled at a (sparse) sampling rate of dt = 0.4

- Fit the NODE parameters to minimize the forecast of the training data : $\underset{\boldsymbol{\theta}}{\text{Minimize}} \quad \mathcal{J}(\Psi_{\boldsymbol{\phi}}(\cdot))$

Training Neural ODEs, example on learning dynamical systems

• Example: model measurements of the Lorenz 63 dynamical system, the data are sampled at a (sparse) sampling rate of dt = 0.4

Adaptive solver

• Fit the NODE parameters to minimize the forecast of the training data : $\underset{\boldsymbol{\theta}}{\text{Minimize}} \quad \mathcal{J}(\Psi_{\boldsymbol{\phi}}(\cdot))$

• Using adaptive solvers results in a substantial increase in the NFE as the ODE fits the training data

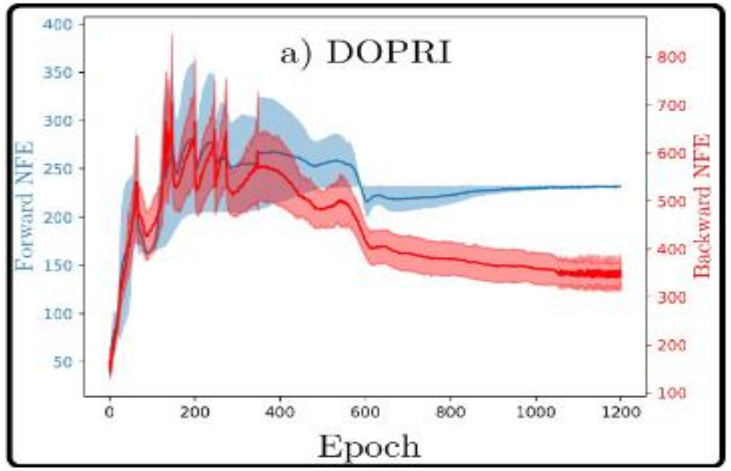
Training Neural ODEs, example on learning dynamical systems

- Example: model measurements of the Lorenz 63 dynamical system, the data are sampled at a (sparse) sampling rate of dt = 0.4

Adaptive solver

- Fit the NODE parameters to minimize the forecast of the training data : $\underset{\boldsymbol{\theta}}{\text{Minimize}} \quad \mathcal{J}(\Psi_{\boldsymbol{\phi}}(\cdot))$

- Using adaptive solvers results in a substantial increase in the NFE as the ODE fits the training data



Training Neural ODEs, example on learning dynamical systems

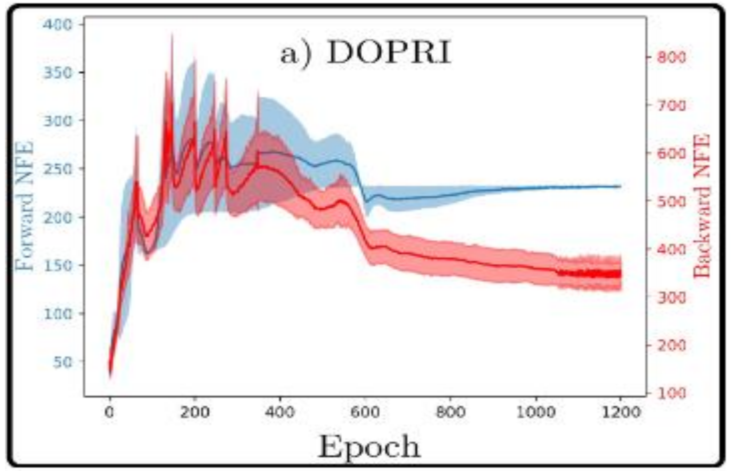
- Example: model measurements of the Lorenz 63 dynamical system, the data are sampled at a (sparse) sampling rate of dt = 0.4

Adaptive solver

- Fit the NODE parameters to minimize the forecast of the training data : $$\underset{\boldsymbol{\theta}}{\text{Minimize}} \quad \mathcal{J}(\Psi_{\boldsymbol{\phi}}(\cdot))$$

- Using adaptive solvers results in a substantial increase in the NFE as the ODE fits the training data

- What if we allow the numerical scheme to adapt to the dynamics and to be trainable ?



Training Neural ODEs, example on learning dynamical systems

- Example: model measurements of the Lorenz 63 dynamical system, the data are sampled at a (sparse) sampling rate of dt = 0.4

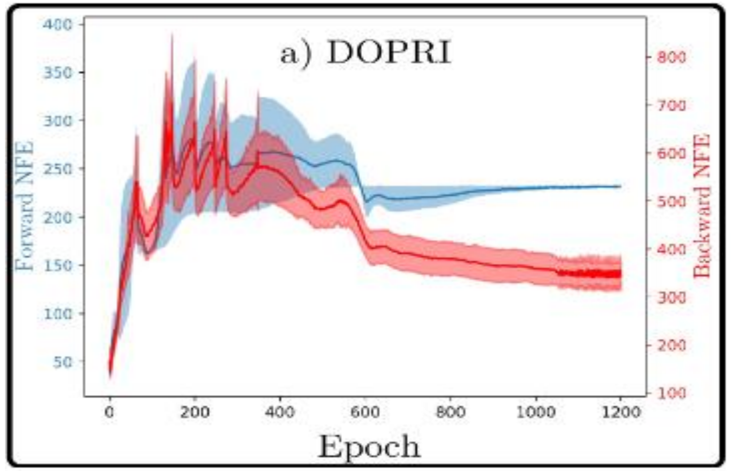
Adaptive solver

- Fit the NODE parameters to minimize the forecast of the training data : $\underset{\boldsymbol{\theta}}{\text{Minimize}} \quad \mathcal{J}(\Psi_{\phi}(\cdot))$

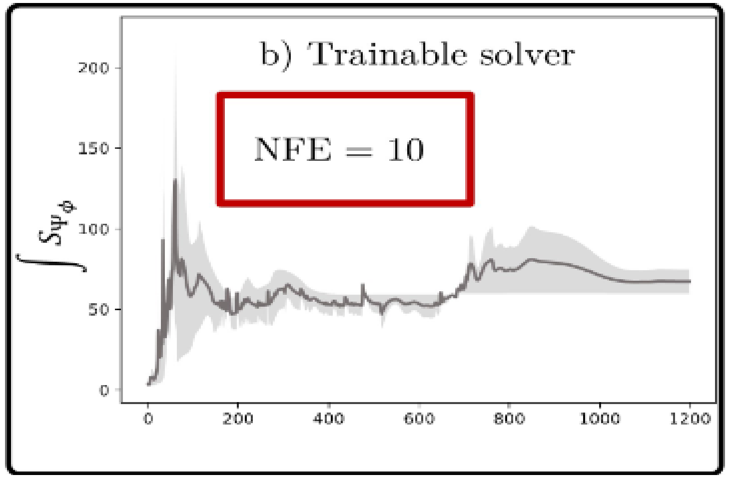
Trainable solver ?

- Using adaptive solvers results in a substantial increase in the NFE as the ODE fits the training data

- What if we allow the numerical scheme to adapt to the dynamics and to be trainable ?



The stability region increases at a fixed NFE!



Trainable Solvers for (N)ODEs

Trainable Solvers for (N)ODEs

- Let us assume a continuous time (Neural) ODE:

$$\dot{\mathbf{z}}_t = f_{\boldsymbol{\theta}}(t, \mathbf{z}_t)$$

Trainable Solvers for (N)ODEs

- Let us assume a continuous time (Neural) ODE:

$$\dot{\mathbf{z}}_t = f_{\boldsymbol{\theta}}(t, \mathbf{z}_t)$$

- The solution of this equation is computed using a Runge Kutta scheme with q stages

$$\begin{cases} \hat{\mathbf{z}}_{t_{n+1}} = \Psi_{\phi_q}(\hat{\mathbf{z}}_{t_n}) = \hat{\mathbf{z}}_{t_n} + h \sum_{i=1}^{q} b_i k_i \\ k_i = f_{\boldsymbol{\theta}}(t_n + c_i h, \hat{\mathbf{z}}_{t_n} + h(\sum_{j=1}^{q} a_{i,j} k_j)) \end{cases}$$

Trainable Solvers for (N)ODEs

- Let us assume a continuous time (Neural) ODE:

$$\dot{\mathbf{z}}_t = f_{\boldsymbol{\theta}}(t, \mathbf{z}_t)$$

- The solution of this equation is computed using a Runge Kutta scheme with q stages

$$\begin{cases} \hat{\mathbf{z}}_{t_{n+1}} = \Psi_{\boldsymbol{\phi}_q}(\hat{\mathbf{z}}_{t_n}) = \hat{\mathbf{z}}_{t_n} + h\sum_{i=1}^{q} b_i k_i \\ k_i = f_{\boldsymbol{\theta}}(t_n + c_i h, \hat{\mathbf{z}}_{t_n} + h(\sum_{j=1}^{q} a_{i,j} k_j)) \end{cases}$$

- where the parameters of the scheme are:

$$\boldsymbol{\phi}_q = \{\mathbf{A} = [a_{i,j}] \in \mathbb{R}^{q\times q}, \mathbf{b} = [b_i] \in \mathbb{R}^q, \mathbf{c} = [c_i] \in \mathbb{R}^q\}$$

Trainable Solvers for (N)ODEs

Trainable Solvers for (N)ODEs

- We optimize the parameters of the NODE θ, jointly to the parameters of the Runge-Kutta scheme ϕ_q :

$$\begin{aligned} \underset{\boldsymbol{\theta},\boldsymbol{\phi}_q}{\text{Minimize}} \quad & \mathcal{J}(\Psi_{\boldsymbol{\phi}_q}(\cdot)) \\ \text{Subject to} \quad & \mathcal{P}_{\boldsymbol{\phi}_q} \\ & \mathcal{S}_{\boldsymbol{\phi}_q} \end{aligned}$$

Trainable Solvers for (N)ODEs

- We optimize the parameters of the NODE θ, jointly to the parameters of the Runge-Kutta scheme ϕ_q :

$$\begin{array}{ll} \underset{\boldsymbol{\theta},\boldsymbol{\phi}_q}{\text{Minimize}} & \mathcal{J}(\Psi_{\boldsymbol{\phi}_q}(\cdot)) \\ \text{Subject to} & \mathcal{P}_{\boldsymbol{\phi}_q} \\ & \mathcal{S}_{\boldsymbol{\phi}_q} \end{array}$$

- where $\mathcal{J}$ is a scalar valued loss function that is subject to order/stability constraitns

Trainable Solvers for (N)ODEs

- We optimize the parameters of the NODE θ, jointly to the parameters of the Runge-Kutta scheme ϕ_q :

$$\begin{array}{lll} \underset{\boldsymbol{\theta},\phi_q}{\text{Minimize}} & \mathcal{J}(\Psi_{\phi_q}(\cdot)) & \\ \text{Subject to} & \qquad \mathcal{P}_{\phi_q} \longleftarrow & \text{Order constraints} \\ & \qquad \mathcal{S}_{\phi_q} \longleftarrow & \text{Stability constraints} \end{array}$$

- where $\mathcal{J}$ is a scalar valued loss function that is subject to order/stability constraitns

Trainable Solvers for (N)ODEs

- We optimize the parameters of the NODE θ, jointly to the parameters of the Runge-Kutta scheme ϕ_q :

$$
\begin{aligned}
&\underset{\boldsymbol{\theta},\boldsymbol{\phi}_q}{\text{Minimize}} \quad && \mathcal{J}(\Psi_{\boldsymbol{\phi}_q}(\cdot)) \\
&\text{Subject to} && \mathcal{P}_{\boldsymbol{\phi}_q} \longleftarrow \text{Order constraints} \\
& && \mathcal{S}_{\boldsymbol{\phi}_q} \longleftarrow \text{Stability constraints}
\end{aligned}
$$

- where $\mathcal{J}$ is a scalar valued loss function that is subject to order/stability constraitns

- The order constraint garentee that the solution of the trainable scheme converges to the analytical solution

Trainable Solvers for (N)ODEs

- We optimize the parameters of the NODE θ, jointly to the parameters of the Runge-Kutta scheme ϕ_q :

$$
\begin{aligned}
&\underset{\boldsymbol{\theta},\boldsymbol{\phi}_q}{\text{Minimize}} && \mathcal{J}(\Psi_{\boldsymbol{\phi}_q}(\cdot)) \\
&\text{Subject to} && \mathcal{P}_{\boldsymbol{\phi}_q} \longleftarrow \text{Order constraints} \\
& && \mathcal{S}_{\boldsymbol{\phi}_q} \longleftarrow \text{Stability constraints}
\end{aligned}
$$

- where $\mathcal{J}$ is a scalar valued loss function that is subject to order/stability constraitns
- The order constraint garentee that the solution of the trainable scheme converges to the analytical solution
- Stability constraints are optional, they can be enforced to garentee the asyptotic stability of the solution

Trainable Solvers for (N)ODEs, order constraints

Trainable Solvers for (N)ODEs, order constraints

- It can be shown assuming mild conditions on the NODE model $\dot{\mathbf{z}}_t = f_\theta(t, \mathbf{z}_t)$ that the order of the Runge-Kutta scheme (in terms of its local truncation error) corresponds to the order of convergence of the Runge-Kutta solution

Trainable Solvers for (N)ODEs, order constraints

- It can be shown assuming mild conditions on the NODE model $\dot{\mathbf{z}}_t = f_{\boldsymbol{\theta}}(t, \mathbf{z}_t)$ that the order of the Runge-Kutta scheme (in terms of its local truncation error) corresponds to the order of convergence of the Runge-Kutta solution

- For Runge-Kutta methods, it can be shown that first order constriants (or consistency) can be enforced in terms of the coeficients $\mathbf{b} = [b_i] \in \mathbb{R}^q$ as folows:

$$\mathcal{C}_{\phi q} : \sum_{i=1}^{q} b_i = 1$$

Trainable Solvers for (N)ODEs, order constraints

- It can be shown assuming mild conditions on the NODE model $\dot{\mathbf{z}}_t = f_{\boldsymbol{\theta}}(t, \mathbf{z}_t)$ that the order of the Runge-Kutta scheme (in terms of its local truncation error) corresponds to the order of convergence of the Runge-Kutta solution

- For Runge-Kutta methods, it can be shown that first order constriants (or consistency) can be enforced in terms of the coeficients $\mathbf{b} = [b_i] \in \mathbb{R}^q$ as folows:

$$\mathcal{C}_{\phi q} : \sum_{i=1}^{q} b_i = 1$$

- This constraint is satisfied exactly in our framework using projected gradient, which makes the trainable schemes at least first order accurate

Trainable Solvers for (N)ODEs, stability constraints 1

Trainable Solvers for (N)ODEs, stability constraints 1

- We consider absolute stability analysis

Trainable Solvers for (N)ODEs, stability constraints 1

- We consider absolute stability analysis
- The notion of absolute stability is based on the following linear test equation:

$$\dot{\mathbf{z}}_t = \mathbf{M}\mathbf{z}_t, \qquad \mathbf{z}_{t_0} = \mathbf{z}_0 \quad \text{where} \quad \mathbf{M} \in \mathbb{R}^{s \times s}$$

Trainable Solvers for (N)ODEs, stability constraints 1

- We consider absolute stability analysis
- The notion of absolute stability is based on the following linear test equation:

$$\dot{\mathbf{z}}_t = \mathbf{M}\mathbf{z}_t, \qquad \mathbf{z}_{t_0} = \mathbf{z}_0 \quad \text{where} \quad \mathbf{M} \in \mathbb{R}^{s \times s}$$

- For this equation, the trainable Runge-Kutta scheme can be written as:

$$\begin{cases} \hat{\mathbf{z}}_{t_0} = \mathbf{z}_{t_0} = \mathbf{z}_0 \\ \hat{\mathbf{z}}_{t+h} = \mathcal{R}_{\Psi_\phi}(h\mathbf{M})\hat{\mathbf{z}}_t \end{cases}$$

Matrix polynomial

Trainable Solvers for (N)ODEs, stability constraints 1

- We consider absolute stability analysis
- The notion of absolute stability is based on the following linear test equation:

$$\dot{\mathbf{z}}_t = \mathbf{M}\mathbf{z}_t, \qquad \mathbf{z}_{t_0} = \mathbf{z}_0 \quad \text{where} \quad \mathbf{M} \in \mathbb{R}^{s \times s}$$

- For this equation, the trainable Runge-Kutta scheme can be written as:

$$\begin{cases} \hat{\mathbf{z}}_{t_0} = \mathbf{z}_{t_0} = \mathbf{z}_0 \\ \hat{\mathbf{z}}_{t+h} = \mathcal{R}_{\Psi_\phi}(h\mathbf{M})\hat{\mathbf{z}}_t \end{cases}$$

Matrix polynomial

- Stability region of a trainable Runge-Kutta scheme can be defined as:

$$S_{\Psi_\phi} = \{x \in \mathbb{C} : \|\mathcal{R}_{\Psi_\phi}(x)\| \leq 1\}$$

Trainable Solvers for (N)ODEs, stability constraints 2

Trainable Solvers for (N)ODEs, stability constraints 2

- Stability constraints of a trainable Runge-Kutta scheme can be defined as:

$$\mathcal{S} : \|\mathcal{R}_{\Psi_\phi}(\lambda h)\| - 1 \leq 0$$

Trainable Solvers for (N)ODEs, stability constraints 2

- Stability constraints of a trainable Runge-Kutta scheme can be defined as:

$$\mathcal{S} : \|\mathcal{R}_{\Psi_\phi}(\lambda h)\| - 1 \leq 0$$

- Where the matrix polynomial of the Runge-Kutta scheme can be written as a function of the Runge-Kutta parameters as follows:

$$\mathcal{R}_{\Psi_\phi}(\lambda h) = \mathbf{b}^T(\mathbf{I} - (\lambda h)\mathbf{A})^{-1}\mathbf{1}$$

Trainable Solvers for (N)ODEs, stability constraints 2

- Stability constraints of a trainable Runge-Kutta scheme can be defined as:

$$\mathcal{S} : \|\mathcal{R}_{\Psi_\phi}(\lambda h)\| - 1 \leq 0$$

- Where the matrix polynomial of the Runge-Kutta scheme can be written as a function of the Runge-Kutta parameters as follows:

$$\mathcal{R}_{\Psi_\phi}(\lambda h) = \mathbf{b}^T(\mathbf{I} - (\lambda h)\mathbf{A})^{-1}\mathbf{1}$$

- Stability constraints are inequality constraints, they can be satisfied efficiently using the penalty method

Numerical experiments, stability constrained Runge-Kutta

Numerical experiments, stability constrained Runge-Kutta

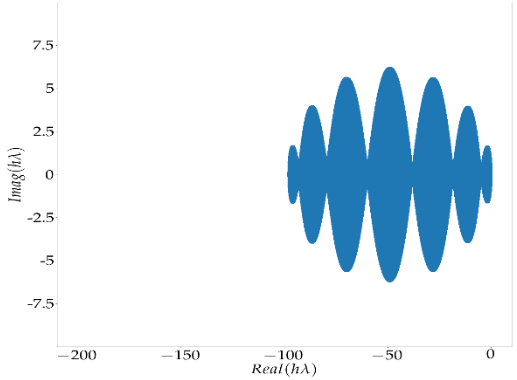
- Until which values of λh, a trainable Runge-Kutta scheme is able to integrate a linear test equation $\dot{\mathbf{z}}_t = \lambda \mathbf{z}_t$?

Numerical experiments, stability constrained Runge-Kutta

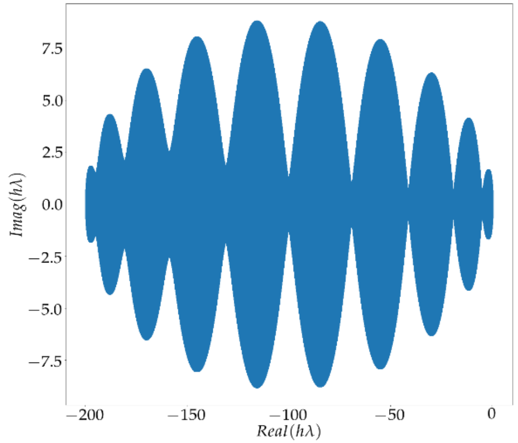
- Until which values of λh, a trainable Runge-Kutta scheme is able to integrate a linear test equation $\dot{\mathbf{z}}_t = \lambda \mathbf{z}_t$?

Real λ

\# stages = 7



\# stages = 10



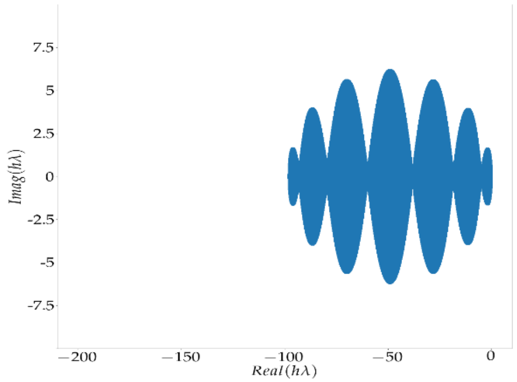
Numerical experiments, stability constrained Runge-Kutta

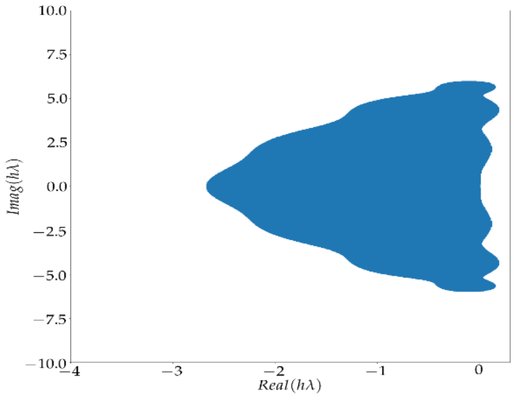
- Until which values of λh, a trainable Runge-Kutta scheme is able to integrate a linear test equation $\dot{\mathbf{z}}_t = \lambda \mathbf{z}_t$?

Real λ

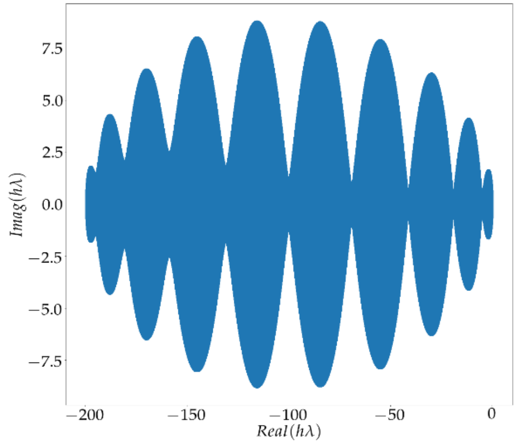
Imaginary λ

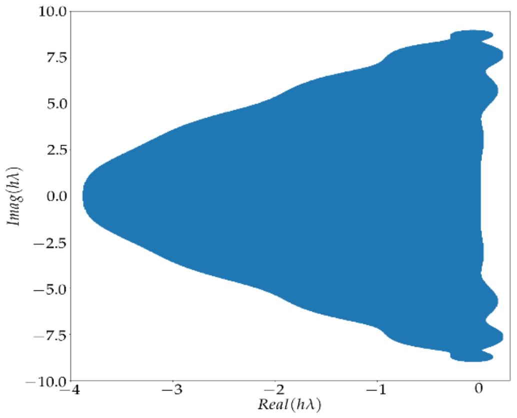
stages = 7





stages = 10





Numerical experiments, Numerical integration of known ODEs

Numerical experiments, Numerical integration of known ODEs

- Can we learn a numerical scheme that adapts to the dynamics of a known ODE ?

Numerical experiments, Numerical integration of known ODEs

- Can we learn a numerical scheme that adapts to the dynamics of a known ODE ?
- Learn a numerical scheme that is able to integrate the Lorenz 63 model for increasing time step values

$$\left\{\begin{array}{lll}\frac{dz_{t,1}}{dt} & = & \sigma\left(z_{t,2}-z_{t,2}\right)\\ \frac{dz_{t,2}}{dt} & = & \rho z_{t,1}-z_{t,2}-z_{t,1}z_{t,3}\\ \frac{dz_{t,3}}{dt} & = & z_{t,1}z_{t,2}-\beta z_{t,3}\end{array}\right.$$

Numerical experiments, Numerical integration of known ODEs

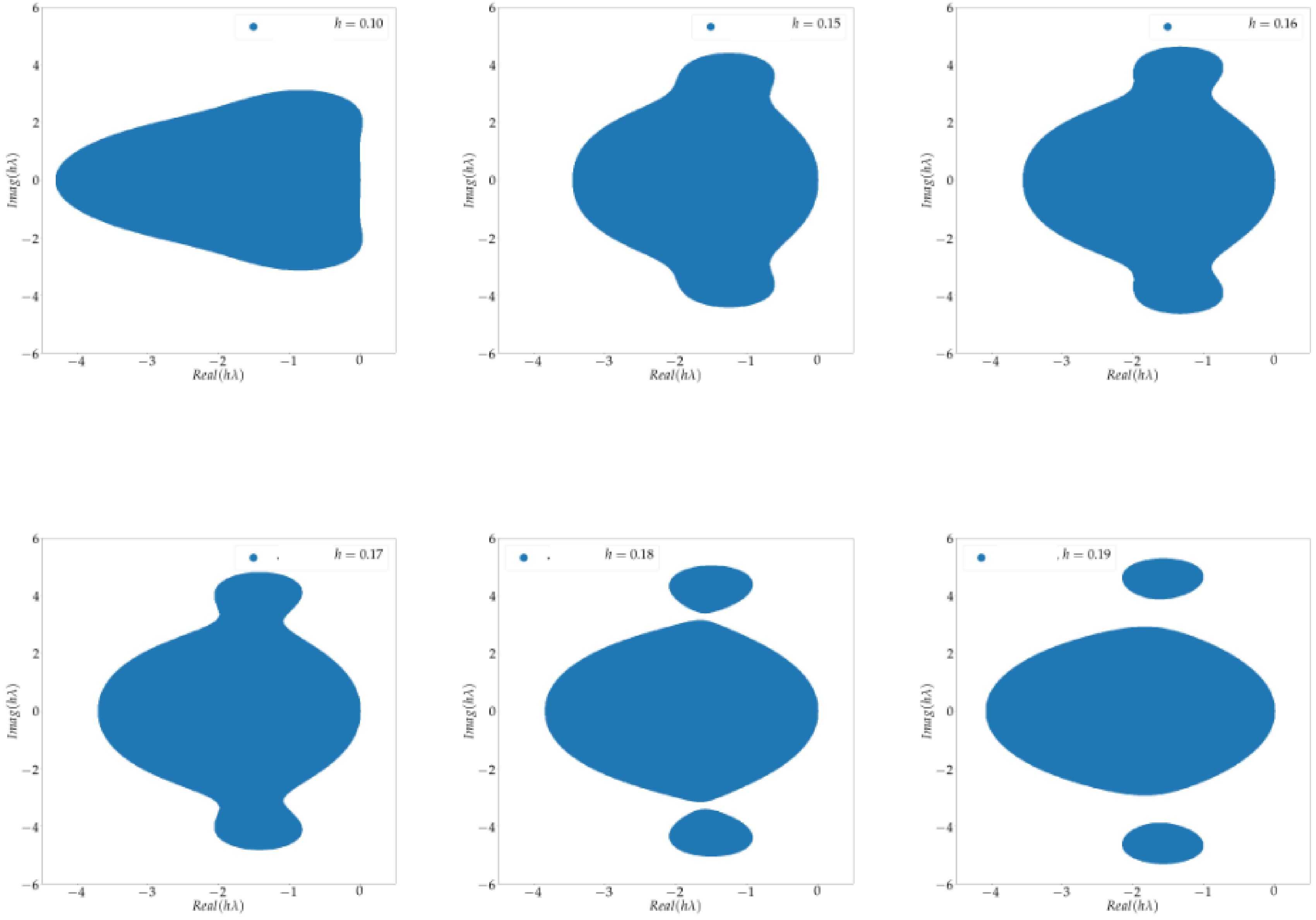
- Can we learn a numerical scheme that adapts to the dynamics of a known ODE ?

- Learn a numerical scheme that is able to integrate the Lorenz 63 model for increasing time step values

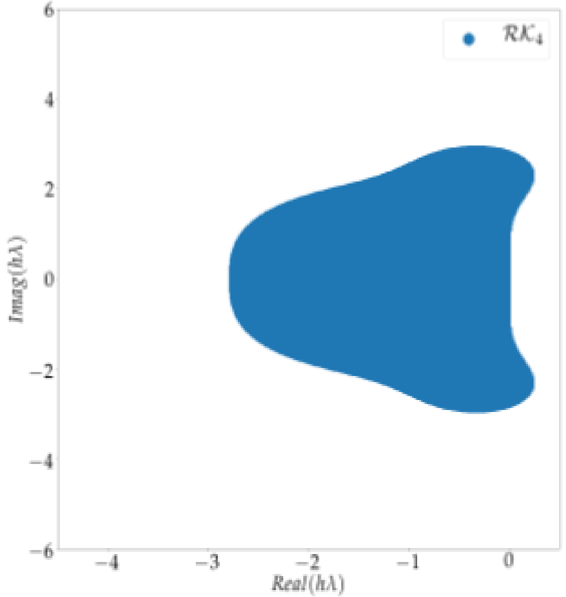
$$\left\{ \frac{dz_{t,1}}{dt} = \sigma \left(z_{t,2} - z_{t,2} \right) \right.$$

h	0.1	0.15	0.16	0.17	0.18	0.19
TRK4	✓	✓	✓	✓	X	X
RK4	✓	X	X	X	X	X

Integration ability of both the classical RK4 and a trainable Runge Kutta with 4 stages (TRK4) on the Lorenz 63 system with different integration time steps



Stability regions of the TRK4 schemes trained on integration time steps ranging from h = 0.1 to h = 0.19



Stability region of the classical RK4

Numerical experiments, Identification of dynamical systems

Numerical experiments, Identification of dynamical systems

- Can we learn a numerical scheme that adapts to the dynamics of a unknown ODE ?

Numerical experiments, Identification of dynamical systems

- Can we learn a numerical scheme that adapts to the dynamics of a unknown ODE ?

- Learn a numerical scheme that is able to identify the dynamics of the Lorenz 63 model given training data with decreasing sampling rate

$$\left\{\begin{array}{lcl} \frac{dz_{t,1}}{dt} & = & \sigma\left(z_{t,2}-z_{t,2}\right) \\ \frac{dz_{t,2}}{dt} & = & \rho z_{t,1}-z_{t,2}-z_{t,1}z_{t,3} \\ \frac{dz_{t,3}}{dt} & = & z_{t,1}z_{t,2}-\beta z_{t,3} \end{array}\right.$$

Numerical experiments, Identification of dynamical systems

- Can we learn a numerical scheme that adapts to the dynamics of a unknown ODE ?

- Learn a numerical scheme that is able to identify the dynamics of the Lorenz 63 model given training data with decreasing sampling rate

Coarser sampling →

Model		$h_1 = 0.2$	$h_2 = 0.3$	$h_3 = 0.4$
SR	$t_0 + h$	9.06	> 10	9.34
	$t_0 + 4h$	6.02	5.81	6.97
Euler	$t_0 + h$	4.27	2.57	1.99
	$t_0 + 4h$	> 10	> 10	7.89
$\mathcal{RK}_4$	$t_0 + h$	2.05	3.10	2.48
	$t_0 + 4h$	3.82	7.33	> 10
Dopri8	$t_0 + h$	**0.005**	**0.0001**	3.1305
	$t_0 + 4h$	0.021	**0.0003**	> 10
$\mathcal{TRK}_{10}$	$t_0 + h$	0.017	0.077	**0.189**
	$t_0 + 4h$	**0.020**	0.23	**1.93**

Forecasting performance of data-driven models for Lorenz-63 dynamical model. Mean Root Mean Squared Error (RMSE) for different forecasting time-steps of the tested models.

Numerical experiments, Identification of dynamical systems

- Can we learn a numerical scheme that adapts to the dynamics of a unknown ODE ?

- Learn a numerical scheme that is able to identify the dynamics of the Lorenz 63 model given training data with decreasing sampling rate

Coarser sampling →

Model		$h_1 = 0.2$	$h_2 = 0.3$	$h_3 = 0.4$
SR	$t_0 + h$	9.06	> 10	9.34
	$t_0 + 4h$	6.02	5.81	6.97
Euler	$t_0 + h$	4.27	2.57	1.99
	$t_0 + 4h$	> 10	> 10	7.89
$\mathcal{RK}_4$	$t_0 + h$	2.05	3.10	2.48
	$t_0 + 4h$	3.82	7.33	> 10
Dopri8	$t_0 + h$	**0.005**	**0.0001**	3.1305
	$t_0 + 4h$	0.021	**0.0003**	> 10
$\mathcal{TRK}_{10}$	$t_0 + h$	0.017	0.077	**0.189**
	$t_0 + 4h$	**0.020**	0.23	**1.93**

Forecasting performance of data-driven models for Lorenz-63 dynamical model. Mean Root Mean Squared Error (RMSE) for different forecasting time-steps of the tested models.

- Continuous time formulation
- Data are sparse, impossible to estimate the derivatives

Numerical experiments, Identification of dynamical systems

- Can we learn a numerical scheme that adapts to the dynamics of a unknown ODE ?

- Learn a numerical scheme that is able to identify the dynamics of the Lorenz 63 model given training data with decreasing sampling rate

Coarser sampling →

Model		$h_1 = 0.2$	$h_2 = 0.3$	$h_3 = 0.4$
SR	$t_0 + h$	9.06	> 10	9.34
	$t_0 + 4h$	6.02	5.81	6.97
Euler	$t_0 + h$	4.27	2.57	1.99
	$t_0 + 4h$	> 10	> 10	7.89
$\mathcal{RK}_4$	$t_0 + h$	2.05	3.10	2.48
	$t_0 + 4h$	3.82	7.33	> 10
Dopri8	$t_0 + h$	**0.005**	**0.0001**	3.1305
	$t_0 + 4h$	0.021	**0.0003**	> 10
$\mathcal{TRK}_{10}$	$t_0 + h$	0.017	0.077	**0.189**
	$t_0 + 4h$	**0.020**	0.23	**1.93**

Forecasting performance of data-driven models for Lorenz-63 dynamical model. Mean Root Mean Squared Error (RMSE) for different forecasting time-steps of the tested models.

Numerical experiments, Identification of dynamical systems

- Can we learn a numerical scheme that adapts to the dynamics of a unknown ODE ?

- Learn a numerical scheme that is able to identify the dynamics of the Lorenz 63 model given training data with decreasing sampling rate

Coarser sampling

Model		$h_1 = 0.2$	$h_2 = 0.3$	$h_3 = 0.4$
SR	$t_0 + h$	9.06	> 10	9.34
	$t_0 + 4h$	6.02	5.81	6.97
Euler	$t_0 + h$	4.27	2.57	1.99
	$t_0 + 4h$	> 10	> 10	7.89
$\mathcal{RK}_4$	$t_0 + h$	2.05	3.10	2.48
	$t_0 + 4h$	3.82	7.33	> 10
Dopri8	$t_0 + h$	**0.005**	**0.0001**	3.1305
	$t_0 + 4h$	0.021	**0.0003**	> 10
$\mathcal{TRK}_{10}$	$t_0 + h$	0.017	0.077	**0.189**
	$t_0 + 4h$	**0.020**	0.23	**1.93**

Forecasting performance of data-driven models for Lorenz-63 dynamical model. Mean Root Mean Squared Error (RMSE) for different forecasting time-steps of the tested models.

- Fixed step size algorithms fail
- Stability region is too small to integrate the data at the sampling rate of the observations

Numerical experiments, Identification of dynamical systems

- Can we learn a numerical scheme that adapts to the dynamics of a unknown ODE ?

- Learn a numerical scheme that is able to identify the dynamics of the Lorenz 63 model given training data with decreasing sampling rate

Coarser sampling →

Model		$h_1 = 0.2$	$h_2 = 0.3$	$h_3 = 0.4$
SR	$t_0 + h$	9.06	> 10	9.34
	$t_0 + 4h$	6.02	5.81	6.97
Euler	$t_0 + h$	4.27	2.57	1.99
	$t_0 + 4h$	> 10	> 10	7.89
$\mathcal{RK}_4$	$t_0 + h$	2.05	3.10	2.48
	$t_0 + 4h$	3.82	7.33	> 10
Dopri8	$t_0 + h$	**0.005**	**0.0001**	3.1305
	$t_0 + 4h$	0.021	**0.0003**	> 10
$\mathcal{TRK}_{10}$	$t_0 + h$	0.017	0.077	**0.189**
	$t_0 + 4h$	**0.020**	0.23	**1.93**

Forecasting performance of data-driven models for Lorenz-63 dynamical model. Mean Root Mean Squared Error (RMSE) for different forecasting time-steps of the tested models.

Numerical experiments, Identification of dynamical systems

- Can we learn a numerical scheme that adapts to the dynamics of a unknown ODE ?

- Learn a numerical scheme that is able to identify the dynamics of the Lorenz 63 model given training data with decreasing sampling rate

Coarser sampling →

Model		$h_1 = 0.2$	$h_2 = 0.3$	$h_3 = 0.4$
SR	$t_0 + h$	9.06	> 10	9.34
	$t_0 + 4h$	6.02	5.81	6.97
Euler	$t_0 + h$	4.27	2.57	1.99
	$t_0 + 4h$	> 10	> 10	7.89
$\mathcal{RK}_4$	$t_0 + h$	2.05	3.10	2.48
	$t_0 + 4h$	3.82	7.33	> 10
Dopri8	$t_0 + h$	**0.005**	**0.0001**	3.1305
	$t_0 + 4h$	0.021	**0.0003**	> 10
$\mathcal{TRK}_{10}$	$t_0 + h$	0.017	0.077	**0.189**
	$t_0 + 4h$	**0.020**	0.23	**1.93**

Forecasting performance of data-driven models for Lorenz-63 dynamical model. Mean Root Mean Squared Error (RMSE) for different forecasting time-steps of the tested models.

- Adaptive solver and trainable schemes are able to get the most accurate results

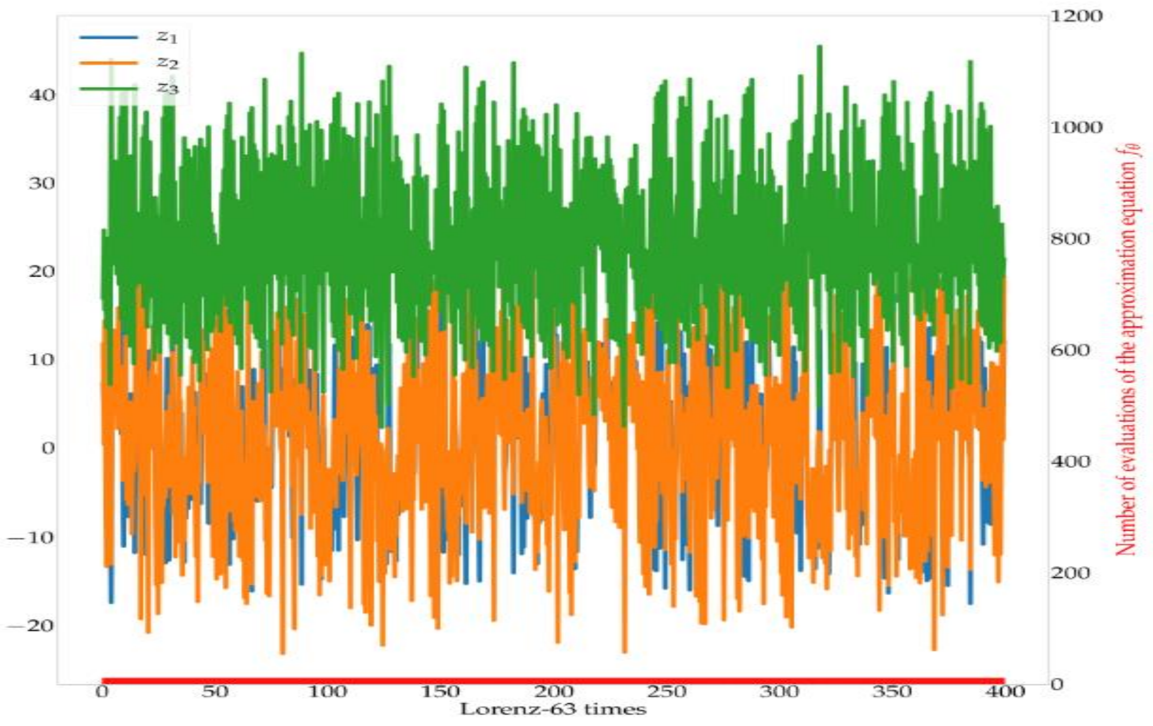
Numerical experiments, Identification of dynamical systems

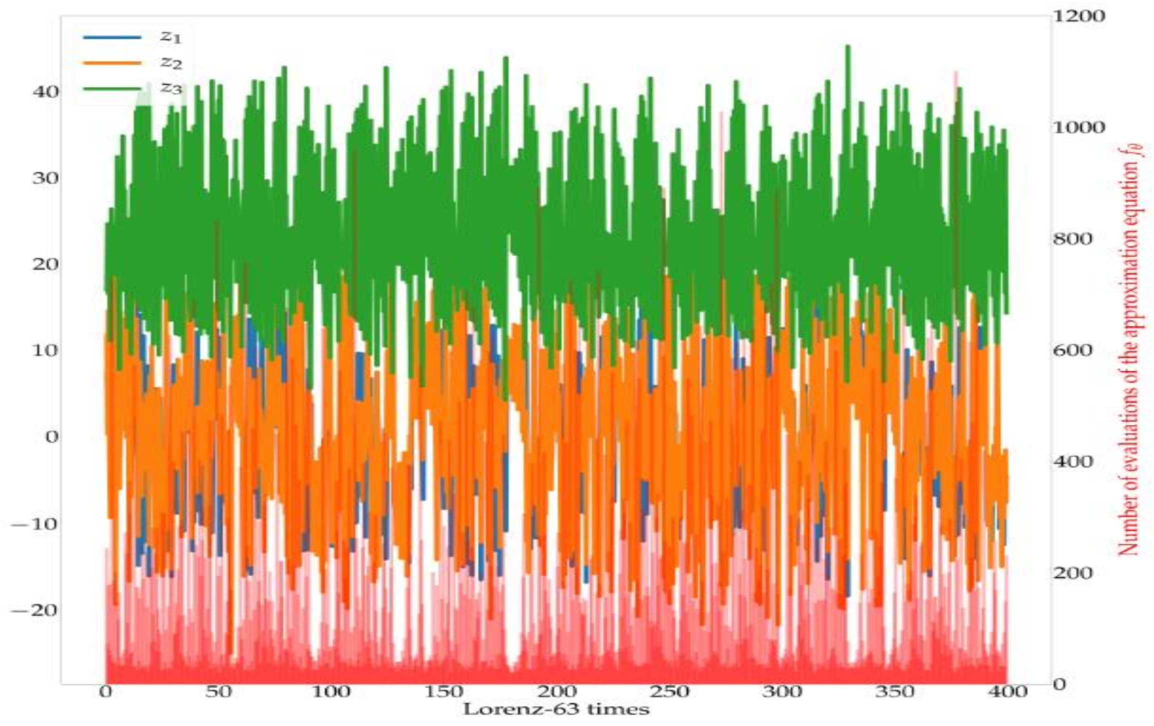
Numerical experiments, Identification of dynamical systems

- Comparing the Number of Function Evaluation (NFE) for both the TRK scheme and the adaptive solver

Numerical experiments, Identification of dynamical systems

- Comparing the Number of Function Evaluation (NFE) for both the TRK scheme and the adaptive solver





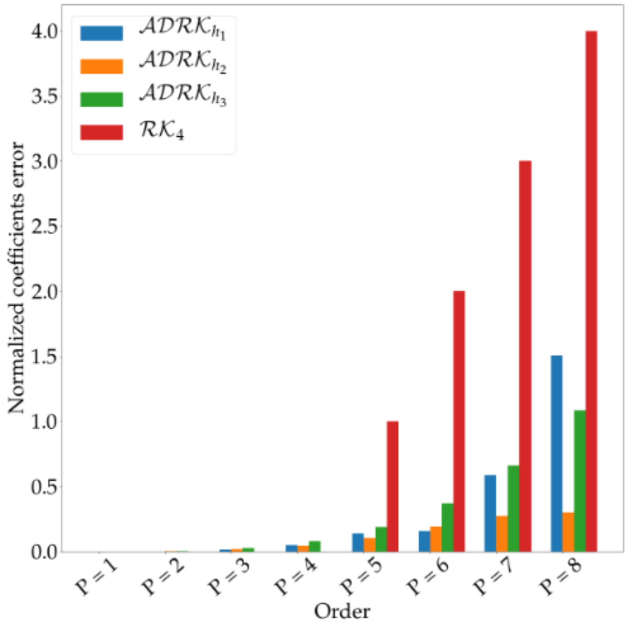
Numerical experiments, Identification of dynamical systems

Numerical experiments, Identification of dynamical systems

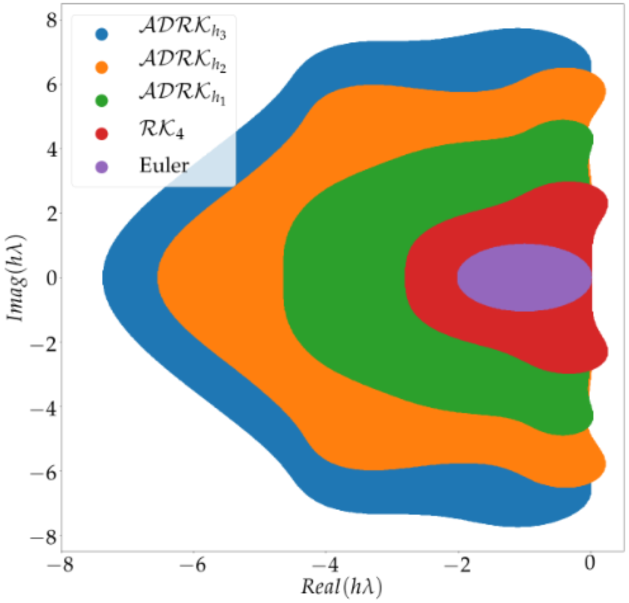
- Evaluating the order and stability properties of the trained Runge-Kutta schemes

Numerical experiments, Identification of dynamical systems

- Evaluating the order and stability properties of the trained Runge-Kutta schemes



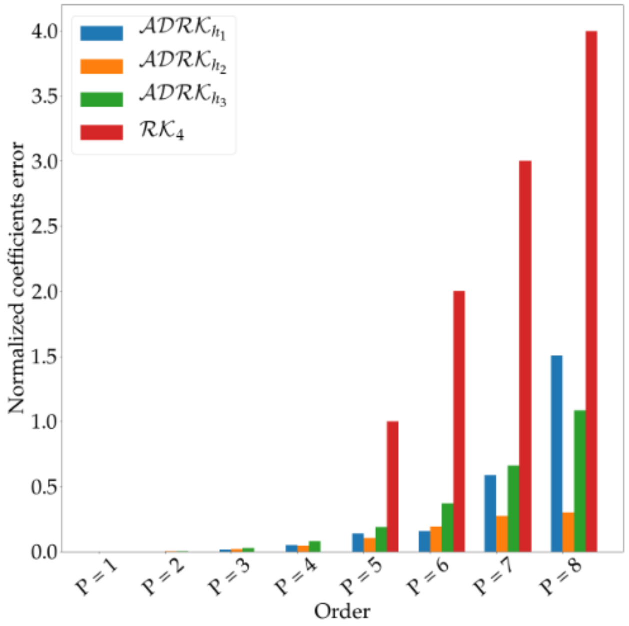
Order coefficients error



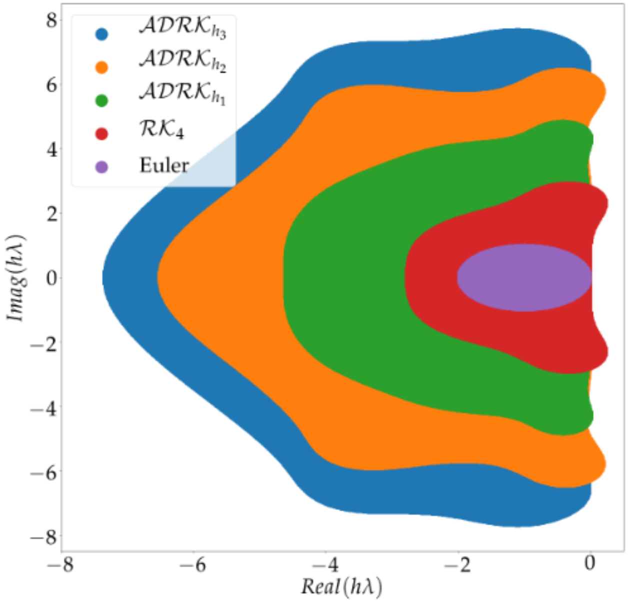
Stability region of the learnt schemes

Numerical experiments, Identification of dynamical systems

- Evaluating the order and stability properties of the trained Runge-Kutta schemes



Order coefficients error



Stability region of the learnt schemes

- The trainable schemes adapt to the dynamics of the learned ODE

Conclusion and perspectives

- Training Numerical schemes jointly with NODE models allows to reduce the computational complexity of NODEs at learning and inference time
- The trained numerical schemes are constrained to guarantee convergence of the solution of the ODE to the analytical one (through the order constraint)
- The order and the stability region of the scheme adapts to the complexity of the ODE, leading to simulations that can operate at a fixed NFE
- Future applications on large scale diffusion models, and high dimensional Partial Differential Equations