Solving the forward and inverse problem in complex systems via machine learning: challenges and perspectives

Constantinos Siettos

Department of Mathematics and Applications "Renato Caccioppoli"

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SSM

www.siettos.net Research Group
NUMADICS
Numerical Analysis, Machine Learning and
Data Mining for Complex and Multiscale Systems

Beware of ~~Greeks~~ Machine Learning Bearing Gifts

Homer, Iliad

Sorry Epeius, the men won't get inside unless our COLP submits a full risk assessment

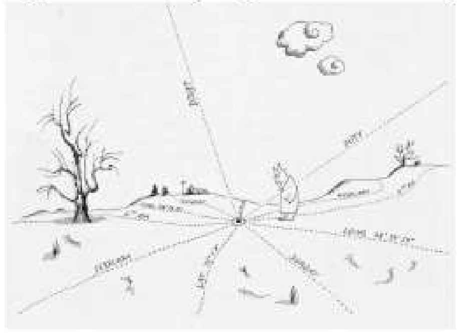
OR A BETTER TITLE WOULD BE:

Gear, C. W. (1981), "Numerical solution of ordinary differential equations: Is there anything left to do?", SIAM Review, 23 (1): 10–24, doi:10.1137/1023002

Open challenging Tasks

- (A) Can ML beat traditional numerical analysis methods for the solution of stiff ODEs and PDEs?,

FORWARD PROBLEM

- (B) Deal with the so-called "curse of dimensionality" when trying to efficiently learn ML models with good generalization properties, and 
- (C) Discover from data the appropriate macroscopic quantities/physics for the emergent dynamics,

This is Mr Smith from Big Data Mining. He says he's found an insight.

INVERSE PROBLEM

- (D) Bridge Machine Learning with Physics-based Modelling, Discover Physical Laws from Data

Numerical Solution of Differential Problems, IVPs

Picard's Theorem: existence of a solution to a given initial value

Consider the IVP with m first-order ordinary differential equations:

$$\frac{d\boldsymbol{y}}{dx} = \boldsymbol{f}(\boldsymbol{y}, x), \boldsymbol{y}(x_0) = \boldsymbol{y}_0, \qquad (2)$$

$\boldsymbol{f} : \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}^m$ being continuous multivariate functions in some closed domain D of the $(x, \boldsymbol{y})$ space containing the point $(x_0, \boldsymbol{y}_0)$.

Then if $\boldsymbol{f}$ satisfies the Lipschitz inequality:

$$||\boldsymbol{f}(x, \boldsymbol{y}) - \boldsymbol{f}(x, \boldsymbol{z})||_\infty \leq K||\boldsymbol{y} - \boldsymbol{z}||_\infty, K > 0 \qquad (3)$$

in D, then there exists a unique continuously differentiable function $\boldsymbol{y}(x)$ which satisfies (2).

Numerical Solution of ODEs

Picard's Theorem

(P. J. Collins, Differential and Integral Equations, Part I, Mathematical Institute Oxford, 1988)
For m first-order differential equations:

$$\frac{d\boldsymbol{y}}{dx} = \boldsymbol{f}(\boldsymbol{y}, x), \boldsymbol{y}(x_0) = \boldsymbol{y}_0, \tag{4}$$

The sequence of functions $\{\boldsymbol{y}^{(n)}\}_0^\infty$ defined recursively as

$$\boldsymbol{y}^{(0)} = \boldsymbol{y}_0$$

$\boldsymbol{y}^{(n)}(x) = \boldsymbol{y}_0 + \int_{x_0}^{x} \boldsymbol{f}(s, \boldsymbol{y}^{(n-1)}(s))ds, n = 1, 2, \ldots (5)$
converges uniformly in the interval D to the solution:

$$\boldsymbol{y}(x) = \boldsymbol{y}_0 + \int_{x_0}^{x} \boldsymbol{f}(s, \boldsymbol{y}(s))ds \tag{6}$$

The uniqueness of the solution follows from the Lipschitz condition.

Numerical Solution of ODEs

Classical Numerical Methods

- One Step Methods
  - Explicit Euler (rectangle rule for integration)
    $\boldsymbol{y}(x_n) = \boldsymbol{y}(x_{n-1}) + dx\boldsymbol{f}(x_{n-1}, \boldsymbol{y}(x_{n-1})),\ n = 1, 2, \ldots,$
    $dx = \frac{X_M - x_{\mathbf{0}}}{N}$
  - Implicit Euler:
    $\boldsymbol{y}(x_n) = \boldsymbol{y}(x_{n-1}) + dx\boldsymbol{f}(x_n, \boldsymbol{y}(x_n)),\ n = 1, 2, \ldots,$
    $dx = \frac{X_M - x_{\mathbf{0}}}{N}.$
  - Runge-Kutta methods
    $\boldsymbol{y}(x_n) = \boldsymbol{y}(x_{n-1}) + \frac{1}{6}dx(\boldsymbol{k}_1 + \boldsymbol{k}_2 + \boldsymbol{k}_3 + \boldsymbol{k}_4),$
    $\boldsymbol{k}_1 = \boldsymbol{f}(x_{n-1}, \boldsymbol{y}(x_{n-1})),$

$\boldsymbol{k}_2 = \boldsymbol{f}(x_{n-1} + \frac{1}{2}dx, \boldsymbol{y}(x_{n-1}) + \frac{1}{2}\boldsymbol{k}_1),\ \boldsymbol{k}_3 = \boldsymbol{f}(x_{n-1} + \frac{1}{2}dx, \boldsymbol{y}(x_{n-1}) + \frac{1}{2}\boldsymbol{k}_2),$

$\boldsymbol{k}_4 = \boldsymbol{f}(x_{n-1} + dx, \boldsymbol{y}(x_{n-1}) + \boldsymbol{k}_3)$

Numerical Solution of ODEs: The Forward Problem
Numerical Methods

- Multistep Methods
  - Adams- Bashforth
  - Adams–Moulton (implicit)

The idea is to approximate the integral below:

$$\boldsymbol{y}(x_n) = \boldsymbol{y}(x_{n-1}) + \int_{x_{n-1}}^{x_n} \boldsymbol{f}(x, \boldsymbol{y}) dx \tag{7}$$

by first approximating $\boldsymbol{f}(x)$ using polynomial interpolation and then integrate the interpolating polynomial. So for example the Adams-Bashforth integration reads:

$$\boldsymbol{y}(x_n) = \boldsymbol{y}(x_{n-1}) + \frac{1}{2} dx[3\boldsymbol{f}(x_{n-1}, \boldsymbol{y}(x_{n-1}) - \boldsymbol{f}(x_{n-2}, \boldsymbol{y}(x_{n-2})] \tag{8}$$

Numerical Solution of ODEs
Stiff Problems

- The unknown solution may exhibit a complex behaviour, including steep gradients, and stiffness which pose difficulties in the numerical solution.
  Shampine, and Gear, A user's view of solving stiff ordinary differential equations, SIAM review, 21, 1–17, 1979.
  Stiff problems, are the ones which integration "with a code that aims at nonstiff problems proves conspicuously inefficient for no obvious reason (such as a severe lack of smoothness in the equation or the presence of singularities)"

Numerical Solution of ODEs

Stiff Problems and Steep Gradients: NOT TO BE CONFUSED

Shampine, and Gear, A user's view of solving stiff ordinary differential equations, SIAM review, 21, 1–17, 1979.
Stiffness should NOT be confused with the presence of steep gradients.
For example, at the regions where the relaxation oscillations of the van der Pol equation exhibit very sharp changes resembling a discontinuity, the equations are not stiff. Finally, note that the spatial discretization of PDEs may lead to stiff systems of ODEs depending on the spatial mesh density.

Numerical Solution of ODEs

Stiff Problems

Predictor-corrector multi-step method (Gear's method-use of Newton's method)

Gear, C. W. (1981), "Numerical solution of ordinary differential equations: Is there anything left to do?", SIAM Review, 23 (1): 10–24, doi:10.1137/1023002

Numerical Solution of PDEs

Finite Differences and Finite Elements

$$Lu = f(u, \lambda) \text{ in } \Omega, \tag{11}$$

with boundary conditions:

$$B_l u = g_l, \ \text{ in } \partial\Omega_l\,, \ \ l = 1, 2, \ldots, m\,, \tag{12}$$

A numerical solution $\tilde{u} = \tilde{u}(\lambda)$ to the above problem at particular values of the parameters λ is typically found iteratively by applying e.g. Newton-Raphson or matrix-free Krylov-subspace methods (Newton-GMRES) on a finite system of M nonlinear algebraic equations resulting from FD, FEM, Spectral methods

Numerical Solution of PDEs

Finite Elements

- We seek for a solution of in N points x_j of the domain Ω according to:

$$u = \sum_{j=1}^{N} w_j \phi_j, \tag{13}$$

where the basis functions ϕ_j are defined so that they satisfy the completeness requirement.

- The scheme can be written as the minimization of the weighted residuals $R_k, k = 1, 2, \ldots N$ defined as:

$$R_k = \int_{\Omega} (Lu - f(u, \lambda))\phi_k \, d\Omega + \sum_{l=1}^{m} \int_{\partial\Omega_k} (B_k u - g_l)\phi_l \, d\sigma \tag{14}$$

Numerical Solution of the Forward and Inverse Problem for Differential Equations
with Machine learning

- Karniadakis G, Kevrekidis I, Lu, Perdikaris P, Wang S, Yang L, (2021) Physics-informed machine learning, Nature Reviews

  -

  From the early 90s:

- Rico-Martinez, K Krischer, IG Kevrekidis et al., (1992) Discrete-vs. continuous-time nonlinear signal processing of Cu electrodissolution data-Runge-Kutta Integrator as ANN
- Lagaris et al. Fotiadis (1998): feedforward neural networks (FNN) for simple differential equations.
- Gonzalez-Garcia et al. Kevrekidis (1998): multilayer neural network scheme that resembles the Runge-Kutta integrator for PDEs (Kuramoto Sivashinsky)

Numerical Solution of the Forward and Inverse Problem for Differential Equations

Machine learning/ Physics Informed Neural Networks

- Deep Learning

  - Han et al. Weinan E. (2018) PNAS Deep Neural Networks-high-dimensional nonlinear parabolic PDEs Black–Scholes, the Hamilton–Jacobi–Bellman and the Allen–Cahn equation.

- Gaussian Processes

  -Raissi, Perdikaris, Karniadakis SIAM J SC (2018)
  -Chen, Hosseini,Owhadi, Stuart, Andrew (2021) J Comp Phys

- Physics-informed neural networks/ automatic differentiation

  - Raissi, Perdikaris, Karniadakis J of Comp Phys (2019) Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations
  - Meng, Li, Zhang,Karniadakis Meth in Appl. Mech. Eng.(2020) PPINN: Parareal physics-informed neural network for time-dependent PDEs, Comp
  - Lu, Meng,Mao, Karniadakis (2021) DeepXDE: A Deep Learning Library for Solving Differential Equations, SIAM Rev.

Numerical Solution of the Forward and Inverse Problem for Differential Equations

with Machine Learning: The “classical Machine Learning way"

Physics Informed Neural Networks

Let's assume a set of m points $\boldsymbol{x}_i \in \Omega \subset \mathbb{R}^d$ of the independent (spatial) variables, defining the grid in the domain Ω, n_Ω points along the boundary of the domain, $\partial\Omega$ and n_t points in the time interval. Then the “classical way" to solve (time-dependent) differential equations in the general form

$$\frac{\partial u}{\partial t} = L(\boldsymbol{x}, u, \nabla u, \nabla^2 u, \ldots), \tag{15}$$

where u satisfies the boundary conditions $Bu = g$, in $\partial\Omega$ (B is the boundary differential operator) involves the solution of a minimization problem of the form:

Physics Informed Neural Networks

Numerical Solution of Differential Equations

with Neural Networks: The “classical Machine Learning way"

$$\min_{\boldsymbol{P},\boldsymbol{Q}} E(\boldsymbol{P},\boldsymbol{Q}) := \sum_{i=1}^{m}\sum_{j=1}^{n_t} \left\| \frac{d\Psi}{dt}(.) - L(\boldsymbol{x}_i, \Psi(.), \nabla\Psi(.), \nabla^2\Psi(.), \dots) \right\|^2 + \tag{16}$$

$$\sum_{j=1}^{n_\Omega} \| B\Psi(.) - g \|^2 ,$$

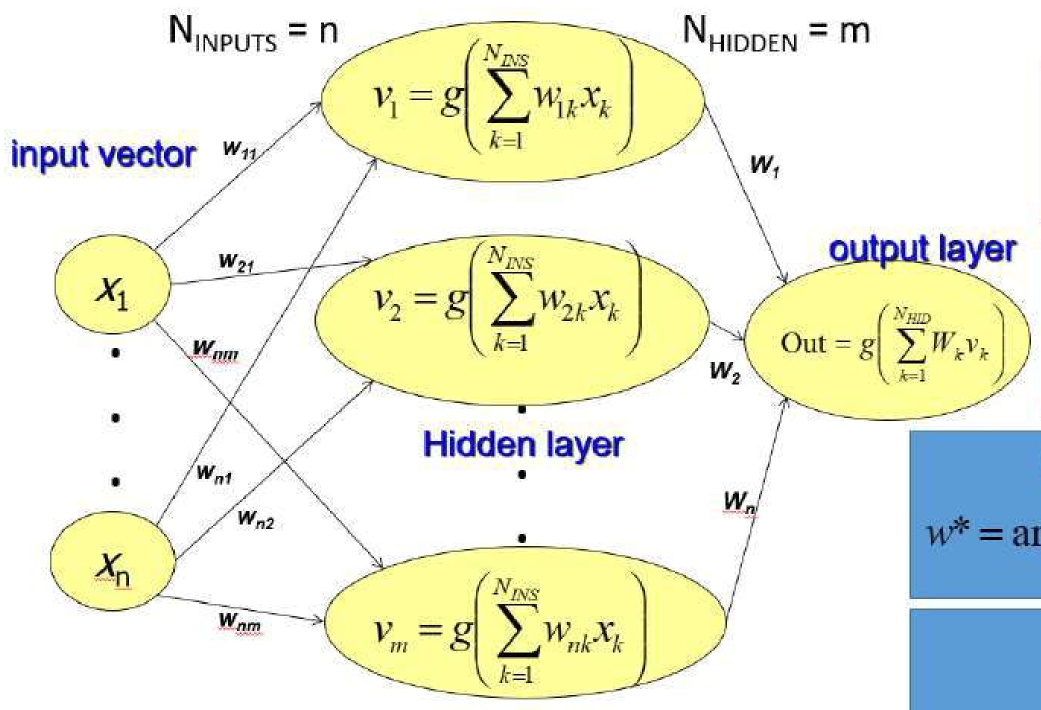
where $\Psi(.) := \Psi(t_j, \boldsymbol{x}_i, N(t_j, \boldsymbol{x}_i, \boldsymbol{P}, \boldsymbol{Q}))$ represents a “trial" function approximating the solution u at $\boldsymbol{x}_i$ and $N(t_j, \boldsymbol{x}_i, \boldsymbol{P}, \boldsymbol{Q})$ is a machine learning algorithm; $\boldsymbol{P}$ contains the parameters of the machine learning scheme (e.g. for a FNN the internal weights $\boldsymbol{W}$, the biases $\boldsymbol{B}$, the weights between the last hidden and the output layer $\boldsymbol{W}^o$), $\boldsymbol{Q}$ contains the hyperparameters.

Physics Informed Neural Networks

Machine Learning

Training is computationally demanding even for the simplest structures!

Iteratively: e.g. with quasi-Newton BFGS, Back-Propagation, Adams etc.



$N_{\text{INPUTS}} = n$
$N_{\text{HIDDEN}} = m$
input vector
x_1
x_n
w_{11}
w_{21}
w_{nm}
w_{n1}
w_{n2}
w_{nm}
$v_1 = g\left(\sum_{k=1}^{N_{INS}} w_{1k} x_k\right)$
$v_2 = g\left(\sum_{k=1}^{N_{INS}} w_{2k} x_k\right)$
$v_m = g\left(\sum_{k=1}^{N_{INS}} w_{nk} x_k\right)$
Hidden layer
W_1
W_2
W_n
output layer
$\text{Out} = g\left(\sum_{k=1}^{N_{HID}} W_k v_k\right)$

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$$

2. Choose loss function

$$l = (\hat{y}, y_i) \in \Re$$

3. Define goal

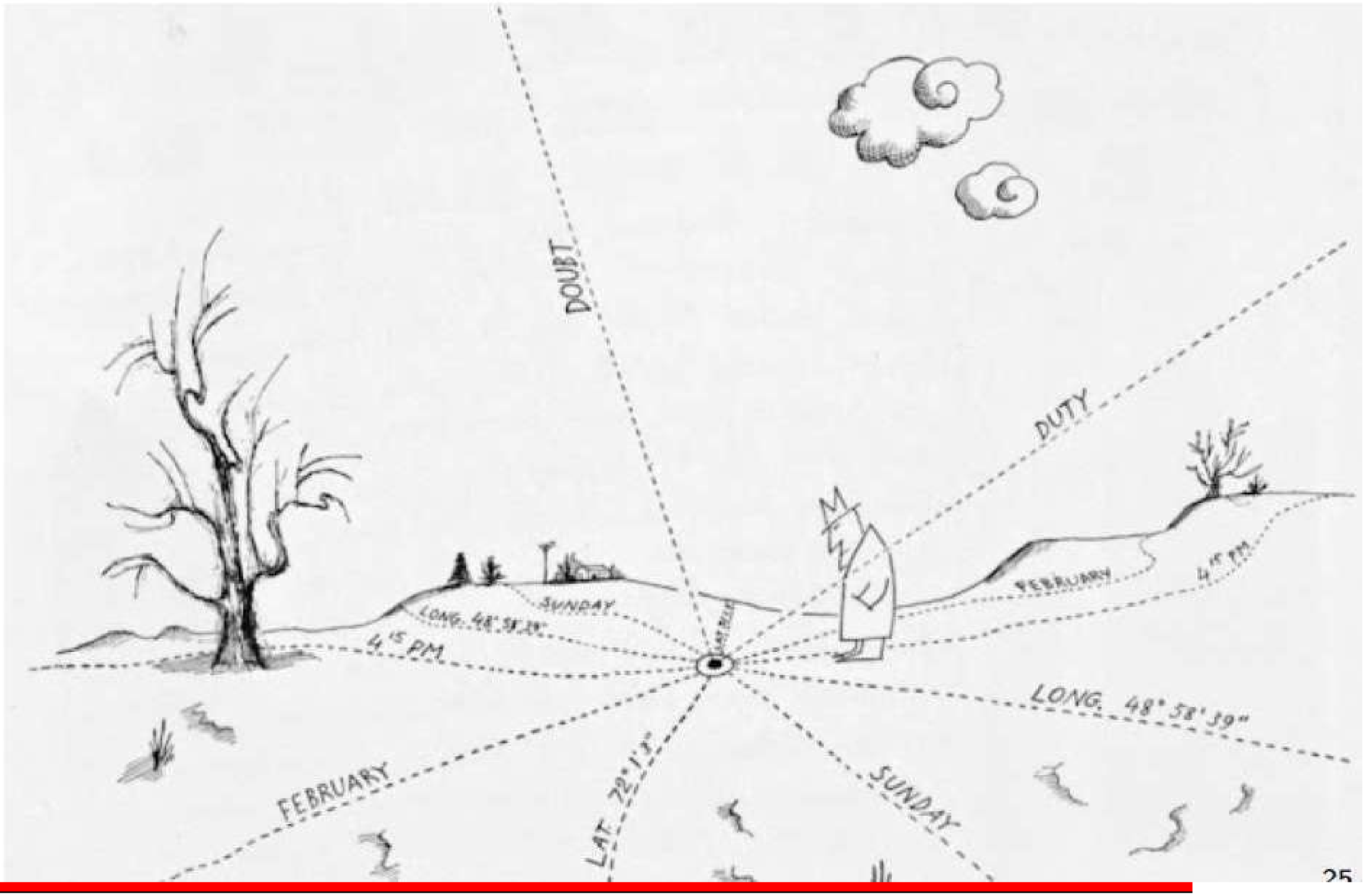
$$w^* = \arg\min_{w} \sum_{i=1}^{N} l(\hat{y}(x_i, \theta), y_i)$$

4. Train

$$w^{k+1} = w^k - \eta_k \nabla l$$

Curse of Dimensionality

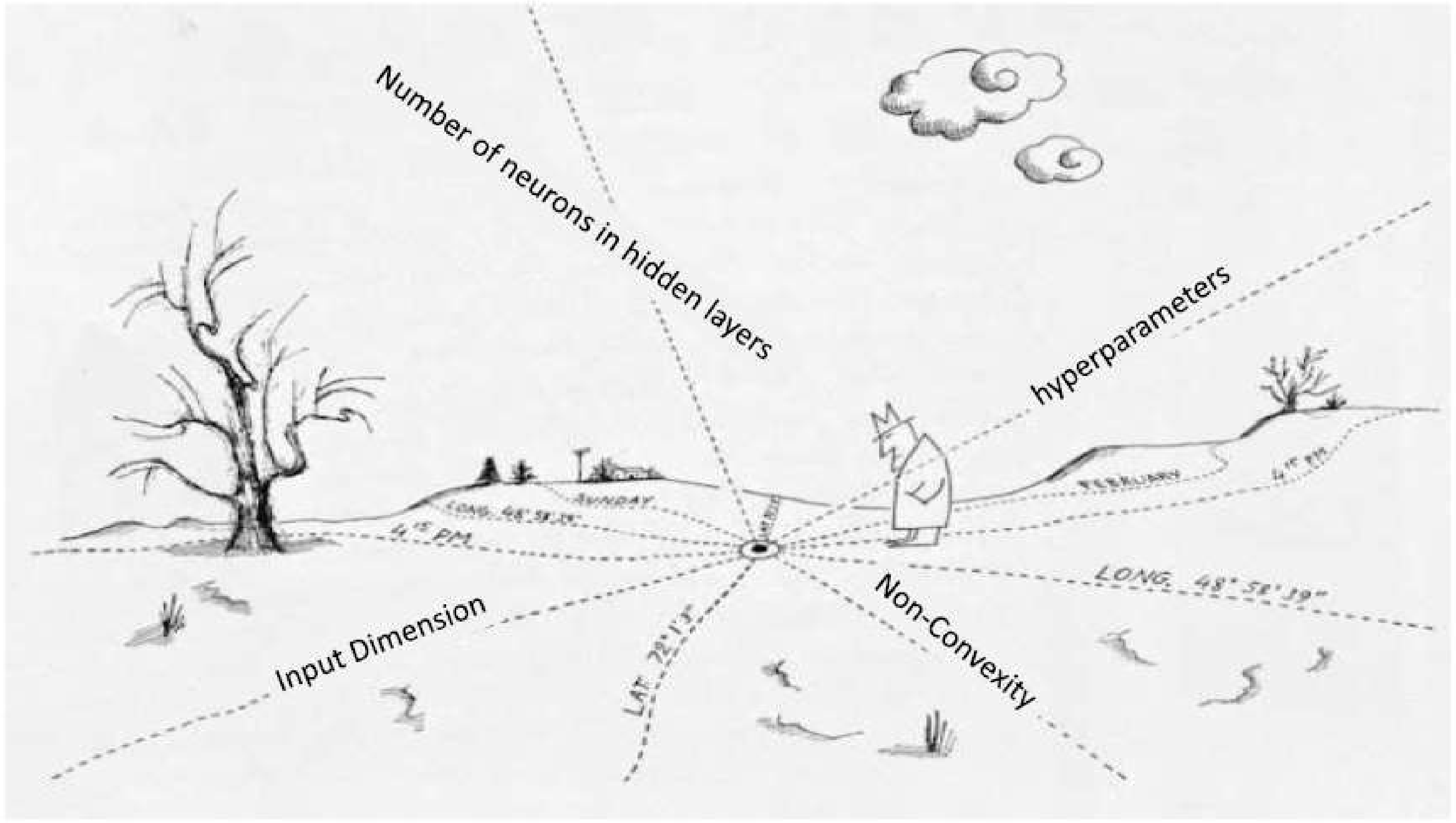
In Life: The Curse of Dimensionality Saul Steinberg 1968



Curse of Dimensionality

In Machine Learning

Bellman Adaptive Control Processes. Princeton University Press,1961.): the number of samples needed to estimate an arbitrary function with a given level of accuracy grows exponentially with respect to the number of input variables



Random Projection Neural Networks

Solving the Gordian Knot **of the curse of dimensionality in optimizaton**

- Schmidt et al. (1992): fixing the weights between the input and the hidden layer at random values, and by solving a linear problem for the output weights the approximation accuracy is equivalent to that obtained with back-propagation.
- Random Vector Functional-Link Networks (RVFLNs) were addressed in Pao et al. (1992) in which the input layer is directly connected also to the output layer, the internal weights are chosen randomly in $[-1, 1]$: the output weights are estimated in one step by solving a system of linear equations.
- Igelnik et al. (1995) proved that RVFLNs are universal approximators for continuous functions on bounded finite-dimensional sets.

Random Projection Neural Networks

Different versions of them

- Randomized and Random Vector Functional Link Networks (RVFLNs): Schmidt et al. (1992)
- Echo-State Neural Networks and Reservoir Computing: Jaeger (2002)
- Extreme Learning Machines: Huang (2006)

The keystone idea behind all these approaches is to use a fixed-weight configuration between the input and the hidden layer, fixed biases for the nodes of the hidden layer, and a linear output layer. Hence, the output is projected linearly onto the functional subspace spanned by the nonlinear basis functions of the hidden layer, and the only unknowns that have to be determined are the weights between the hidden and the output layer.

Huang (2014) An insight into extreme learning machines: random neurons, random features and kernels, Cognitive Computation, 6, 376–390.

Random Projection Neural Networks

The feasibility of this approach can been justified by the celebrated Johnson and Lindenstrauss (JL) Theorem:

Theorem (Johnson and Lindenstrauss)

Let $\mathcal{X}$ be a set of n points in $\mathbb{R}^d$. Then, $\forall\, \epsilon \in (0,1)$ and $k \in \mathbb{N}$ such that $k \geq O(\frac{\ln n}{\epsilon^2})$, there exists a map $\boldsymbol{F} : \mathbb{R}^d \rightarrow \mathbb{R}^k$ such that

$$(1-\epsilon)\|\boldsymbol{u}-\boldsymbol{v}\|_2^2 \leq \|\boldsymbol{F}(\boldsymbol{u})-\boldsymbol{F}(\boldsymbol{v})\|_2^2 \leq (1+\epsilon)\|\boldsymbol{u}-\boldsymbol{v}\|_2^2 \quad \forall\, \boldsymbol{u}, \boldsymbol{v} \in \mathcal{X}. \tag{17}$$

Note that while the above theorem is deterministic, its proof relies on probabilistic techniques combined with Kirszbraun's theorem to yield a so-called extension mapping.

In particular, it can be shown that one of the many such embedding maps is simply a linear projection matrix with suitable random entries. Then, the JL Theorem may be proved using the following lemma.

Lemma

Let $\boldsymbol{F}(\boldsymbol{u})$ be the random projection defined by

$$\boldsymbol{F}(\boldsymbol{u}) = \frac{1}{\sqrt{k}}, \boldsymbol{R}\,\boldsymbol{u}, \quad \boldsymbol{u} \in \mathbb{R}^d,$$

where $\boldsymbol{R} = [r_{ij}] \in \mathbb{R}^{k\times d}$ has components which are i.i.d. random variables sampled from a normal distribution. Then, $\forall\, \boldsymbol{u} \in \boldsymbol{X}$

$$(1-\epsilon)\|\boldsymbol{u}\|^2 \leq \|\boldsymbol{F}(\boldsymbol{u})\|^2 \leq (1+\epsilon)\|\boldsymbol{u}\|^2$$

is true with probability $p \geq 1 - 2\exp\left(-(\epsilon^2-\epsilon^3)\frac{k}{4}\right)$.

. Rahimi, B. Recht, Weighted sums of random kitchen sinks: replacing minimization with randomization in learning., in: Nips, 2008,263 pp. 1313–1320.

Theorem 3.4. (cf. Theorem 3.1 and 3.2 in [41])) Consider the parametric set activation functions on $\mathsf{X} \subseteq \mathbb{R}^d$, $\psi(\boldsymbol{x};\boldsymbol{\alpha}) : \mathsf{X} \times \mathsf{A} \to \mathbb{R}$ parametrized by random variables $\boldsymbol{\alpha}$ in A, that satisfy $\sup_{\boldsymbol{x},\boldsymbol{\alpha}} |\phi(\boldsymbol{x},\boldsymbol{\alpha})| \leq 1$. Let p be a probability distribution on A and μ be a probability measure on X and the corresponding norm $\|f\|_{L^2(\mu)} = \int_{\mathsf{X}} f(\boldsymbol{x})^2 \mu(d\boldsymbol{x})$. Define the set:

$$\mathsf{G}_p \equiv \left\{ g(\boldsymbol{x}) = \underbrace{\int_{\mathsf{A}} w(\boldsymbol{\alpha})\phi(\boldsymbol{x};\boldsymbol{\alpha})d\boldsymbol{\alpha}}_{\text{Function in RKHS}} : \|g\|_{p(\alpha)} < \infty \right\}, \ \|g\|_p := \sup_{\alpha \in \mathsf{A}} \|w(\boldsymbol{\alpha})/p(\boldsymbol{\alpha})\|. \tag{24}$$

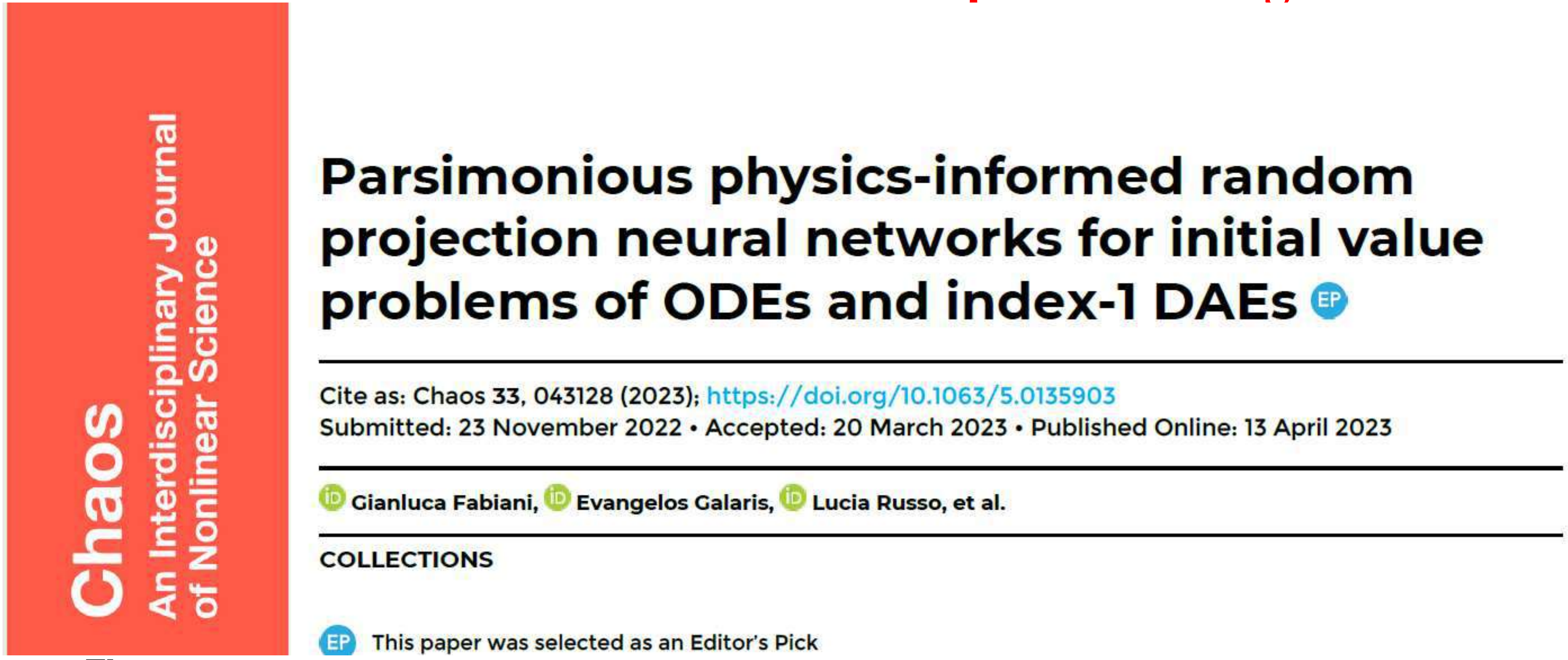
Fix a function g^* in G_p. Then, for any $\delta > 0$, there exist $N \in \mathbb{N}$, and $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_N$ of $\boldsymbol{\alpha}$ drawn i.i.d. from p, and a function $\hat{g}$ in the random set of finite sums

$$\hat{\mathsf{G}}_\alpha \equiv \left\{ \hat{g} : \hat{g}(\boldsymbol{x}) = \sum_{j=1}^{N} w_j \phi(\boldsymbol{x};\boldsymbol{\alpha}_j) \right\} \tag{25}$$

such that

$$\sqrt{\int_{\mathsf{X}} (g^*(\boldsymbol{x}) - \hat{g}(\boldsymbol{x}))^2 d\boldsymbol{\mu}(\boldsymbol{x})} \leq \frac{||g^*||_p}{\sqrt{N}} \left(1 + \sqrt{2\log\frac{1}{\delta}}\right), \tag{26}$$

The Forward Problem: think on Latent Spaces and High Dimensions



Chaos
An Interdisciplinary Journal of Nonlinear Science

Parsimonious physics-informed random projection neural networks for initial value problems of ODEs and index-1 DAEs EP

Cite as: Chaos 33, 043128 (2023); https://doi.org/10.1063/5.0135903
Submitted: 23 November 2022 • Accepted: 20 March 2023 • Published Online: 13 April 2023

Gianluca Fabiani, Evangelos Galaris, Lucia Russo, et al.

COLLECTIONS

EP This paper was selected as an Editor's Pick

Theorem

For an IVP problem in the canonical form or in the semi-explicit form for which the Picard-Lindelöf Theorem holds true, the PIRPNN solution Ψ_{Ni} with N Gaussian basis functions whose shape parameters α_{ji} are drawn i.i.d. from a uniform distribution across the sample space, converges uniformly to the actual solution profile $\boldsymbol{u}(t)$ in a closed time interval $[t_0 \quad t_{end}]$ with an upper bound of the order of $O(\frac{1}{\sqrt{N}})$ with a probability $1-\delta$ for any $\delta > 0$.

THE FORWARD PROBLEM

The Adaptive Scheme: Error Control

Let us assume we have solved the problem up to the interval $[t_{k-1}, t_k]$, hence we have found $u_i^{(k-1)}$ and we are seeking $u_i^{(k)}$ in the current interval $[t_k, t_{k+1}]$ with width $\Delta x_t = t_{k+1} - t_k$. If in the current interval the regularized Gauss-Newton scheme does not converge to a specific tolerance within a number of iterations (here set to 4),

thus redefining a new guess t_{k+1}^* for t_{k+1}:

$$\Delta t_k^* = 0.8\gamma \cdot \Delta t_k, \qquad \text{with} \qquad \gamma = \left(\frac{1}{err}\right)^{\frac{1}{\nu+1}}, \tag{31}$$

where

$$err = \left\| \frac{F(W^{o(\nu)})}{AbsTol + RelTol \cdot \frac{d\Psi^{(k)}}{dt}} \right\|_{l^2}, \tag{32}$$

THE FORWARD PROBLEM

Continuation method

Proposition

Let $\boldsymbol{\Psi}(t_k) \in \mathbb{R}^m$ be the solution found with PIRPNN at the end of the time interval $[t_{k-1} \quad t_k]$. Then, an initial guess for the weights of the PIRPNN for the time interval $[t_k \quad t_{k+1}]$ is given by:

$$\hat{\boldsymbol{W}}^o = \frac{d\boldsymbol{\Psi}(t_k)}{dt} \frac{\boldsymbol{\Phi}^T}{||\boldsymbol{\Phi}||_{l_2}^2}, \tag{33}$$

where $\hat{\boldsymbol{W}}^o \in \mathbb{R}^{m \times N}$ is the matrix with the initial guess of the output weights of the m PIRPNNs and $\boldsymbol{\Phi} \in \mathbb{R}^N$ is the vector containing the values of the random basis functions in the interval $[t_k \quad t_{k+1}]$.

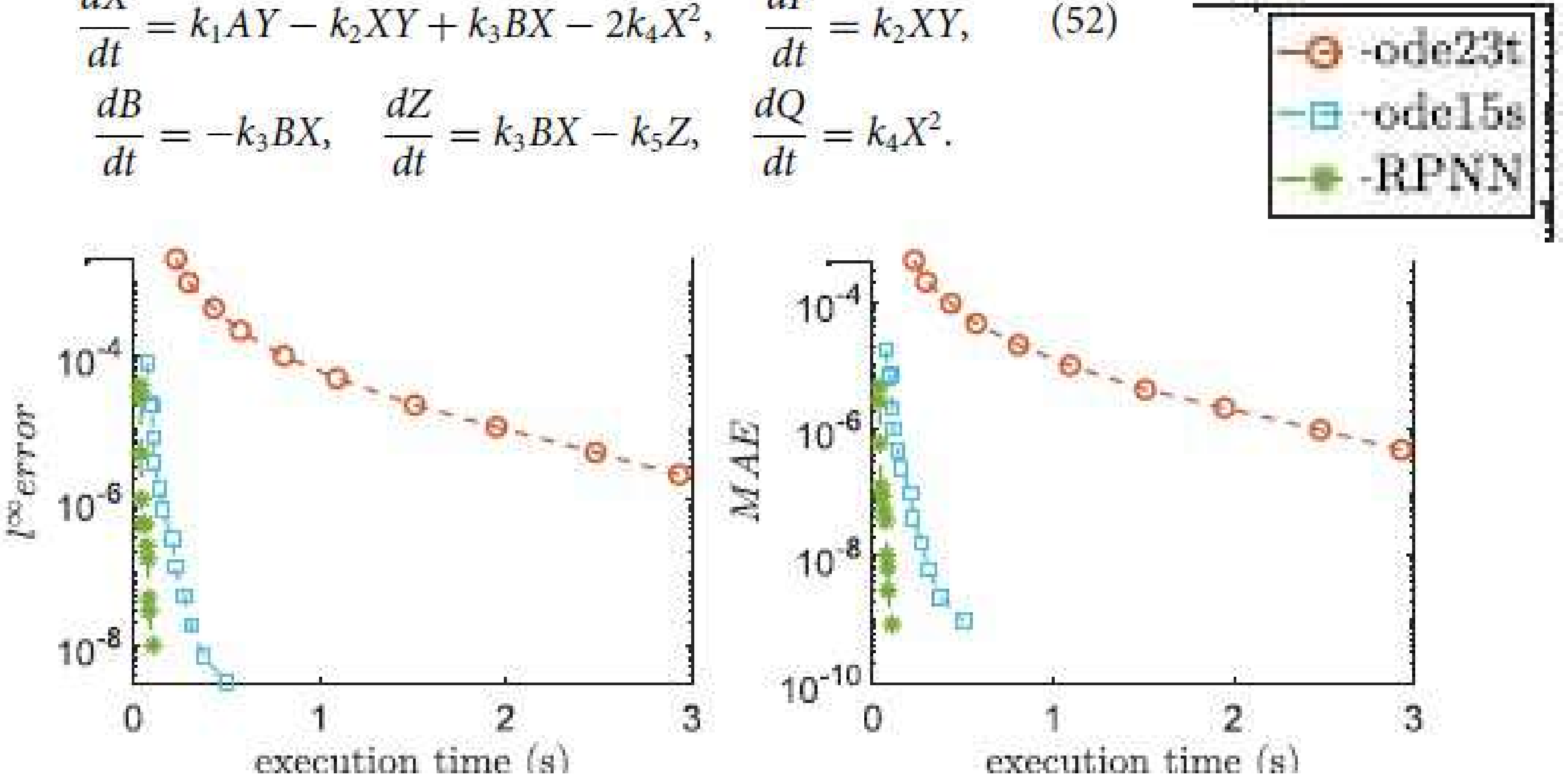
The Forward Problem: think on Latent Spaces and High Dimensions

The Belousov–Zhabotinsky chemical reactions model is given by the following system of seven ODEs:[61,86]

$$\frac{dA}{dt} = -k_1AY, \quad \frac{dY}{dt} = -k_1AY - k_2XY + k_5Z,$$

$$\frac{dX}{dt} = k_1AY - k_2XY + k_3BX - 2k_4X^2, \quad \frac{dP}{dt} = k_2XY, \tag{52}$$

$$\frac{dB}{dt} = -k_3BX, \quad \frac{dZ}{dt} = k_3BX - k_5Z, \quad \frac{dQ}{dt} = k_4X^2.$$



Comparison with DeepXDE

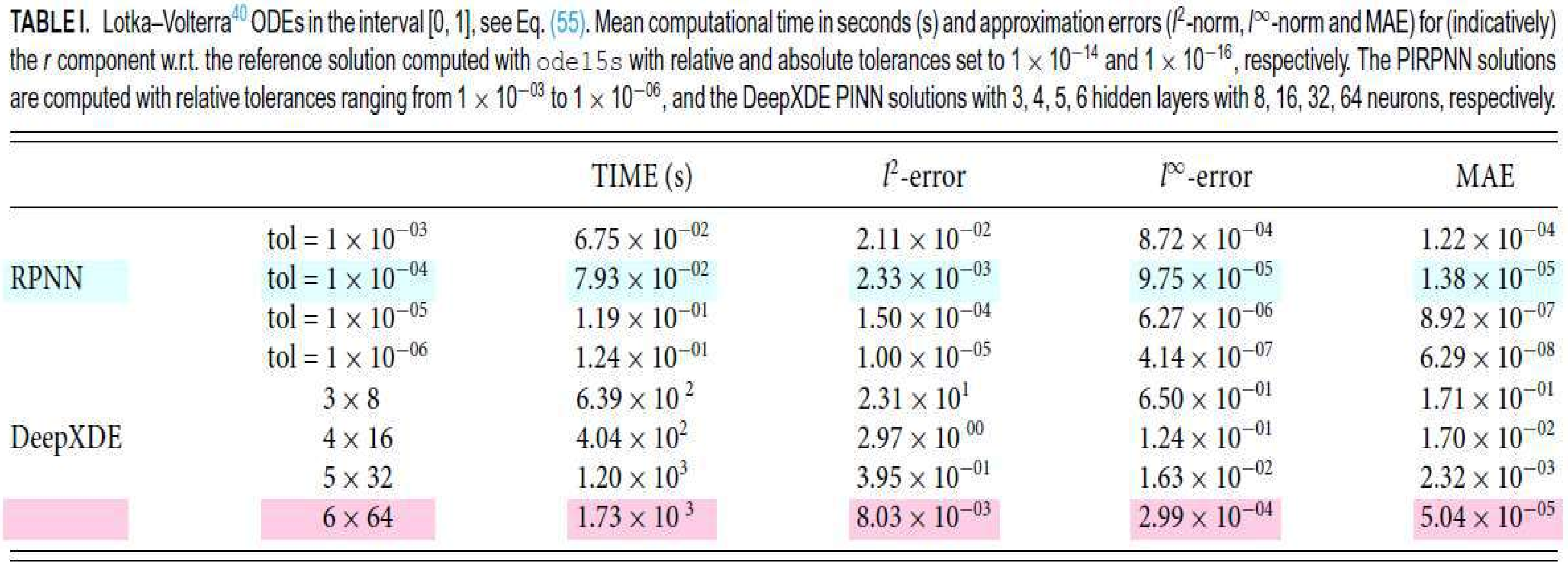


TABLE I. Lotka–Volterra[40] ODEs in the interval [0, 1], see Eq. (55). Mean computational time in seconds (s) and approximation errors (l^2-norm, l^∞-norm and MAE) for (indicatively) the r component w.r.t. the reference solution computed with `ode15s` with relative and absolute tolerances set to 1×10^{-14} and 1×10^{-16}, respectively. The PIRPNN solutions are computed with relative tolerances ranging from 1×10^{-03} to 1×10^{-06}, and the DeepXDE PINN solutions with 3, 4, 5, 6 hidden layers with 8, 16, 32, 64 neurons, respectively.

		TIME (s)	l^2-error	l^∞-error	MAE
	tol = 1×10^{-03}	6.75×10^{-02}	2.11×10^{-02}	8.72×10^{-04}	1.22×10^{-04}
RPNN	tol = 1×10^{-04}	7.93×10^{-02}	2.33×10^{-03}	9.75×10^{-05}	1.38×10^{-05}
	tol = 1×10^{-05}	1.19×10^{-01}	1.50×10^{-04}	6.27×10^{-06}	8.92×10^{-07}
	tol = 1×10^{-06}	1.24×10^{-01}	1.00×10^{-05}	4.14×10^{-07}	6.29×10^{-08}
	3×8	6.39×10^{2}	2.31×10^{1}	6.50×10^{-01}	1.71×10^{-01}
DeepXDE	4×16	4.04×10^{2}	2.97×10^{00}	1.24×10^{-01}	1.70×10^{-02}
	5×32	1.20×10^{3}	3.95×10^{-01}	1.63×10^{-02}	2.32×10^{-03}
	6×64	1.73×10^{3}	8.03×10^{-03}	2.99×10^{-04}	5.04×10^{-05}

Our scheme is around 20 000 times faster than DeepCDE for getting
a comparable numerical accuracy!

Numerical Solution of PDEs

$$L\tilde{u}_N(x_i; w) = f(\tilde{u}_N(x_i; w), \lambda), \quad i = 1, \dots, M_\Omega$$
$$B_l\tilde{u}_N(x_k; w) = g_l(x_k), \quad k = 1, \dots, M_l, \quad l = 1, \dots, m.$$

By approximating the solution with RPNNs, we get a system of M nonlinear equations (number of collocation points) with N unknowns (number of hidden neurons) that can be rewritten in a compact way as $F_k(w, \lambda) = 0, \quad k = 1, \dots, M$ where for $k = 1, \dots, M_\Omega$, we have:

$$F_k(w, \lambda) = L\left(\sum_{i=1}^{N} w_j \psi(\alpha_j \cdot x_i + \beta_j)\right) - f\left(\sum_{i=1}^{N} w_j \psi(\alpha_j \cdot x_i + \beta_j)\right) = 0,$$

while for the l-th boundary condition, for $k = 1, \dots, M_l$ we have:

$$F_k(w, \lambda) = B_l\left(\sum_{i=1}^{N} w_j \psi(\alpha_j \cdot x_i + \beta_j)\right) - g\left(\sum_{i=1}^{N} w_j \psi(\alpha_j \cdot x_i + \beta_j)\right) = 0.$$

Coupling with Numerical Bifurcation Theory Tools

Solution branches past saddle-node bifurcations (limit/turning points) can be traced by applying the so called "pseudo" arc-length continuation method (Chan and Keller, 1982). This involves the parametrization of both $\tilde{u}(w)$ and λ by the arc-length s on the solution branch. The solution is sought in terms of both $\tilde{u}(w; s)$ and $\lambda(s)$ in an iterative manner, by solving until convergence the following augmented system:

$$\begin{bmatrix} \nabla_w F & \nabla_\lambda F \\ \nabla_w N & \nabla_\lambda N \end{bmatrix} \cdot \begin{bmatrix} dw^{(n)}(s) \\ d\lambda^{(n)}(s) \end{bmatrix} = - \begin{bmatrix} F(w^{(n)}(s), \lambda(s)) \\ N(\tilde{u}(w^{(n)}; s), \lambda^{(n)}(s)) \end{bmatrix}, \qquad (36)$$

where

$$\nabla_\lambda F = \begin{bmatrix} \frac{\partial F_1}{\partial \lambda} & \frac{\partial F_2}{\partial \lambda} & \dots & \frac{F_M}{\partial \lambda} \end{bmatrix}^T,$$

$$N(\tilde{u}(w^{(n)}; s), \lambda^{(n)}(s)) =$$

$$(\tilde{u}(w^{(n)}; s) - \tilde{u}(w; s)_{-2})^T \cdot \frac{(\tilde{u}(w)_{-2} - \tilde{u}(w)_{-1})}{ds} +$$

$$(\lambda^{(n)}(s) - \lambda_{-1}) \cdot \frac{(\lambda_{-2} - \lambda_{-1})}{ds} - ds,$$

The one- and two-dimensional Liouville–Bratu–Gelfand Problem

The Liouville–Bratu–Gelfand model arises in many physical and chemical systems. It is an elliptic partial differential equation which in its general form is given by:

$$\Delta u(x) + \lambda e^{u(x)} = 0 \quad x \in \Omega, \tag{42}$$

with homogeneous Dirichlet conditions

$$u(x) = 0\,, \; x \in \partial\Omega. \tag{43}$$

The domain that we consider here is the $\Omega = [0,1]^d$ in R^d , $d = 1, 2$.

The one dimensional Liouville–Bratu–Gelfand Problem

Analytical Solution

The one-dimensional problem admits an analytical solution given by (Mohsen, 2014):

$$u(x) = 2\ln\frac{\cosh\theta}{\cosh\theta(1-2x)}, \text{ where } \theta \text{ is such that } \cosh\theta = \frac{4\theta}{\sqrt{2\lambda}}. \tag{44}$$

For $0 < \lambda < \lambda_c$ the problem admits two branches of solutions that meet at $\lambda_c \sim 3.513830719$, a limit point (saddle-node bifurcation) that marks the onset of two branches of solutions with different stability, while beyond that point no solutions exist.

The one dimensional Liouville–Bratu–Gelfand Problem

Numerical Accuracy

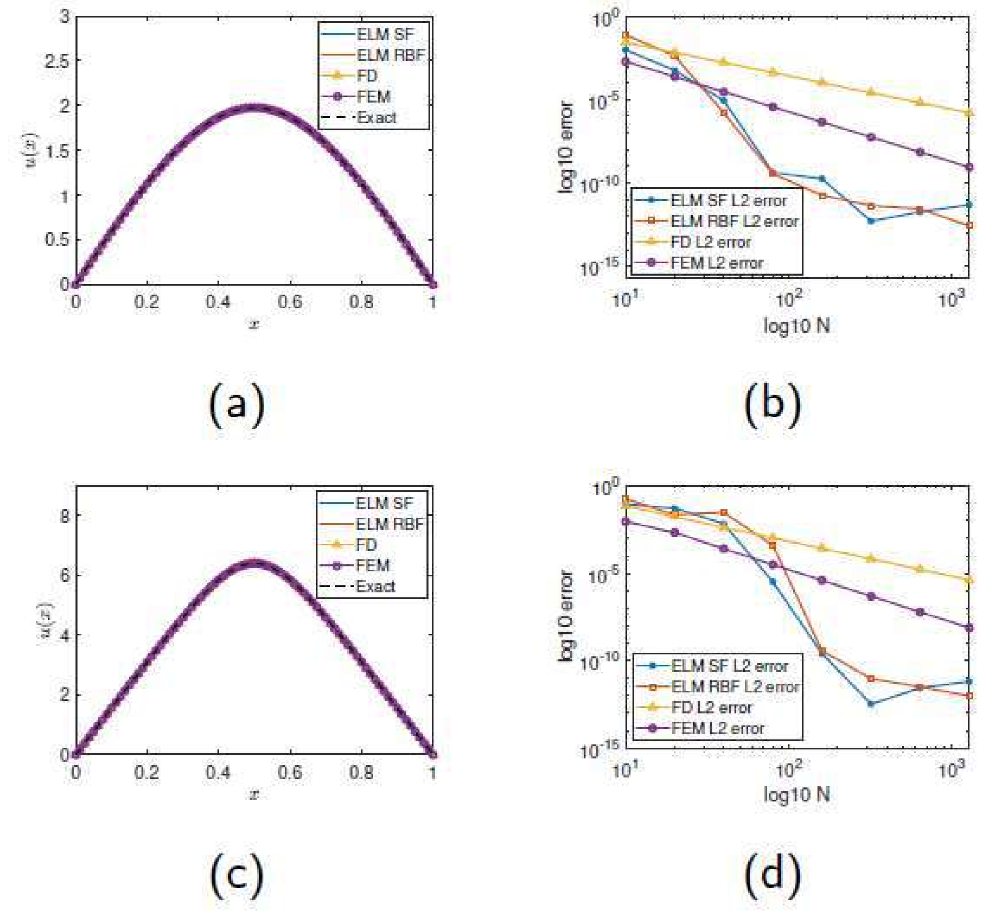


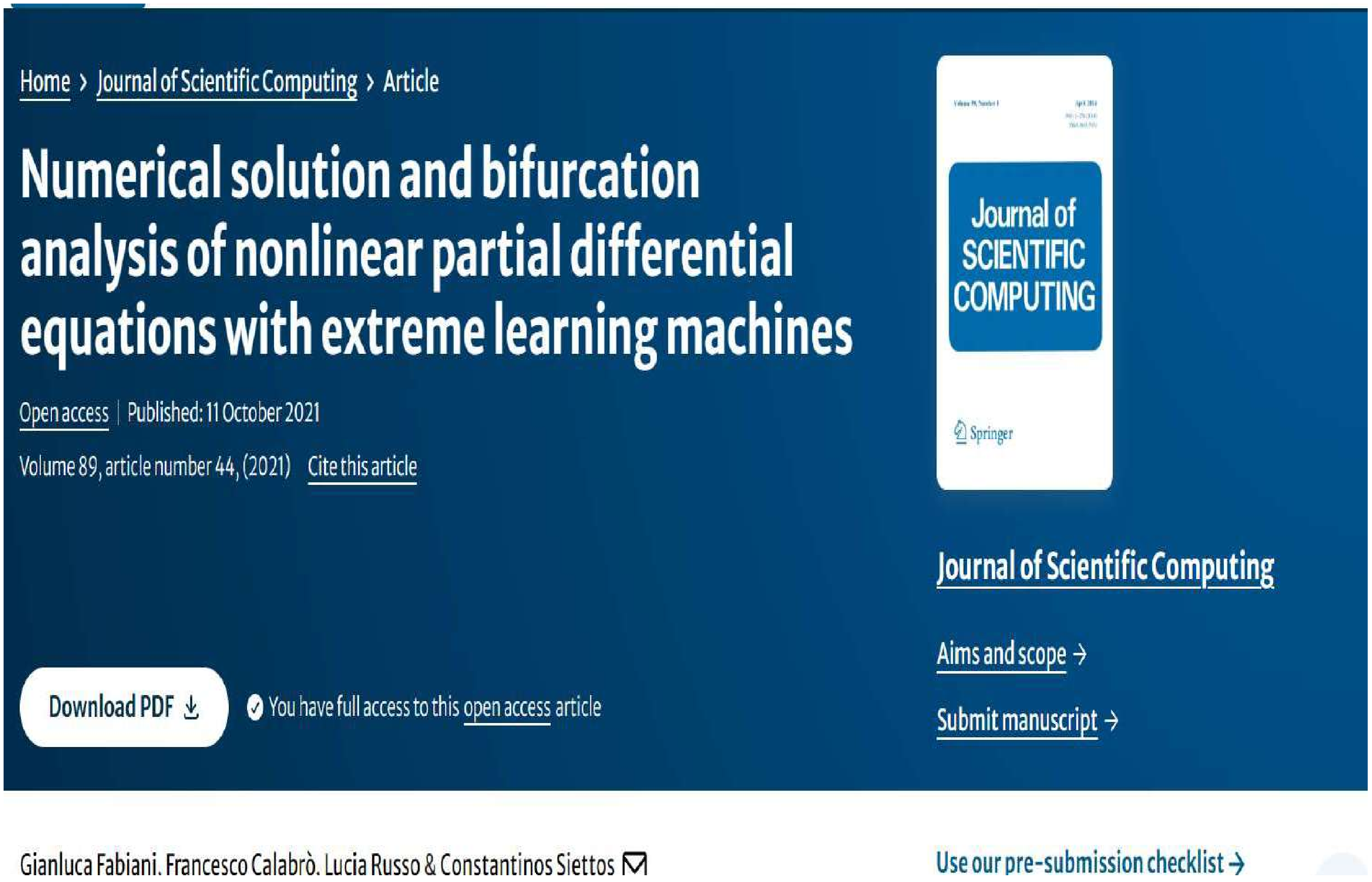
Figure: (a) unstable solution at $\lambda = 3$ for $N = 40$. (b) L_2 with respect to the exact unstable solution at $\lambda = 3$ w.r.t N. (c) unstable solution at $\lambda = 0.2$ $N = 40$. (d) L_2 with respect to the exact unstable solution at $\lambda = 0.2$ w.r.t N.

The one dimensional Liouville–Bratu–Gelfand Problem

Execution Times

	ELM SF			ELM RBF		
N	5%	mean	95%	5%	mean	95%
80	7.16e-03	8.14e-03	9.27e-03	2.07e-03	2.31e-03	2.61e-03
160	3.81e-02	4.23e-02	4.93e-02	3.53e-03	4.31e-03	4.96e-03
320	1.09e-02	1.16e-02	1.30e-02	7.12e-03	7.96e-03	8.57e-03
640	3.19e-02	3.38e-02	3.58e-02	2.81e-02	3.01e-02	3.17e-02
	FD			FEM		
N	5%	mean	95%	5%	mean	95%
80	1.93e-04	2.60e-04	2.65e-04	2.58e-03	2.88e-03	3.03e-03
160	5.72e-04	7.01e-04	8.24e-04	5.76e-03	6.32e-03	6.89e-03
320	1.59e-03	1.86e-03	2.08e-03	1.15e-02	1.17e-02	1.20e-02
640	8.77e-03	9.07e-03	9.49e-03	3.00e-02	3.11e-02	3.21e-02

Table: Execution times for the Bratu PDE with Dirichlet boundary conditions and $\lambda = 1$.



Home › Journal of Scientific Computing › Article

Numerical solution and bifurcation analysis of nonlinear partial differential equations with extreme learning machines

Open access | Published: 11 October 2021

Volume 89, article number 44, (2021) Cite this article

You have full access to this open access article

Journal of Scientific Computing

Aims and scope →

Submit manuscript →

Gianluca Fabiani. Francesco Calabrò. Lucia Russo & Constantinos Siettos

Use our pre-submission checklist →

Solving the Inverse Problem

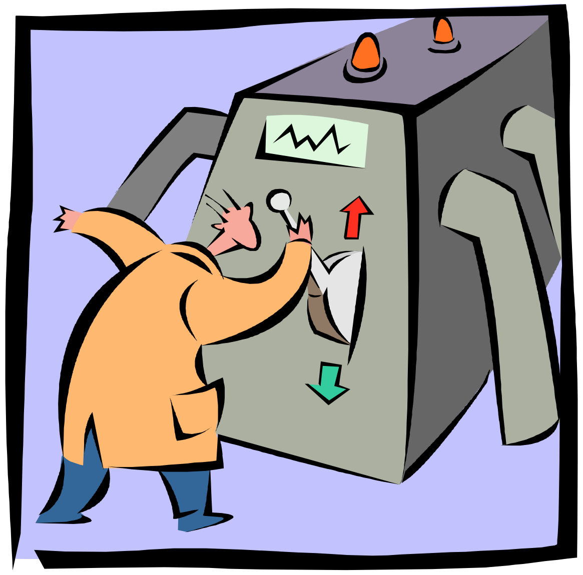
HAVE MODEL

ODES/PDEs/SDEs

NUMERICAL ANALYSIS
DESIGN CONTROLLERS

ODES/PDEs/SDEs

SURROGATE MODELS
DISCRETE MAPS



HAVE EXPERIMENT
DATA

PERFORM IDENTIFICATION
MACHINE LEARNING

Solving the Inverse Problem with MACHINE LEARNING

González-García, Rico-Martìnez, Kevrekidis, Identification of distributed parameter systems: A neural net based approach, Computers & chemical engineering, 22,S965–S968,1998

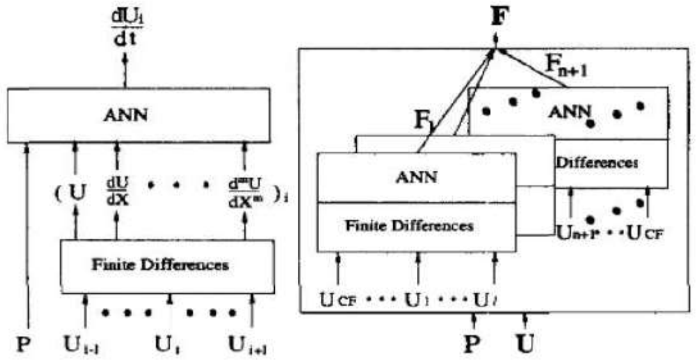


Figure 2: Evaluation of the right-hand-side (RHS) of the set of ODEs: On the left, at a point and on the right, for all points of the grid to form the vector of RHS F.

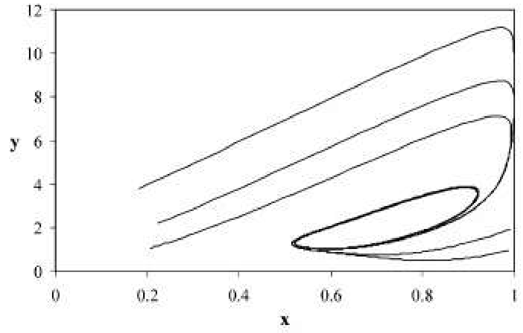
U^{k+1}

ELSEVIER

Fuzzy Sets and Systems 126 (2002) 89–104

FUZZY sets and systems

www.elsevier.com/locate/fss



Truncated Chebyshev series approximation of fuzzy systems for control and nonlinear system identification

C.I. Siettos, G.V. Bafas, A.G. Boudouvis *

HAVE MODELS. Classical Feedback Linearization: The idea

Consider the problem of designing a controller for a nonlinear system of the form

$$x(k+1) = f(\mathrm{x}(k), u(k))$$

Transform the nonlinear equations into a linear system by a means of feedback and/or change of variables. After this a linear stabilizing state feedback is designed

2-Step Implementation:

1. Find a state transformation $z = T(x)$ and an input transformation $u = \Psi(x,v)$ so that the nonlinear system dynamics is transformed into a equivalent linear time-invariant dynamics controllable system of the form

$$\mathrm{z}(k+1) = \mathrm{Az}(k) + \mathrm{b}\,v(k)$$

2. Use standard linear techniques (e.g. pole placement) to design $v\ (k)\ = -\ K\ z(k)$

However, applicability is severely limited by a set of rather restrictive conditions

(Isidori, Nonlinear control systems, 3rd Ed, Springer, 1995)

Feedback Linearization in 1 step

$$x(k+1) = f(x(k), u(k))$$

The motivation: By-pass the restrictive conditions → Overcome step 1: intermediate step of transforming the original nonlinear system into a linear controllable one with an external reference input

The proposed idea: 1-Step Implementation: Seek to simultaneously implement a nonlinear operator $z = T(x)$ **and**

a state feedback control law $u = -c\, z = -c\, T(x)$

that induce linear closed loop dynamics in a <u>single-step</u>: $z(k+1) = A\, z(k),$

$$z(k+1) = T(x(k+1)) = T(f(x(k), -cT(x(k)))) = A\, \boldsymbol{z(k)} = \boldsymbol{AT(x)}$$

Solve the system of Nonlinear Functional Equations (NFE's)

$$T(\boldsymbol{f}(x, -cT(x))) = AT(x),$$
$$T(0) = 0$$

Under a set of assumptions, the above system of NFE's admits a unique locally analytic and invertible solution

Feedback Linearization in 1 step: Assumptions

The following assumptions are made:

Assumption 1: *The $(n \times n)$ matrix $\mathcal{C}$:*

$$\mathcal{C} = [G|JG|...|J^{n-1}G] \qquad J = \frac{\partial f}{\partial x}(0,0) \qquad G = \frac{\partial f}{\partial u}(0,0) \neq 0$$

has rank n: rank$(\mathcal{C}) = n$ (local controllability rank condition).

Assumption 2: *The eigenspectrum $\sigma(A)$ of matrix A comprises eigenvalues: $k_i \in \sigma(A)$, $i = 1, ...n$ that all lie inside the unit disc on the complex plane (Poincaré domain).*

Assumption 3: *The eigenspectra $\sigma(A), \sigma(J)$ of matrices A and J respectively are disjoint: $\sigma(A) \cap \sigma(J) = \emptyset$.*

Assumption 4: *The eigenvalues k_i of A are not related to the eigenvalues λ_j of the Jacobian matrix J through any equations of the type:*

$$\prod_{i=1}^{n} k_i^{m_i} = \lambda_j$$

$(j = 1, ..., n)$, where all the m_i's are non-negative integers that satisfy the condition:

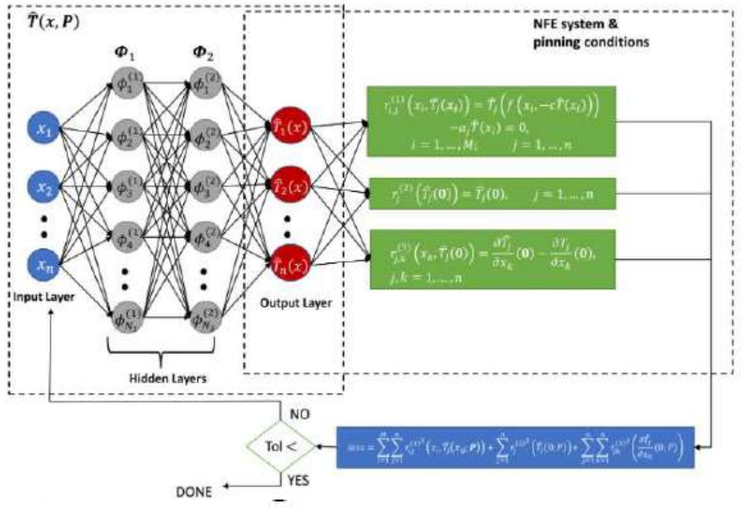
$$\sum_{i=1}^{n} m_i > 0$$

Assumption 5: *The pair of matrices (c, A) is chosen such that the following matrix O:*

$$O = \begin{bmatrix} c \\ cA \\ . \\ . \\ cA^{n-1} \end{bmatrix}$$

has rank n: rank$(O) = n$ (observability rank condition on the (c, A) pair).

Feedback Linearization in 1 step via Physics-Informed Neural Networks



*Vargas Alvarez H., Fabiani, F., Kazantzis, N., Siettos, C., Kevrekidis, I.G., 2023, Discrete-Time Nonlinear Feedback Linearization via Physics-Informed Machine Learning. **Journal of Computational Physics**, 478, 111953.*

$$\mathcal{L}(P) = \sum_{i=1}^{M}\sum_{j=1}^{n} {r_{ij}^{(1)}}^2(x_{ij}, \hat{T}(x_{ij};P)) + \sum_{j=1}^{n} {r_j^{(2)}}^2(\hat{T}_j(0;P)) + \sum_{j=1}^{n}\sum_{k=1}^{n} {r_{jk}^{(3)}}^2(\frac{\partial \hat{T}_j}{\partial x_k}(0;P))$$

$$r_{ij}^{(1)}(x_i, \hat{T}(x_i)) = \hat{T}_j\left(f(x_i, -c\hat{T}(x_i))\right) - \alpha_j \hat{T}(x_i), \qquad i = 1,\dots,M, \quad j = 1,2,\dots,n$$

where α_j is the j-th row of the matrix A and $\hat{T}_j$ is j-th output component of $\hat{T}$, and:

$$r_j^{(2)}(\hat{T}_j(0)) = \hat{T}_j(0), \quad r_{jk}^{(3)}(x_k, \hat{T}_j(0)) = \frac{\partial \hat{T}_j}{\partial x_k}(0) - \frac{\partial T_j}{\partial x_k}(0), \quad j,k = 1,2,\dots,n,$$

where $\frac{\partial T_j}{\partial x_k}(0)$ is the (j,k)-th element of the Jacobian matrix of $T(x)$ computed at the equilibrium, obtained by solving the system of equations in (14).Notice that for our illustrations, in the loss function, we consider all three terms equally weighted.

$$\frac{\partial T}{\partial x}(0)\frac{\partial f}{\partial x}(0,0) - A\frac{\partial T}{\partial x}(0) = \frac{\partial T}{\partial x}(0)\frac{\partial f}{\partial u}(0,0)c\frac{\partial T}{\partial x}(0).$$

Finally, in order to compute the Jacobian matrix $\frac{\partial \hat{T}}{\partial x}$, let's consider the j-th component, say $\hat{T}_j$, of the transformation $\tilde{T} = (\hat{T}_1, \hat{T}_2, \dots, \hat{T}_j, \dots, \hat{T}_n)$, so that the element (j,k) of the Jacobian matrix is given by:

$$\frac{\partial}{\partial x_k}\hat{T}_j(x) = \sum_{i=1}^{N_2} W^o_{ij}\phi_i^{(2)'}\left(\sum_{s=1}^{N_1} W^{(2)}_{si}\phi_s^{(1)}\left(\sum_{h=1}^{n} W^{(1)}_{hs}x_h + \beta_s^{(1)}\right) + \beta_i^{(2)}\right)\left(\sum_{s=1}^{N_1} W^{(2)}_{si}W^{(1)}_{ks}\phi_s^{(1)'}\left(\sum_{h=1}^{n} W^{(1)}_{hs}x_h + \beta_s^{(1)}\right)\right) \tag{24}$$

and equivalently, in matrix form is expressed as follows:

$$\frac{\partial \hat{T}_j}{\partial x_k} = W^{j(o)T} \cdot \left(\Phi_2'(W^{(2)T}\Phi_1(W^{(1)T}x + \beta^{(1)}) + \beta^{(2)}) \odot \left(W^{(2)T} \cdot (W_k^{(1)T} \odot \Phi_1'(W^{(1)T}x + \beta^{(1)}))\right)\right), \tag{25}$$

where $W^{j(o)}$ is the j-th column of the matrix $W^{(o)}$.

The derivative of the loss function w.r.t. an unknown parameter, say, $p \in \boldsymbol{P}$ is calculated as follows:

$$\frac{\partial \mathcal{L}(\boldsymbol{P})}{\partial p} = \sum_{i=1}^{M}\sum_{j=1}^{n} r_{ij}^{(1)}\frac{\partial r_{ij}^{(1)}}{\partial p} + \sum_{j=1}^{n} r_j^{(2)}\frac{\partial r_j^{(2)}}{\partial p} + \sum_{j=1}^{n}\sum_{k=1}^{n} r_{jk}^{(3)}\frac{\partial r_{jk}^{(3)}}{\partial p}, \tag{26}$$

Hence, for the three residuals $r^{(i)}, i = 1, 2, 3$, we have:

$$\frac{\partial r_{ij}^{(1)}}{\partial p} = \frac{\partial \hat{T}_j\left(f(x_i, -c^T\hat{T}(x_i))\right)}{\partial p}\frac{\partial f(x_i, -c^T\hat{T}(x_i))}{\partial u} \cdot (-c^T\frac{\partial \hat{T}(x_i))}{\partial p}) - {\alpha_j}^T\frac{\partial \hat{T}(x_i)}{\partial p} \tag{27}$$

$$\frac{\partial r_j^{(2)}}{\partial p} = \frac{\partial \hat{T}_j(x_0)}{\partial p}, \tag{28}$$

$$\frac{\partial r_{jk}^{(3)}(x_k, \hat{T}_j(\boldsymbol{x}_0))}{\partial p} = \frac{\partial^2 \hat{T}_j}{\partial p \partial x_k}(x_0) \tag{29}$$

$$i = 1, \dots, M, \quad j, k = 1, \dots, n, \quad p \in \boldsymbol{P} \tag{30}$$

- for $p = W_{hq}^{(o)}$:

$$\frac{\partial \hat{T}_j(x)}{\partial W_{hq}^{(o)}} = \begin{Bmatrix} \phi_h^{(2)}\left(\sum_{s=1}^{N_1} W_{sh}^{(2)}\phi_s^{(1)}\left(\sum_{k=1}^{n} W_{ks}^{(1)}x_k + \beta_s^{(1)}\right) + \beta_h^{(2)}\right) & if & q = j \\ 0 & if & q \neq j \end{Bmatrix} \tag{31}$$

- for $p = W_{hq}^{(2)}$

$$\frac{\partial \hat{T}_j(x)}{\partial W_{hq}^{(2)}} = W_{hj}^{o}\phi_h^{(2)\prime}\left(\sum_{s=1}^{N_1} W_{sh}^{(2)}\phi_s^{(1)}\left(\sum_{k=1}^{n} W_{ks}^{(1)}x_k + \beta_s^{(1)}\right) + \beta_h^{(2)}\right)\phi_q^{(1)}\left(\sum_{k=1}^{n} W_{ks}^{(1)}x_k + \beta_s^{(1)}\right) \tag{32}$$

- for $p = W_{hq}^{(1)}$

$$\frac{\partial \hat{T}_j(x)}{\partial W_{hq}^{(1)}} = \sum_{i=1}^{N_2} W_{ij}^{o}\phi_i^{(2)\prime}\left(\sum_{s=1}^{N_1} W_{si}^{(2)}\phi_s^{(1)}\left(\sum_{k=1}^{n} W_{ks}^{(1)}x_k + \beta_s^{(1)}\right) + \beta_i^{(2)}\right)\left(W_{hi}^{(2)}x_q\phi_h^{(1)\prime}\left(\sum_{k=1}^{n} W_{kh}^{(1)}x_k + \beta_h^{(1)}\right)\right) \tag{33}$$

- for $p = \beta_h^{(o)}$

$$\frac{\partial \hat{T}_j(x)}{\partial \beta_h^{(o)}} = \begin{Bmatrix} 1 & if & h = j \\ 0 & if & h \neq j \end{Bmatrix} \tag{34}$$

- for $p = \beta_h^{(2)}$

$$\frac{\partial \hat{T}_j(x)}{\partial \beta_h^{(2)}} = W_{hj}^{o}\phi_h^{(2)}\left(\sum_{s=1}^{N_1} W_{sh}^{(2)}\phi_s^{(1)}\left(\sum_{k=1}^{n} W_{ks}^{(1)}x_k + \beta_s^{(1)}\right) + \beta_h^{(2)}\right) \tag{35}$$

- for $p = \beta_h^{(1)}$

$$\frac{\partial \hat{T}_j(x)}{\partial \beta_h^{(1)}} = \sum_{i=1}^{N_2} W_{ij}^{o}\phi_i^{(2)\prime}\left(\sum_{s=1}^{N_1} W_{si}^{(2)}\phi_s^{(1)}\left(\sum_{k=1}^{n} W_{ks}^{(1)}x_k + \beta_s^{(1)}\right) + \beta_i^{(2)}\right)\left(W_{hi}^{(2)}\phi_h^{(1)}\left(\sum_{k=1}^{n} W_{kh}^{(1)}x_k + \beta_h^{(1)}\right) + \beta_i^{(2)}\right). \tag{36}$$

Feedback Linearization in 1 step via PINNs: The problem

$$x_1(t+1) = exp(0.3x_2(t))\sqrt{(1+x_1(t)+x_2(t))} - 1 - 0.4x_2(t) + 0.5u(t)$$
$$x_2(t+1) = 0.5\ln(1+x_1(t)+x_2(t)) + 0.4x_2(t)$$

$T(\Phi(x,\text{-c}T(x)))=AT(x)$, $T(0)=0$

The Jacobian matrix of the above system at the equilibrium $(0,0)$ is $\frac{\partial f}{\partial x}(0,0) = \begin{bmatrix} 0.5 & 0.4 \\ 0.5 & 0.9 \end{bmatrix}$, and its eigenvalues $\lambda_1 = 0.2101$ and $\lambda_2 = 1.1899$. The matrix A is chosen to be $A = \begin{bmatrix} 0.5 & 0.3 \\ 0.5 & 0.4 \end{bmatrix}$, with eigenvalues $k_1 = 0.8405$ and $k_2 = 0.0595$. Due to the choice of matrix A, its eigenvalues are not related to the eigenvalues of the Jacobian matrix $\frac{\partial f}{\partial x}(0,0)$ through any equations of the type (5), (6). Moreover, the following row vector c was chosen:

$$c = [1 \quad 0].$$

$$T_1(exp(0.3x_2(t))\sqrt{(1+x_1(t)+x_2(t))} - 1 - 0.4x_2(t) - 0.5T_1,$$
$$0.5\ln(1+x_1(t)+x_2(t)) + 0.4x_2(t)) = 0.5T_1 + 0.3T_2$$
$$T_2(exp(0.3x_2(t))\sqrt{(1+x_1(t)+x_2(t))} - 1 - 0.4x_2(t) - 0.5T_1,$$
$$0.5\ln(1+x_1(t)+x_2(t)) + 0.4x_2(t)) = 0.5T_1 + 0.4T_2$$
$$T_1(0,0) = 0$$
$$T_2(0,0) = 0,$$

Feedback Linearization via PINNs: The problem

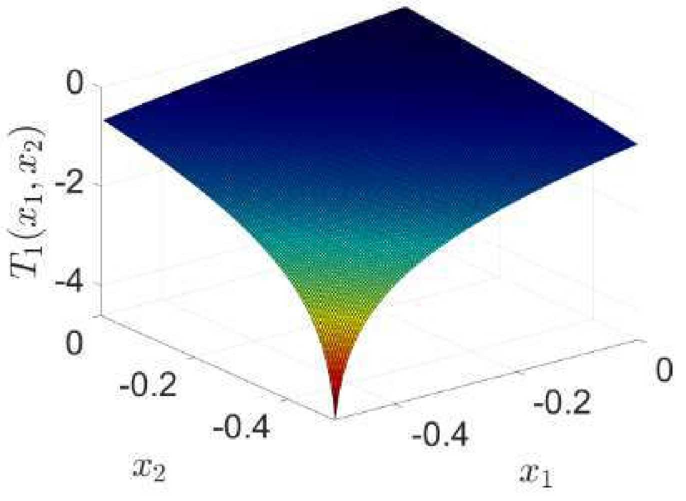
$$\frac{\partial T}{\partial x}(0,0) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix},$$

with a $det[T] \neq 0$, results in a locally invertible around the equilibrium point $(x_1, x_2) = (0,0)$ solution, that can be also calculated analytically in closed-form :

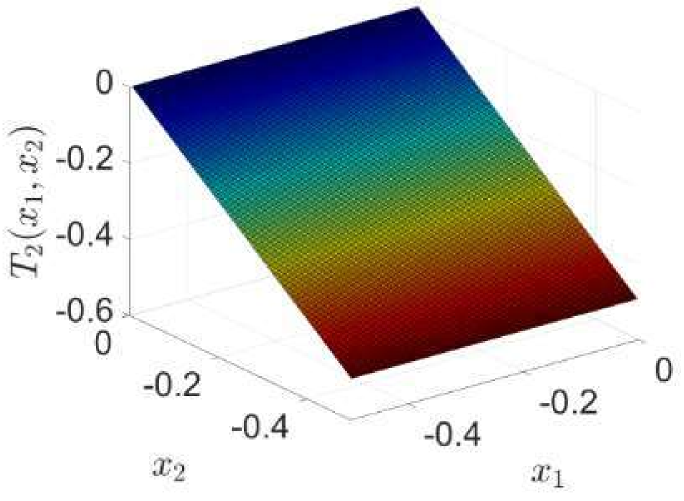
$$T_1(x_1, x_2) = \ln(1 + x_1 + x_2), \quad T_2(x_1, x_2) = x_2.$$

the proposed feedback-linearizing and pole-placing nonlinear feedback control law can be explicitly written as follows:

$$u = -cT(x) = -\ln(1 + x_1 + x_2).$$



(a)



(b)

Figure 2: Analytical solution of the NFEs (39) in $[-0.495, 0] \times [-0.495, 0]$. (a) $T_1(x_1, x_2) = \ln(1 + x_1 + x_2)$. A steep-gradient at $(-0.495, -0.495)$ is due to the presence of a singular point at $(x_1, x_2) = (-0.5, -0.5)$. (b) $T_2(x_1, x_2) = x_2$.

Training with Tensor Flow/Keras & Home-Made Matlab code

Automatic Differentiation
2 hidden layers,
5 neurons each,
BFGS

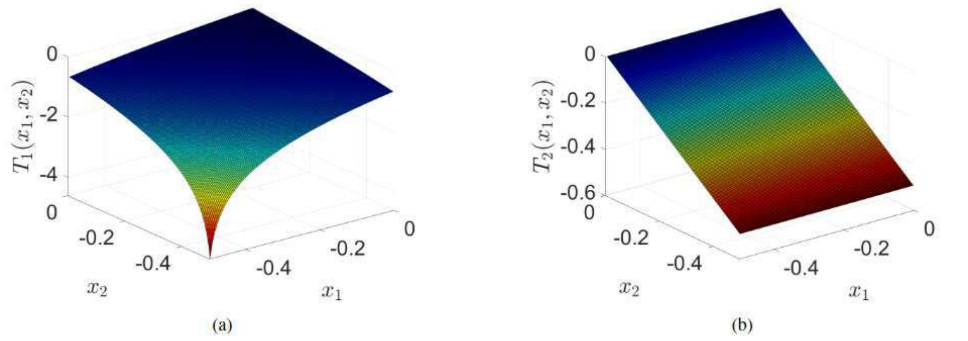
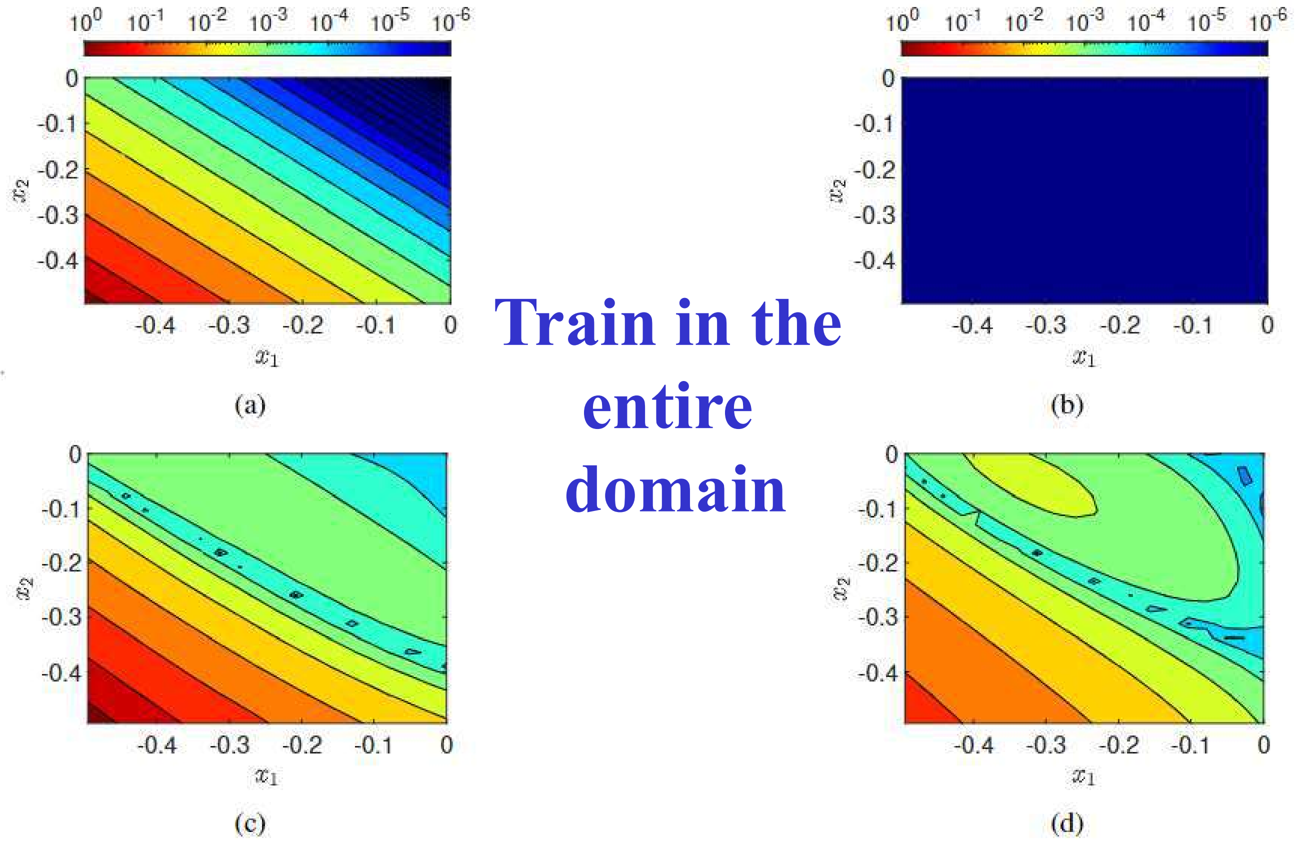


Figure 2: Analytical solution of the NFEs (39) in $[-0.495, 0] \times [-0.495, 0]$. (a) $T_1(x_1, x_2) = \ln(1 + x_1 + x_2)$. A steep-gradient at $(-0.495, -0.495)$ is due to the presence of a singular point at $(x_1, x_2) = (-0.5, -0.5)$. (b) $T_2(x_1, x_2) = x_2$.

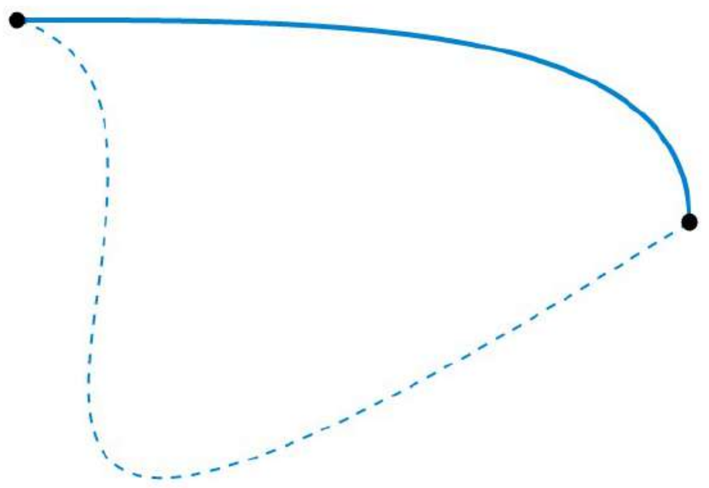


Train in the entire domain

6-th order Polynomial Expansion

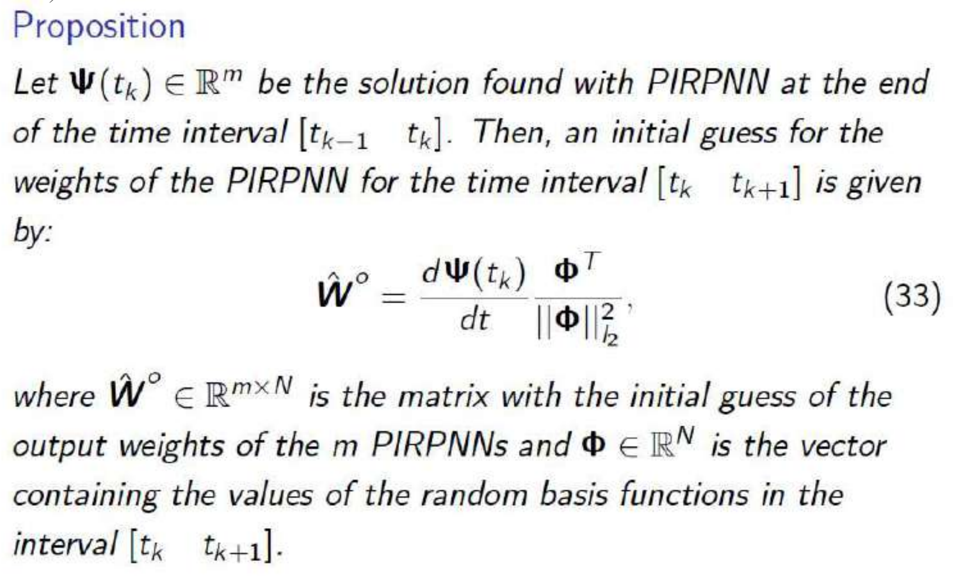
PINN Tensor Flow/Keras

Numerical Analysis and Continuation Matters!



Integrate concepts from Numerical Analysis & Dynamical Systems
- Homotopy
- Continuation methods

Gianluca Fabiani, Evangelos Galaris, Lucia Russo, and Constantinos Siettos Parsimonious physics-informed random projection neural networks for initial value problems of ODEs and index-1 DAEs, Chaos 33, 043128 (2023) (Editor's Pick)

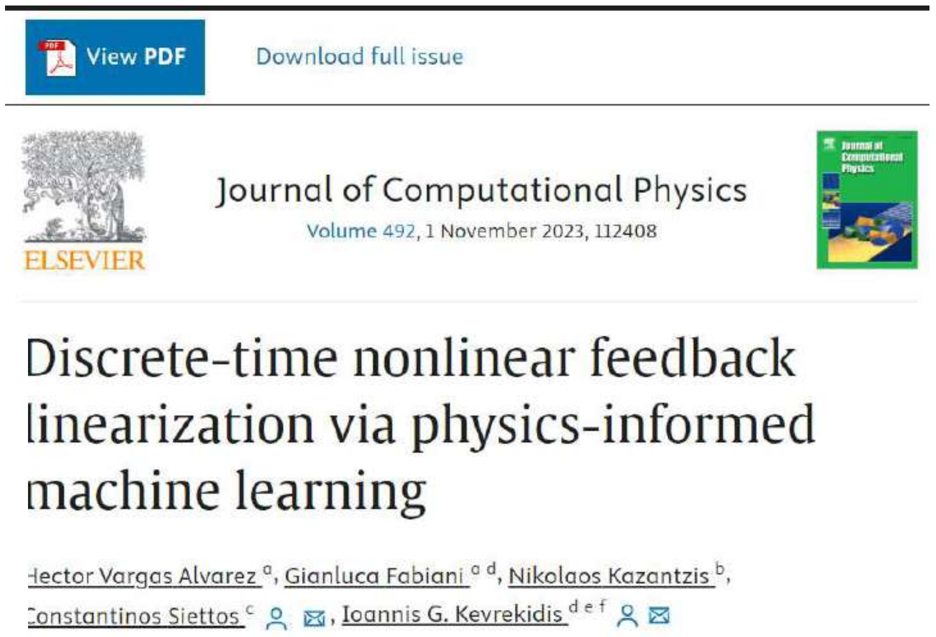


Proposition

Let $\mathbf{\Psi}(t_k) \in \mathbb{R}^m$ be the solution found with PIRPNN at the end of the time interval $[t_{k-1} \quad t_k]$. Then, an initial guess for the weights of the PIRPNN for the time interval $[t_k \quad t_{k+1}]$ is given by:

$$\hat{\mathbf{W}}^o = \frac{d\mathbf{\Psi}(t_k)}{dt}\frac{\mathbf{\Phi}^T}{||\mathbf{\Phi}||^2_{l_2}}, \qquad (33)$$

where $\hat{\mathbf{W}}^o \in \mathbb{R}^{m \times N}$ is the matrix with the initial guess of the output weights of the m PIRPNNs and $\mathbf{\Phi} \in \mathbb{R}^N$ is the vector containing the values of the random basis functions in the interval $[t_k \quad t_{k+1}]$.



View PDF Download full issue

ELSEVIER

Journal of Computational Physics
Volume 492, 1 November 2023, 112408

Discrete-time nonlinear feedback linearization via physics-informed machine learning

Hector Vargas Alvarez [a], Gianluca Fabiani [a d], Nikolaos Kazantzis [b], Constantinos Siettos [c], Ioannis G. Kevrekidis [d e f]

Numerical Analysis and Continuation Matters!

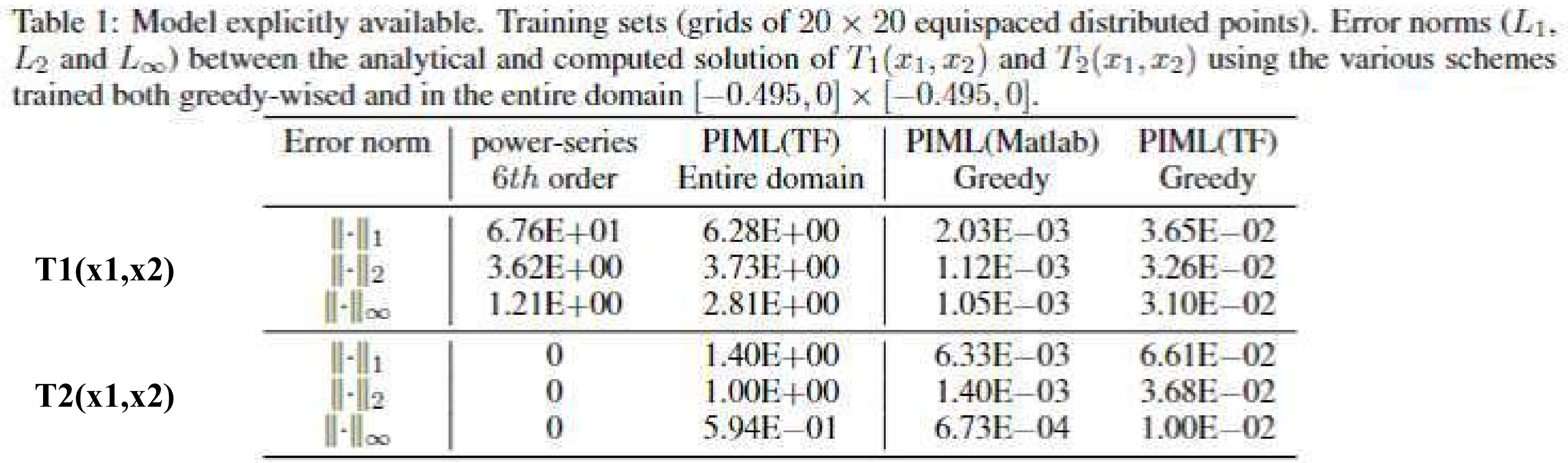


Table 1: Model explicitly available. Training sets (grids of 20×20 equispaced distributed points). Error norms (L_1, L_2 and L_∞) between the analytical and computed solution of $T_1(x_1, x_2)$ and $T_2(x_1, x_2)$ using the various schemes trained both greedy-wised and in the entire domain $[-0.495, 0] \times [-0.495, 0]$.

	Error norm	power-series 6th order	PIML(TF) Entire domain	PIML(Matlab) Greedy	PIML(TF) Greedy
T1(x1,x2)	$\lVert\cdot\rVert_1$	6.76E+01	6.28E+00	2.03E−03	3.65E−02
	$\lVert\cdot\rVert_2$	3.62E+00	3.73E+00	1.12E−03	3.26E−02
	$\lVert\cdot\rVert_\infty$	1.21E+00	2.81E+00	1.05E−03	3.10E−02
T2(x1,x2)	$\lVert\cdot\rVert_1$	0	1.40E+00	6.33E−03	6.61E−02
	$\lVert\cdot\rVert_2$	0	1.00E+00	1.40E−03	3.68E−02
	$\lVert\cdot\rVert_\infty$	0	5.94E−01	6.73E−04	1.00E−02

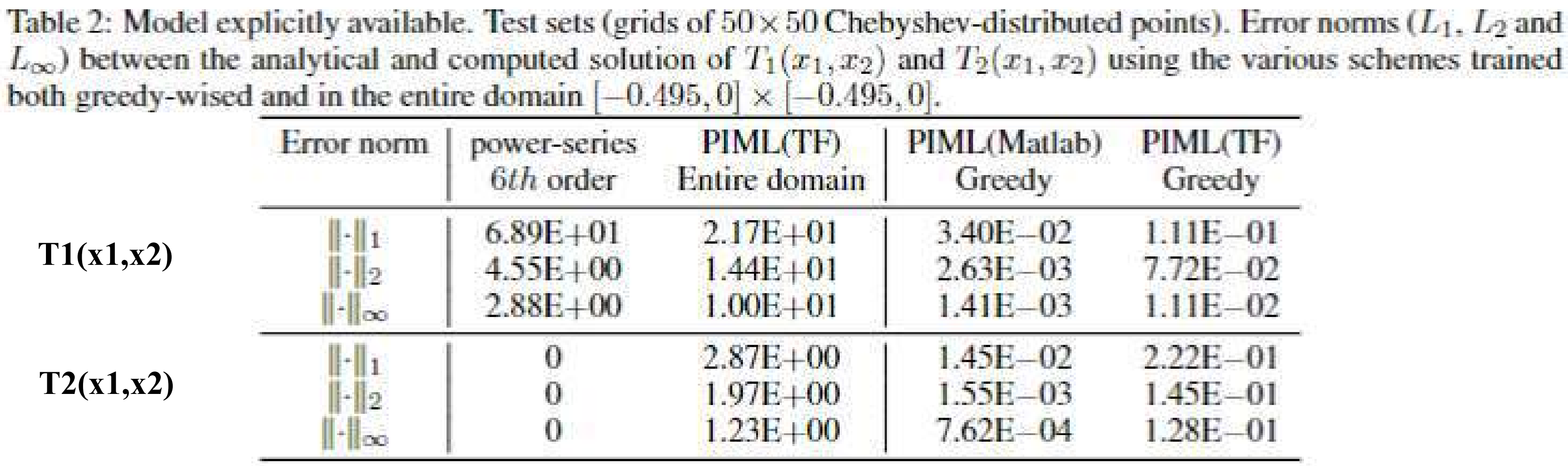


Table 2: Model explicitly available. Test sets (grids of 50×50 Chebyshev-distributed points). Error norms (L_1, L_2 and L_∞) between the analytical and computed solution of $T_1(x_1, x_2)$ and $T_2(x_1, x_2)$ using the various schemes trained both greedy-wised and in the entire domain $[-0.495, 0] \times [-0.495, 0]$.

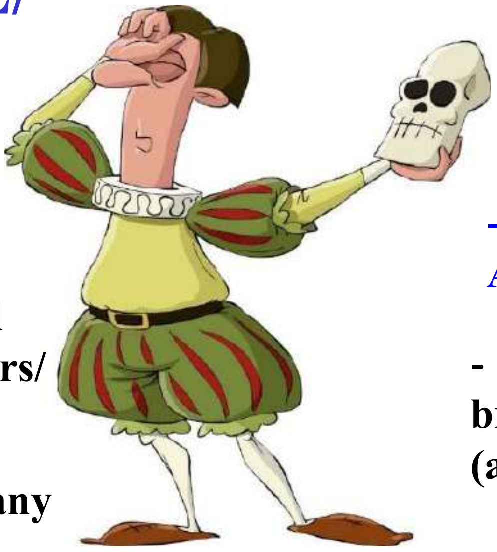
	Error norm	power-series 6th order	PIML(TF) Entire domain	PIML(Matlab) Greedy	PIML(TF) Greedy
T1(x1,x2)	$\lVert\cdot\rVert_1$	6.89E+01	2.17E+01	3.40E−02	1.11E−01
	$\lVert\cdot\rVert_2$	4.55E+00	1.44E+01	2.63E−03	7.72E−02
	$\lVert\cdot\rVert_\infty$	2.88E+00	1.00E+01	1.41E−03	1.11E−02
T2(x1,x2)	$\lVert\cdot\rVert_1$	0	2.87E+00	1.45E−02	2.22E−01
	$\lVert\cdot\rVert_2$	0	1.97E+00	1.55E−03	1.45E−01
	$\lVert\cdot\rVert_\infty$	0	1.23E+00	7.62E−04	1.28E−01

That is the question

...to Learn Or Not to Learn?

Physics-Informed ML/ Deep Learning

- When we have only Data/ No high-fidelity simulators
- When we want to get some physical insight on the Differential Operators/ Closures
- When we want to run something many times "quick and dirty"
- When we have some physical insight (variables, a generic PDE) for the emergent Dynamics and want to refine it through high-fidelity simulations



Equation-Free Numerical Analysis Toolkit

- When we have Agent-Based (ABs) simulators
- When we want to do numerical bifurcation analysis (also accurately identify tipping points)
- When we want to design controllers for Abs in a data-driven way
- Name it!

Numerical Analysis of Dynamical Systems

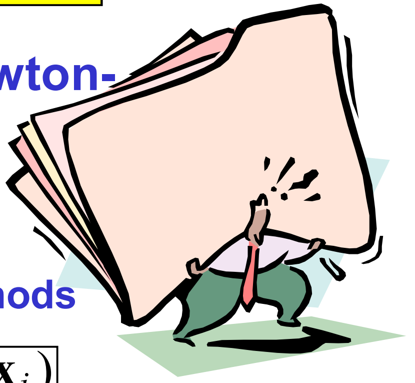
- Have Equations (ODEs/ PDE's) $\frac{\partial u(x,t)}{\partial t} = L(u(x,t),p)$

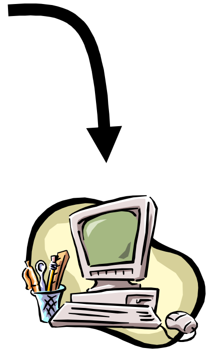
- Can apply terrific numerical analysis and control methods
- (e.g. finite elements, bifurcation analysis, feedback linearization)

$$\begin{tikzcd} u(x,t) \arrow[r] & \boxed{\varphi_\tau} \arrow[r] & u(x,t+\tau) \end{tikzcd}$$

Maps/Large-Scale Systems Analysis

Jacobian too big to apply direct solvers (e.g. Newton-Raphson, Jacobi)





- Instead, use well-established iterative linear algebra methods (matrix-free methods).

$$\boxed{\underline{\mathbf{J}}(\underline{\mathbf{x}}_i)\cdot\underline{\mathbf{e}} = \lim_{\varepsilon \to 0} \frac{\underline{\mathbf{f}}(\underline{\mathbf{x}}_i + \varepsilon\underline{\mathbf{e}}) - \underline{\mathbf{f}}(\underline{\mathbf{x}}_i)}{\varepsilon}}$$

-These are based on input-output data and can be used for finding steady states (even unstable ones!), critical eigenvalues and eigenvectors

(Kelley, Iterative methods for linear & nonlinear Equations, 1995)

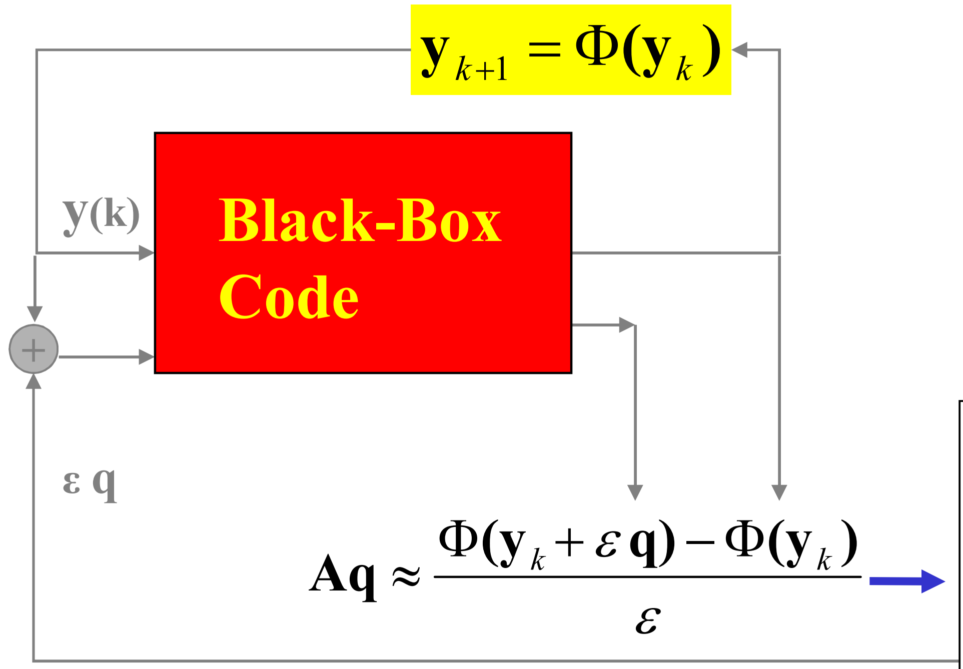
- As example, the Arnoldi eigensolver in the next slide…..

Methods in the Krylov Subspace: Arnoldi Eigensolver & GMRES

No need of "transparent" equations, just a black box code that will integrate it over a step size that it has been chosen.

GMRES $\longrightarrow$ SOLVE AX=B

SYSTEM AROUND THE STEADY STATE

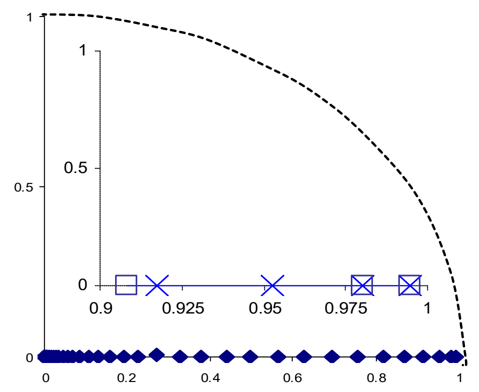


$$\mathbf{Aq} \approx \frac{\Phi(\mathbf{y}_k + \varepsilon \mathbf{q}) - \Phi(\mathbf{y}_k)}{\varepsilon} \longrightarrow$$

Critical Eigenvalues

$\uparrow$

ARNOLDI



```
Set $q_1$ with $\|q_1\| = 1$
    For j =1,m
    (1) Calculation    $Aq_j$
    (2) Calculation    $h_{t,j} = \langle Aq_j, q_t \rangle, t = 1,2,\ldots,j$
    (3) $r_j = Aq_j - \sum_{t=1}^{j} h_{t,j} q_t$
    (4) $h_{j+1,j} = \langle r_j, r_j \rangle^{1/2}$
    (5) $q_{j+1} = r_j / h_{j+1,j}$
    End For
```

In step m the algorithm creates an orthogonal basis in Krylov subspace K_m

$Q_m = \{q_1, Aq_1, \ldots, A^{m-1} q_{m-1}\}$

The projection of A in K_m results to an upper Hessenberg $H_m = Q_m^T A Q_m$ with elements h_{ij}

What if we have an Agent-based simulator (High-Fidelity simulator) but not a ROM (PDEs or ODEs) in a closed form?

$$\cancel{x_{k+1} = F_T(x_k, p)}$$

Thus, **let as assume, that we have an agent-based** dynamical model that, given a atomistic/individual-based/ detailed distribution of states

$$U_k \equiv U(t_k) \in R^N, N >> 1$$

at time $t_k = kT_U$ will report the values of the evolved /detailed

distribution after a time interval T_U

$$U_{k+1} = C_{T_U}(U_k, p).$$

$C_{T_U} : R^N \times R^m \to R^N$ is the time-evolution agent-based operator

The assumption for building ROMs

$$U_{k+1} = C_{T_U}(U_k, p). \quad C_{T_U} : R^N \times R^m \rightarrow R^N \quad N>>1$$

A basic assumption that we will make is that after some time $t > T_U$

The **emergent coarse-grained dynamics** are governed by a few variables, say

$$x \in R^n, n << N$$

Usually, these «few» observables **are the first few moments of the underlying microscopic distribution**.

This implies that there **is a slow coarse-grained manifold** that can be parametrized by *x*

The assumption for building ROMs

Fenichels'theorem

For the existence of a coarse-grained slow manifold

$$U_{k+1} = C_{T_U}(U_k, p) \qquad C_{T_U} : R^N \times R^m \to R^N$$

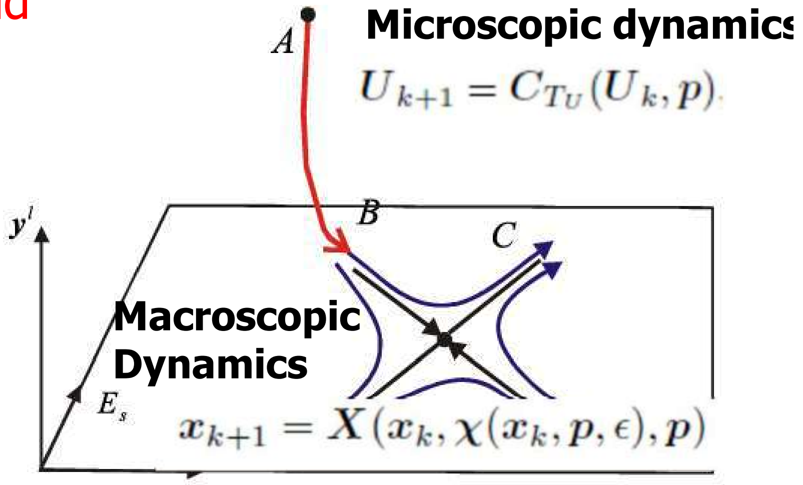
This implies that exists a slow coarse-grained manifold:

$$\begin{aligned} x_{k+1} &= X(x_k, y_k, p, \epsilon) \\ \epsilon y_{k+1} &= Y(x_k, y_k, p, \epsilon) \end{aligned}$$

By Fenichel's theorem: $x_{k+1} = X(x_k, \chi(x_k, p, \epsilon), p)$

on a smooth manifold defined by

$$M_\epsilon = \{(x, y) \in R^n \times R^{N-n} : y = \chi(x, p, \epsilon)\}$$



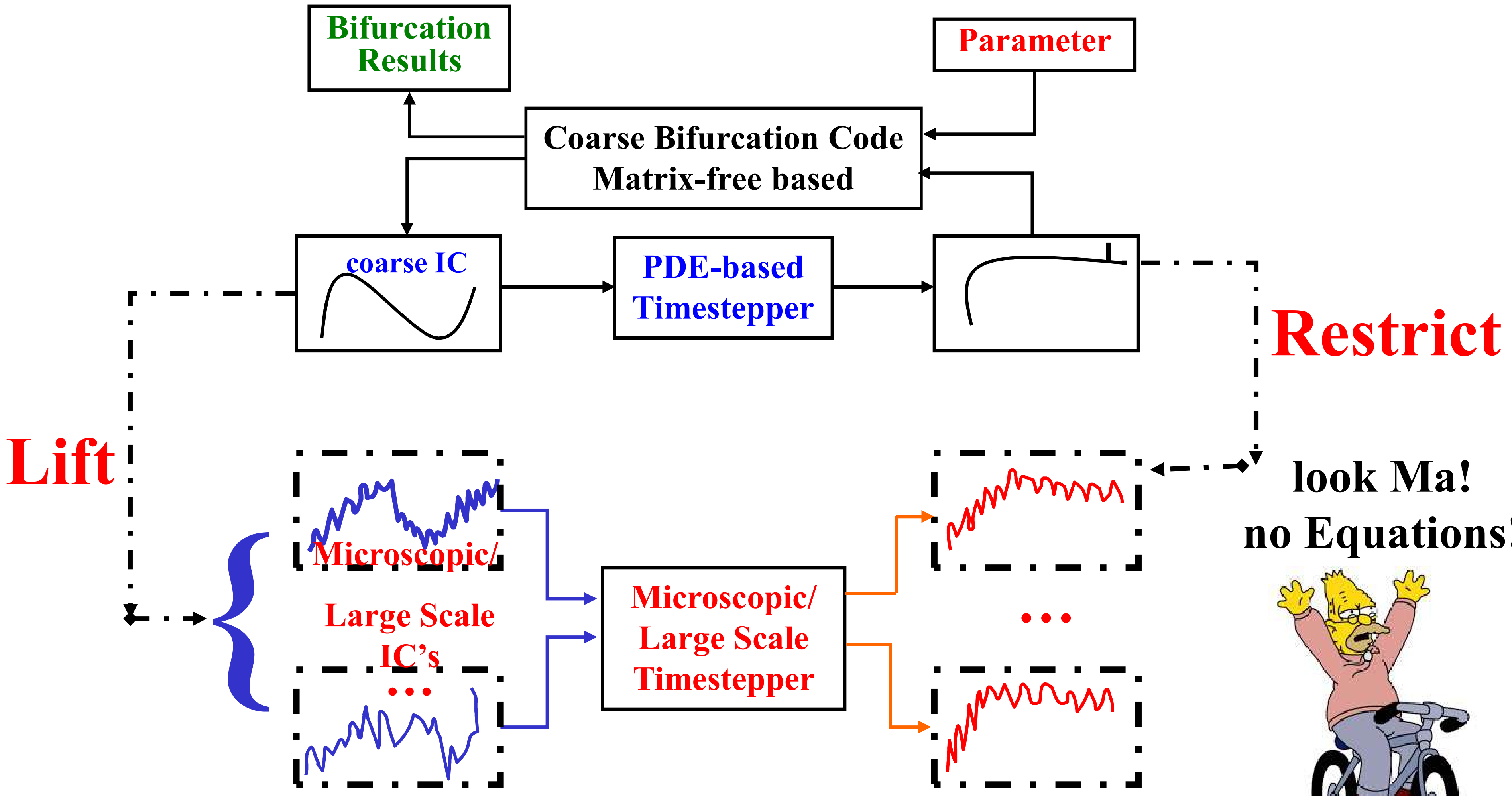
A
Microscopic dynamics
$U_{k+1} = C_{T_U}(U_k, p)$
y'
B
C
Macroscopic Dynamics
E_s
$x_{k+1} = X(x_k, \chi(x_k, p, \epsilon), p)$

Trajectories in the neighborhood of a coarse-grained saddle point. Starting from a detailed distribution (point A) the system will, under the assumptions, relax on the slow coarse-grained manifold, E_s (AB orbit). Then the coarse-grained dynamics will evolve towards the coarse-grained unstable manifold (BC orbit)

Bridging micro to macro: The Equation-Free Approach

…. or else …. How to find fixed points without the equations

(Kevrekidis et al., *Comm. Math. Sciences*, 1, 715-762, 2003)

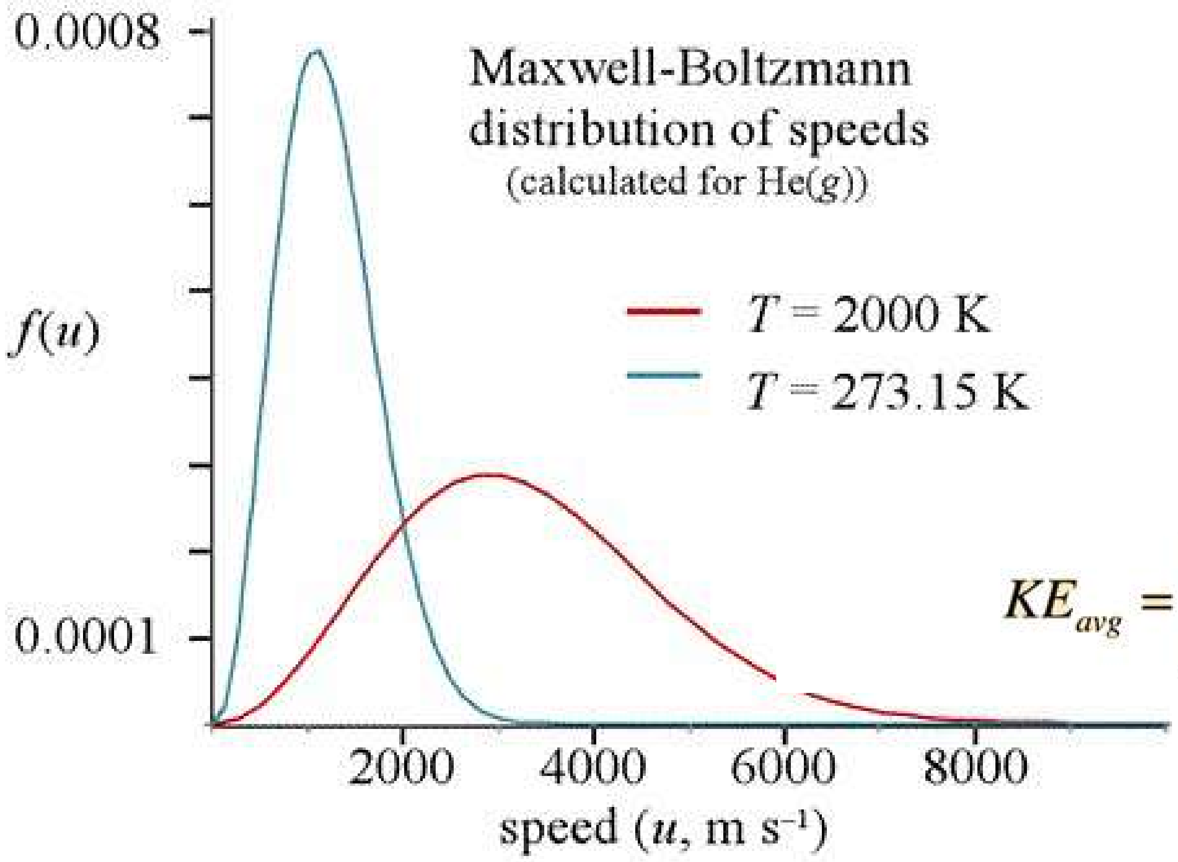


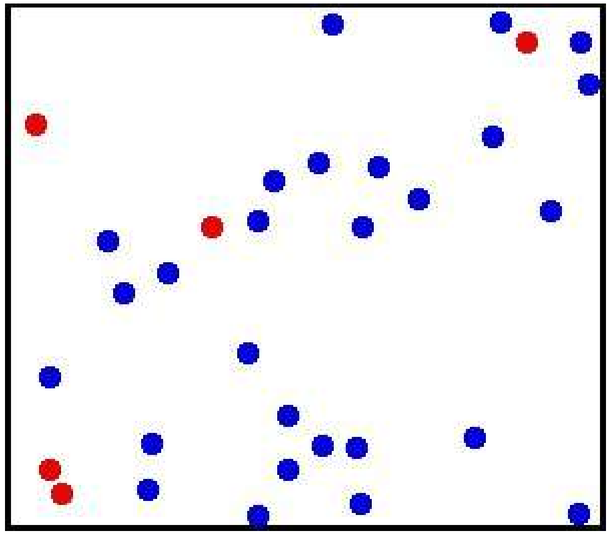
Bifurcation Results
Parameter
Coarse Bifurcation Code Matrix-free based
coarse IC
PDE-based Timestepper
Restrict
Lift
Microscopic/ Large Scale IC's
…
Microscopic/ Large Scale Timestepper
…

look Ma!
no Equations!

The Equation-Free Approach: Restrict and Lift

From distributions to continuum level variables and back

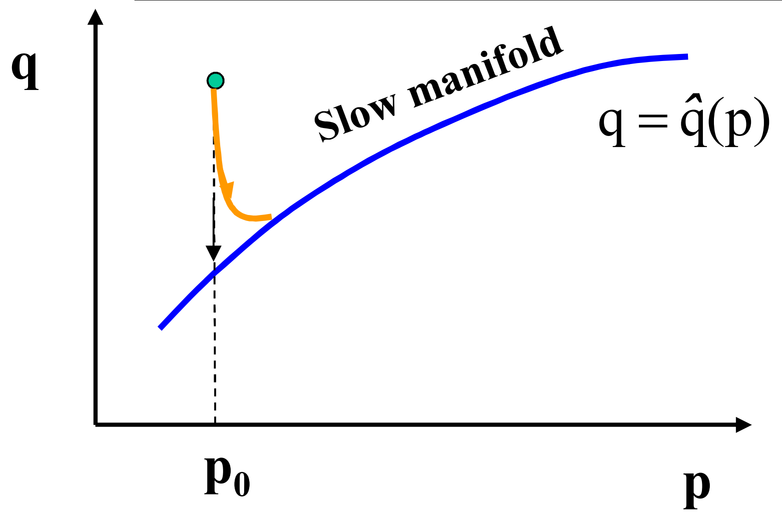




$$KE_{avg} = \left[\overline{\frac{1}{2}mv^2}\right] = \frac{3}{2}kT$$

probability density function for speed $$f(u) = \left[\frac{M}{2\pi RT}\right]^{3/2} \cdot 4\pi u^2 \cdot e^{-Mu^2/2RT}$$

SLOW & FAST DYNAMICS

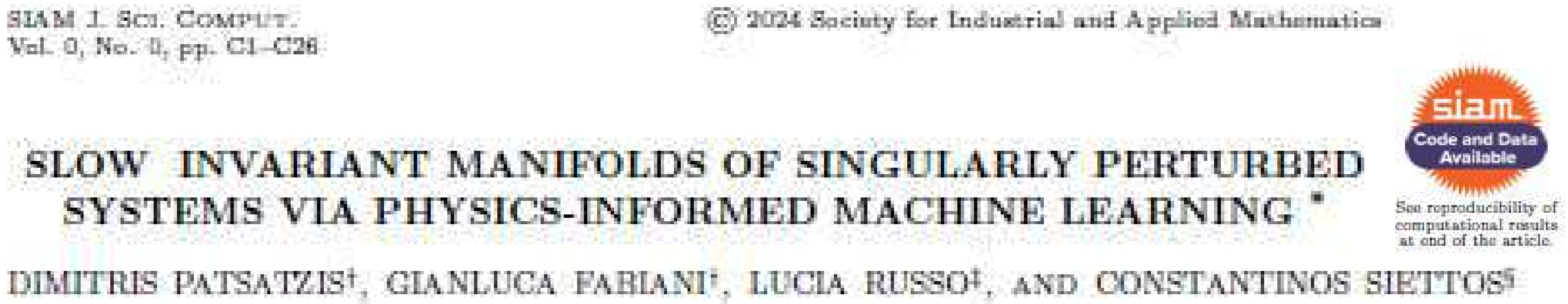


$$\frac{dp}{dt}=f(p,q)$$

$$\frac{dq}{dt}=\frac{1}{\varepsilon}g(p,q)$$

ε small ➔ $\left.\begin{array}{l} g(p,q)\approx 0 \\ q\approx\hat{q}(p) \end{array}\right\}$ **very fast**

So soon $\dot{p}=\boldsymbol{f}(p,q)\approx\boldsymbol{f}(p,\hat{q}(p))\equiv\tilde{\boldsymbol{f}}(p)$ **Singular perturbed systems**


SIAM J. SCI. COMPUT.
Vol. 0, No. 0, pp. C1–C26

© 2024 Society for Industrial and Applied Mathematics

SLOW INVARIANT MANIFOLDS OF SINGULARLY PERTURBED SYSTEMS VIA PHYSICS-INFORMED MACHINE LEARNING *

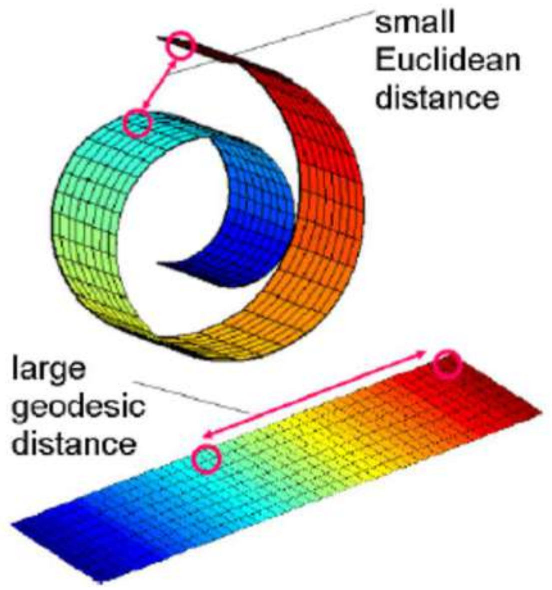
siam Code and Data Available

See reproducibility of computational results at end of the article.

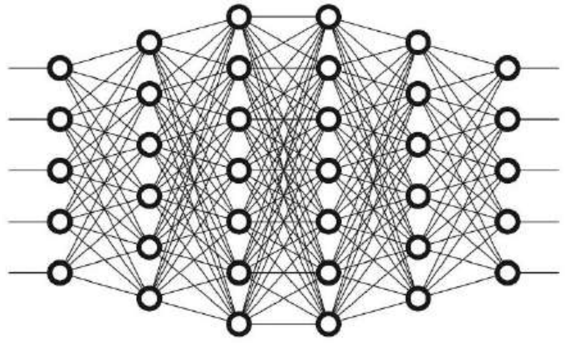
DIMITRIS PATSATZIS[†], GIANLUCA FABIANI[†], LUCIA RUSSO[‡], AND CONSTANTINOS SIETTOS[§]

Next Generation Scientific Machine Learning

Bridge niche Numerical Analysis (Equation-Free) and SciML (PINNs, DeepONet) to create Emergent Latent Spaces in which the New Physics will be learned and and solved!



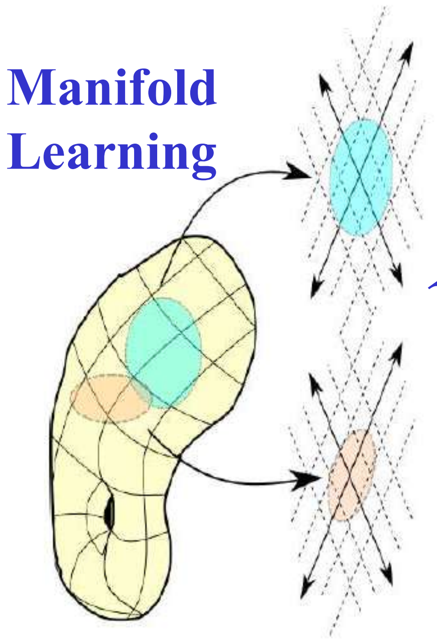
DeepONet

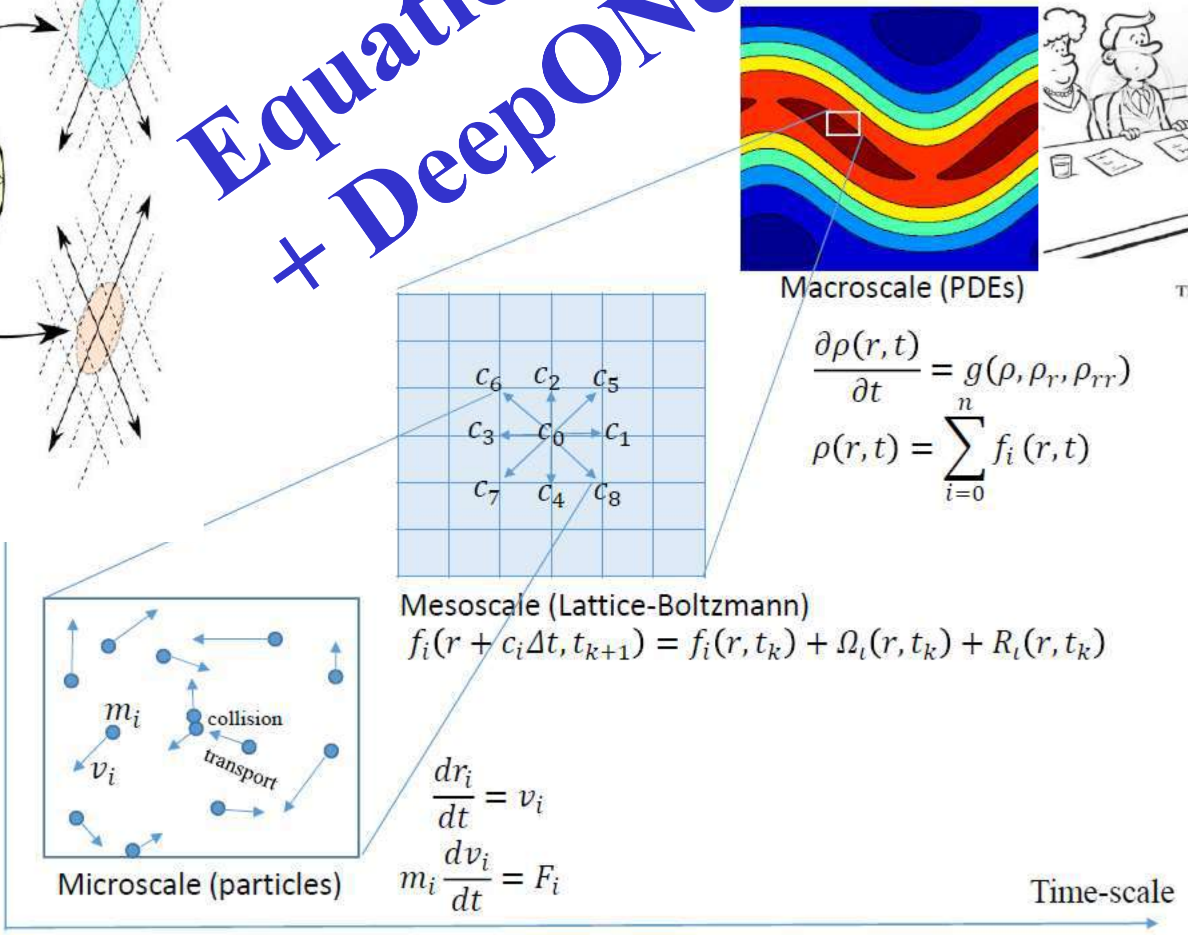


Numerical Analysis with No Equations, No Variables.

Next Generation Scientific Machine Learning

Equation-Free + DeepONets




Macroscale (PDEs)
Mesoscale (Lattice-Boltzmann)
c_6 c_2 c_5 c_3 c_0 c_1 c_7 c_4 c_8
Microscale (particles)
m_i
v_i
collision
transport
Time-scale

$$\frac{\partial \rho(r,t)}{\partial t} = g(\rho, \rho_r, \rho_{rr})$$

$$\rho(r,t) = \sum_{i=0}^{n} f_i(r,t)$$

$$f_i(r + c_i\Delta t, t_{k+1}) = f_i(r,t_k) + \Omega_i(r,t_k) + R_i(r,t_k)$$

$$\frac{dr_i}{dt} = v_i$$

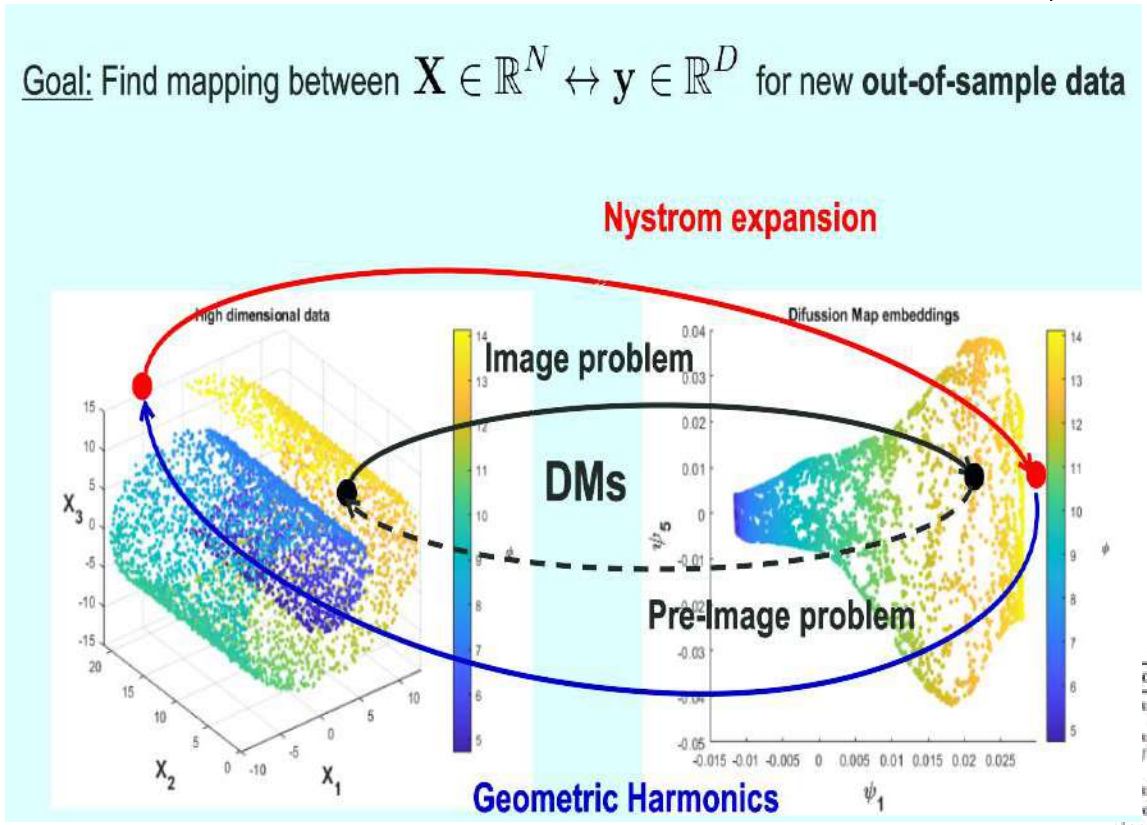
$$m_i \frac{dv_i}{dt} = F_i$$

This is Mr Smith from Big Data Mining.
He says he's found an insight.

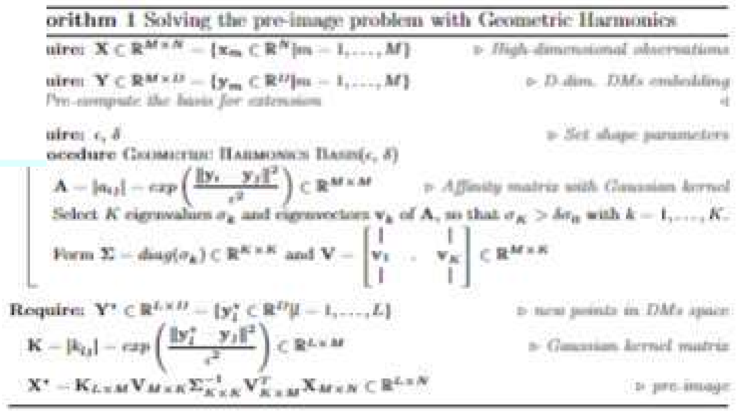
Data-Driven Encoding- Decoding

Encoder (Nystrom Extension to DMs) coder (Geometric Harmonics)

Coifman, R.R. and Lafon, S., 2006. Geometric harmonics: a novel tool for multiscale out-of-sample extension of empirical functions. *Applied and Computational Harmonic Analysis*, *21*(1), pp.31-52.



Goal: Find mapping between $\mathbf{X} \in \mathbb{R}^N \leftrightarrow \mathbf{y} \in \mathbb{R}^D$ for new **out-of-sample data**



```
orithm 1 Solving the pre-image problem with Geometric Harmonics
uire: $\mathbf{X} \subset \mathbb{R}^{M\times N} = \{\mathbf{x}_m \subset \mathbb{R}^N | m = 1,\dots,M\}$    ▷ High-dimensional observations
uire: $\mathbf{Y} \subset \mathbb{R}^{M\times D} = \{\mathbf{y}_m \subset \mathbb{R}^D | m = 1,\dots,M\}$    ▷ D-dim. DMs embedding
Pre-compute the basis for extension    ◁
uire: $\epsilon$, $\delta$    ▷ Set shape parameters
ocedure GEOMETRIC HARMONICS BASIS($\epsilon$, $\delta$)
    $\mathbf{A} = [a_{ij}] = exp\left(\frac{\|\mathbf{y}_i - \mathbf{y}_j\|^2}{\epsilon^2}\right) \subset \mathbb{R}^{M\times M}$    ▷ Affinity matrix with Gaussian kernel
    Select $K$ eigenvalues $\sigma_k$ and eigenvectors $\mathbf{v}_k$ of $\mathbf{A}$, so that $\sigma_K > \delta\sigma_0$ with $k = 1,\dots,K$.
    Form $\Sigma = diag(\sigma_k) \subset \mathbb{R}^{K\times K}$ and $\mathbf{V} = \begin{bmatrix} | & & | \\ \mathbf{v}_1 & \dots & \mathbf{v}_K \\ | & & | \end{bmatrix} \subset \mathbb{R}^{M\times K}$
Require: $\mathbf{Y}^* \subset \mathbb{R}^{L\times D} = \{\mathbf{y}^*_l \subset \mathbb{R}^D | l = 1,\dots,L\}$    ▷ new points in DMs space
  $\mathbf{K} = [k_{lj}] = exp\left(\frac{\|\mathbf{y}^*_l - \mathbf{y}_j\|^2}{\epsilon^2}\right) \subset \mathbb{R}^{L\times M}$    ▷ Gaussian kernel matrix
  $\mathbf{X}^* = \mathbf{K}_{L\times M}\mathbf{V}_{M\times K}\Sigma^{-1}_{K\times K}\mathbf{V}^T_{K\times M}\mathbf{X}_{M\times N} \subset \mathbb{R}^{L\times N}$    ▷ pre-image
```

Next Generation Equation-Free

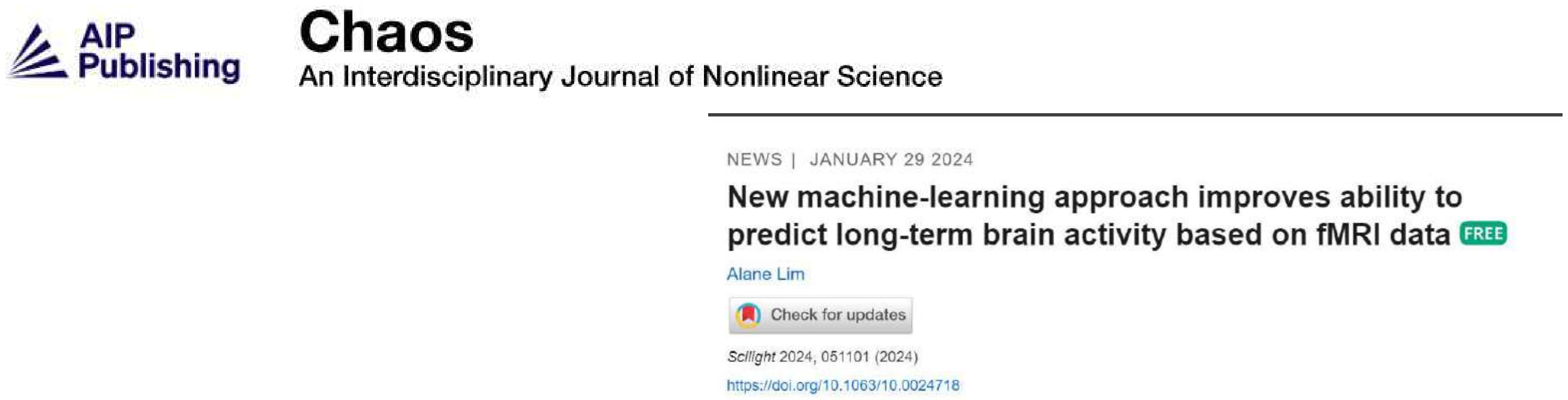
Journal of Computational Physics

Volume 478, 1 April 2023, 111953

Data-driven control of agent-based models: An Equation/Variable-free machine learning approach

Dimitrios G. Patsatzis [a], Lucia Russo [a], Ioannis G. Kevrekidis [b], Constantinos Siettos [c]

FEATURED ARTICLE


AIP Publishing

Chaos

An Interdisciplinary Journal of Nonlinear Science

NEWS | JANUARY 29 2024

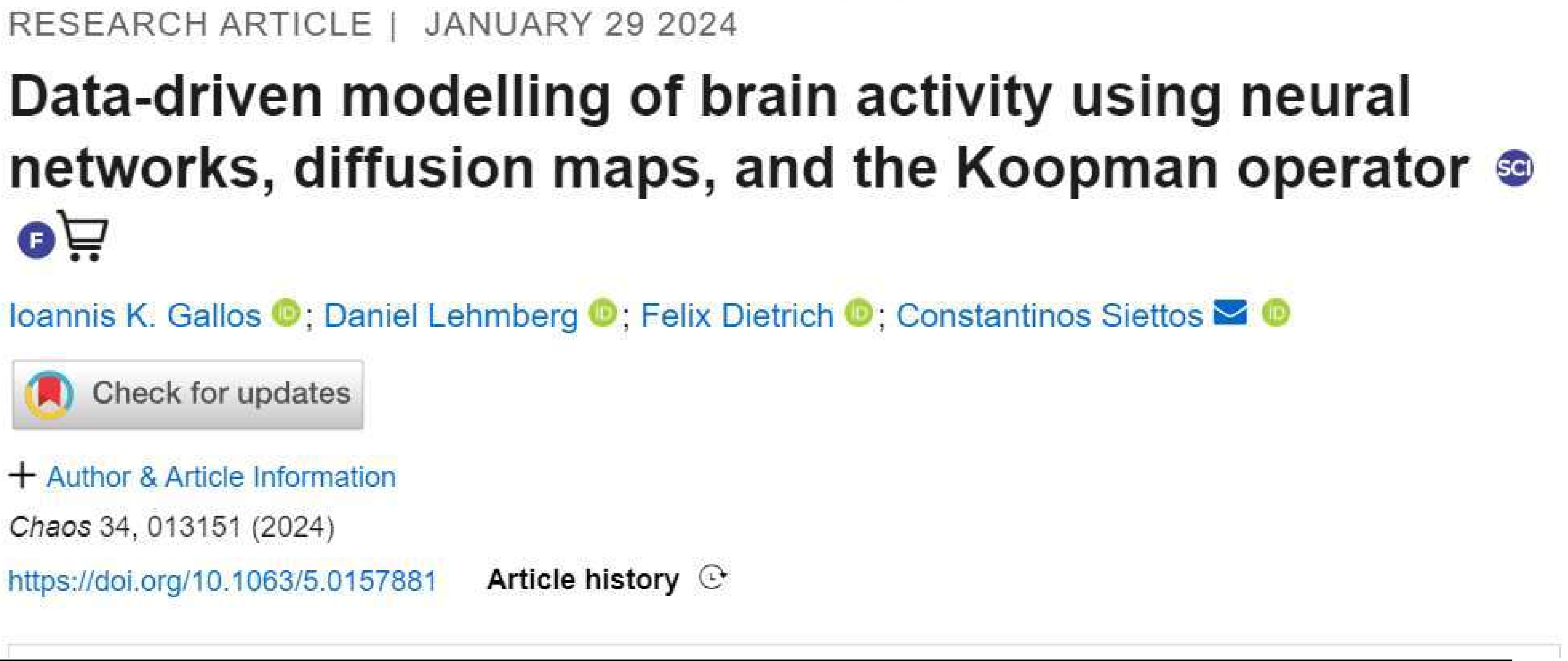
New machine-learning approach improves ability to predict long-term brain activity based on fMRI data FREE

Alane Lim

Check for updates

Scilight 2024, 051101 (2024)

https://doi.org/10.1063/10.0024718


RESEARCH ARTICLE | JANUARY 29 2024

Data-driven modelling of brain activity using neural networks, diffusion maps, and the Koopman operator

Ioannis K. Gallos; Daniel Lehmberg; Felix Dietrich; Constantinos Siettos

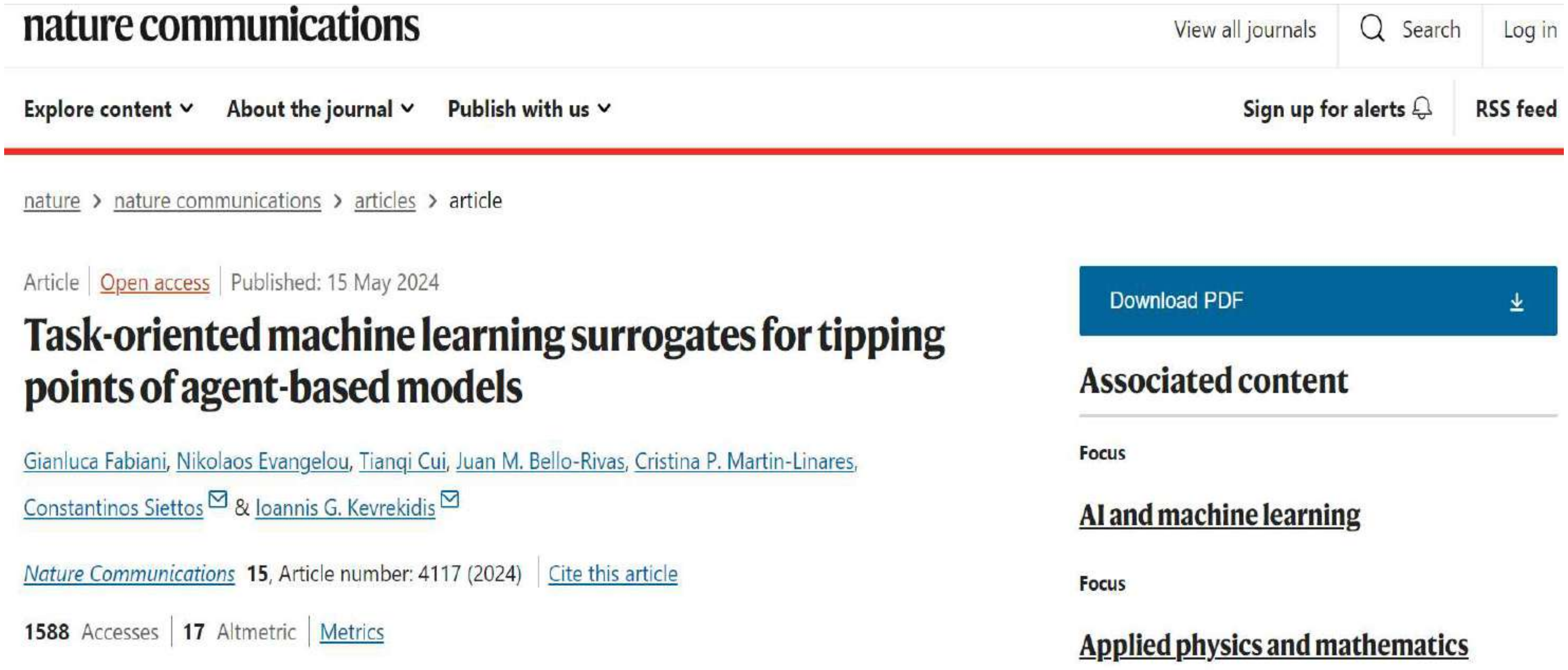
Check for updates

+ Author & Article Information

Chaos 34, 013151 (2024)

https://doi.org/10.1063/5.0157881 **Article history**

Featured articles


nature communications

View all journals | Search | Log in

Explore content | About the journal | Publish with us | Sign up for alerts | RSS feed

nature > nature communications > articles > article

Article | Open access | Published: 15 May 2024

Task-oriented machine learning surrogates for tipping points of agent-based models

Gianluca Fabiani, Nikolaos Evangelou, Tianqi Cui, Juan M. Bello-Rivas, Cristina P. Martin-Linares, Constantinos Siettos & Ioannis G. Kevrekidis

Nature Communications **15**, Article number: 4117 (2024) | Cite this article

1588 Accesses | **17** Altmetric | Metrics

Download PDF

Associated content

Focus

AI and machine learning

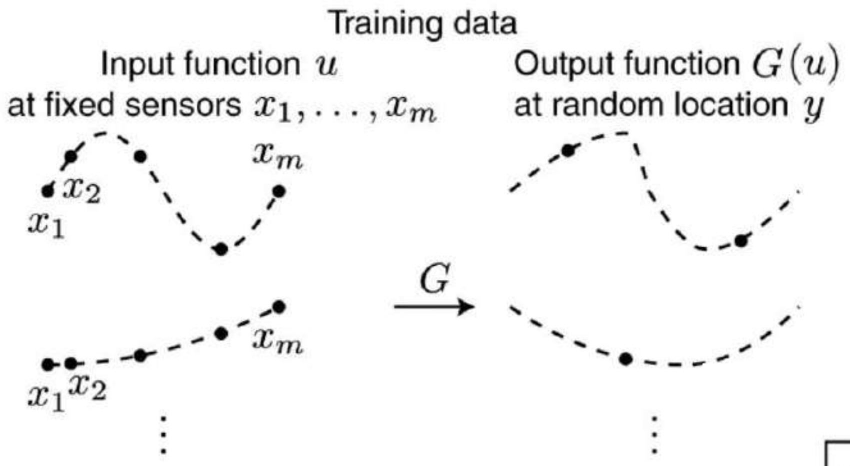
Focus

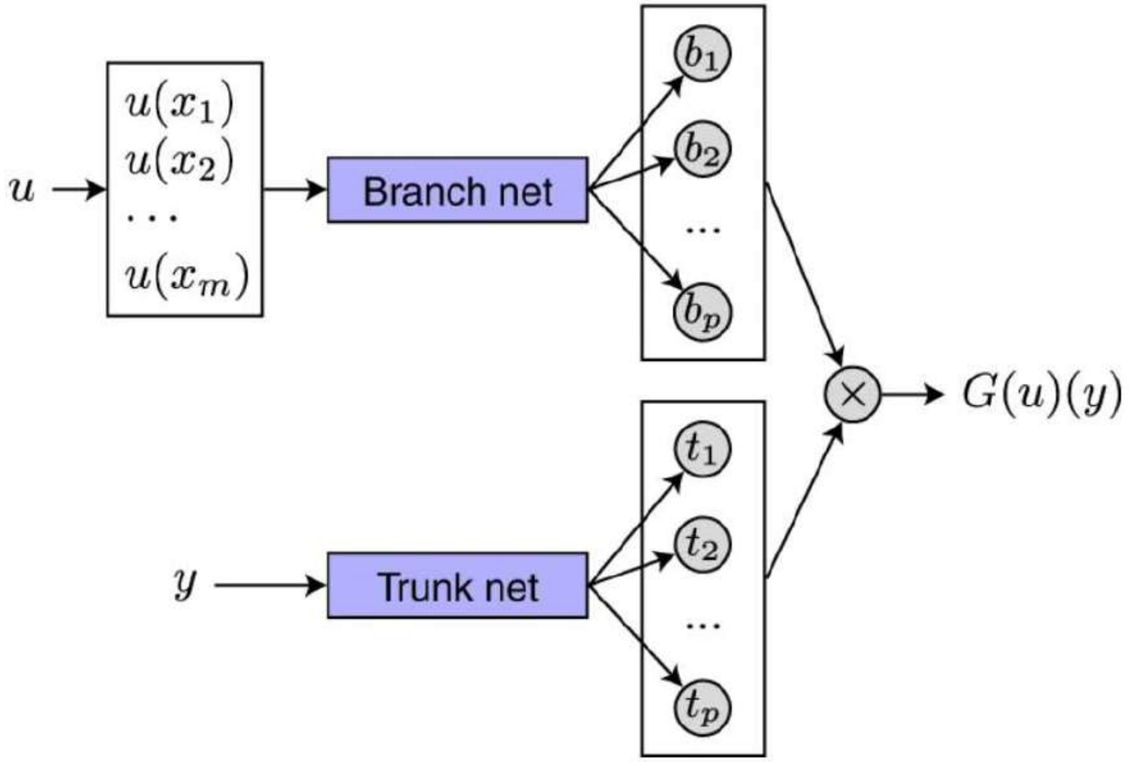
Applied physics and mathematics

DISCOVERING EVOLUTION AND SOLUTION OPERATORS FROM DATA

The DEEP-O-NET

Lu, L., Jin, P., Pang, G. *et al.* Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nat Mach Intell* **3**, 218–229 (2021).





DISCOVERING EVOLUTION AND SOLUTION OPERATORS FROM DATA

Chen, T. & Chen, H. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Trans. Neural Networks* **6**, 911–917 (1995).

Theorem 1 (Universal Approximation Theorem for Operator). *Suppose that σ is a continuous non-polynomial function, X is a Banach space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator, which maps V into $C(K_2)$. Then for any $\epsilon > 0$, there are positive integers n, p and m, constants c_i^k, ξ_{ij}^k, θ_i^k, $\zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \ldots, n$, $k = 1, \ldots, p$ and $j = 1, \ldots, m$, such that*

$$\left| G(u)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{\text{branch}} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{\text{trunk}} \right| < \epsilon \qquad (1)$$

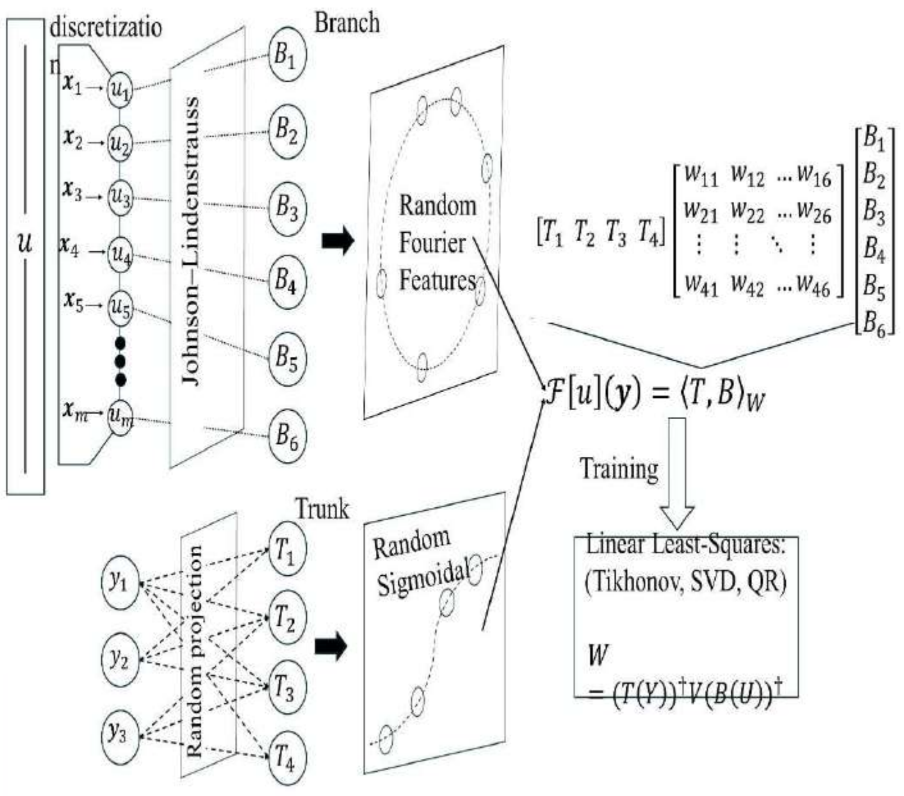
holds for all $u \in V$ and $y \in K_2$. Here, $C(K)$ is the Banach space of all continuous functions defined on K with norm $\| f \|_{C(K)} = \max_{x \in K} |f(x)|$.

RandONet: Shallow-Networks with Random Projections for learning linear and nonlinear operators https://arxiv.org/abs/2406.05470

DeepOnets are powerful but expensive:
- High-dimensional parameter space
- Significant computational cost
- Require parallel/GPU-based hardware
- Hence, they may result in moderate numerical approximation accuracy

RandOnets
We introduce Random projection-based Operator Networks to deal with the above-mentioned challenges.
We employ:
- Random projections
- Shallow Feedforward Neural Networks
- Established nice numerical analysis least-squares solvers for near-optimal accuracy and extremely fast training

Theoretical findings
Building on previous works, we prove the universal approximation property of RandOnets


discretization
Branch
u
x_1 u_1
x_2 u_2
x_3 u_3
x_4 u_4
x_5 u_5
x_m u_m
Johnson–Lindenstrauss
B_1 B_2 B_3 B_4 B_5 B_6
Random Fourier Features
$[T_1\ T_2\ T_3\ T_4]$
$\begin{bmatrix} w_{11} & w_{12} & \dots w_{16} \\ w_{21} & w_{22} & \dots w_{26} \\ \vdots & \vdots & \ddots & \vdots \\ w_{41} & w_{42} & \dots w_{46} \end{bmatrix}$
$\begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \end{bmatrix}$
$\mathcal{F}[u](y) = \langle T, B\rangle_W$
Training
Linear Least-Squares: (Tikhonov, SVD, QR)
$W = (T(Y))^{\dagger} V(B(U))^{\dagger}$
Trunk
y_1 y_2 y_3
Random projection
T_1 T_2 T_3 T_4
Random Sigmoidal

The proposed scheme
- embeds the space of the spatial locations (y)
- embeds the space of the discretized functions

onto randomly parametrized embeddings such as the Johnson-Lindenstrauss for linear operators and random fourier features for nonlinear operators

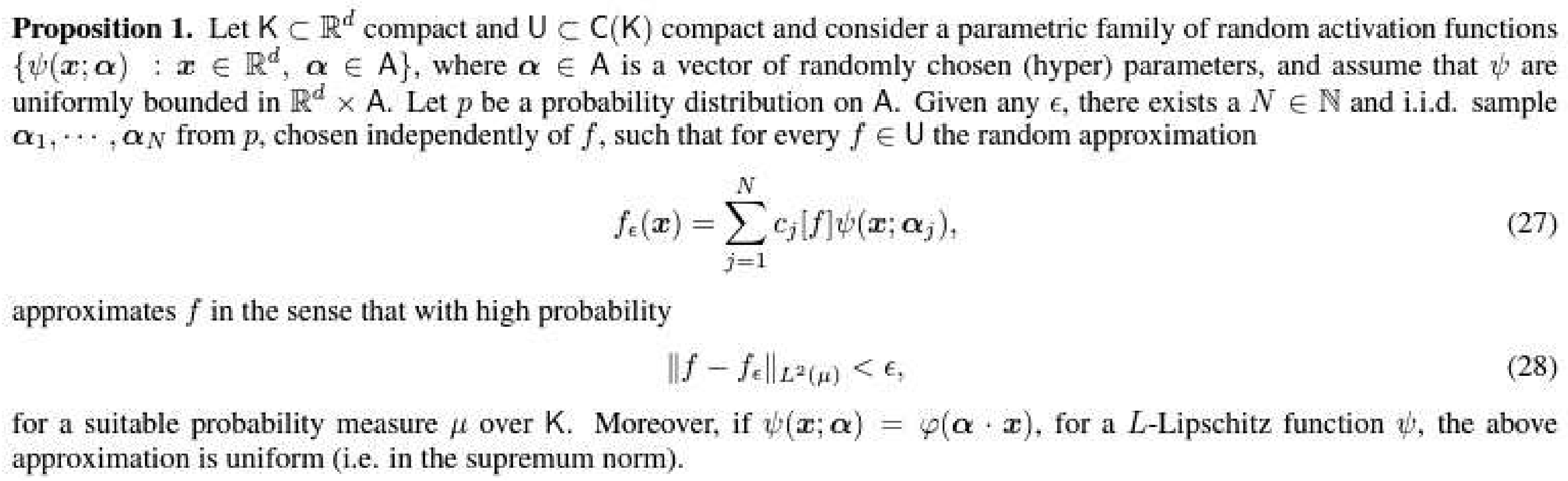
Numerical Results
We focus on the approximation of evolution operators (right-hand-sides) of PDEs.
First, we consider some pedagogical ODE examples
- antiderivative operators
- a gravity pendulum with external force

PDE evolution operators:
- Linear Diffusion-advection-reaction
- The Burgers' Equation
- The Allen-Cahn Phased Field PDE

Conclusion
RandONets outperform the vanilla DeepOnet in terms of numerical approximation accuracy and computational times by orders of magnitude

RandONet: Shallow-Networks with Random Projections for learning linear and nonlinear operators

https://arxiv.org/abs/2406.05470



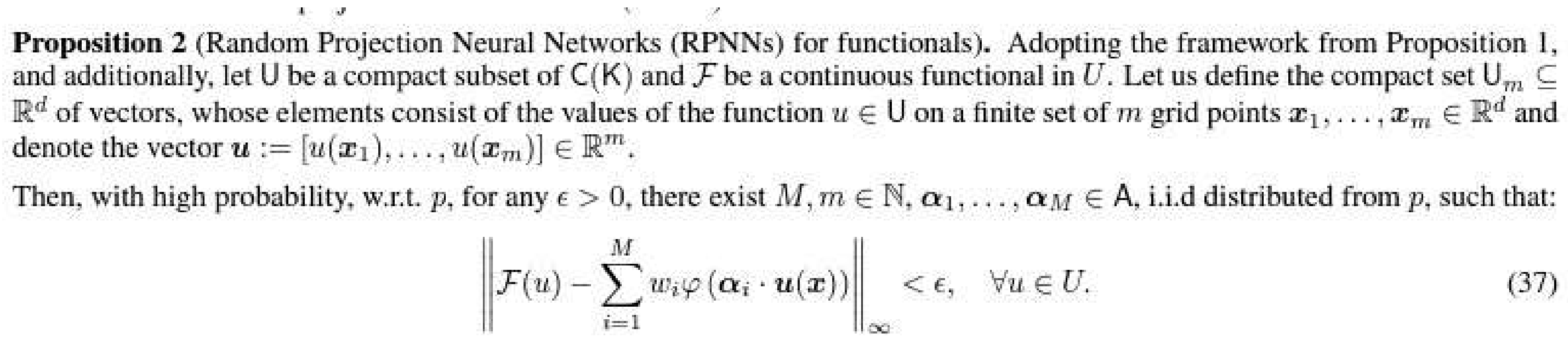
Proposition 1. Let $\mathsf{K} \subset \mathbb{R}^d$ compact and $\mathsf{U} \subset C(\mathsf{K})$ compact and consider a parametric family of random activation functions $\{\psi(\boldsymbol{x};\boldsymbol{\alpha}) \ : \boldsymbol{x} \in \mathbb{R}^d, \ \boldsymbol{\alpha} \in \mathsf{A}\}$, where $\boldsymbol{\alpha} \in \mathsf{A}$ is a vector of randomly chosen (hyper) parameters, and assume that ψ are uniformly bounded in $\mathbb{R}^d \times \mathsf{A}$. Let p be a probability distribution on A. Given any ϵ, there exists a $N \in \mathbb{N}$ and i.i.d. sample $\boldsymbol{\alpha}_1, \cdots, \boldsymbol{\alpha}_N$ from p, chosen independently of f, such that for every $f \in \mathsf{U}$ the random approximation

$$f_\epsilon(\boldsymbol{x}) = \sum_{j=1}^{N} c_j[f]\psi(\boldsymbol{x};\boldsymbol{\alpha}_j), \tag{27}$$

approximates f in the sense that with high probability

$$\|f - f_\epsilon\|_{L^2(\mu)} < \epsilon, \tag{28}$$

for a suitable probability measure μ over K. Moreover, if $\psi(\boldsymbol{x};\boldsymbol{\alpha}) = \varphi(\boldsymbol{\alpha} \cdot \boldsymbol{x})$, for a L-Lipschitz function ψ, the above approximation is uniform (i.e. in the supremum norm).



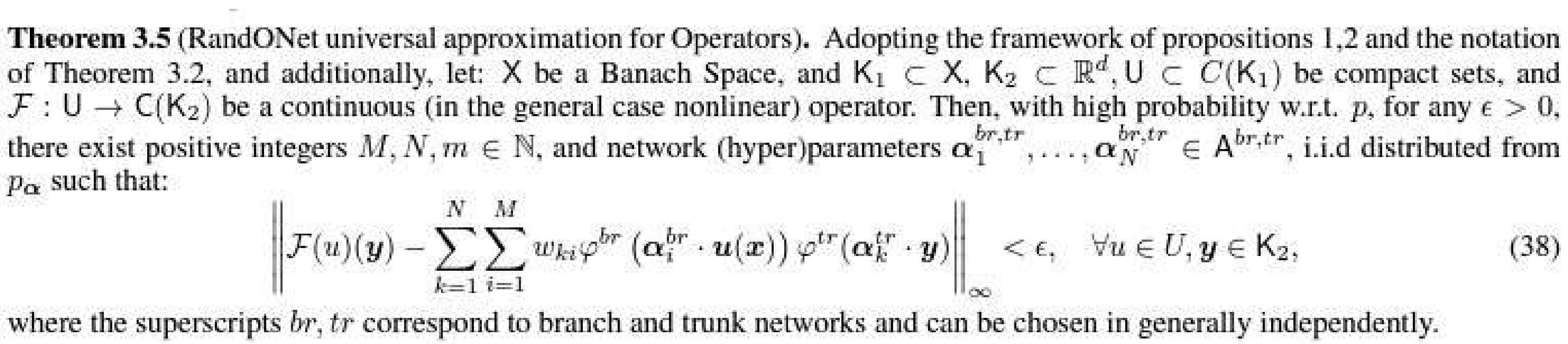
Proposition 2 (Random Projection Neural Networks (RPNNs) for functionals)**.** Adopting the framework from Proposition 1, and additionally, let U be a compact subset of $C(\mathsf{K})$ and $\mathcal{F}$ be a continuous functional in U. Let us define the compact set $\mathsf{U}_m \subseteq \mathbb{R}^d$ of vectors, whose elements consist of the values of the function $u \in \mathsf{U}$ on a finite set of m grid points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \in \mathbb{R}^d$ and denote the vector $\boldsymbol{u} := [u(\boldsymbol{x}_1), \ldots, u(\boldsymbol{x}_m)] \in \mathbb{R}^m$.

Then, with high probability, w.r.t. p, for any $\epsilon > 0$, there exist $M, m \in \mathbb{N}$, $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_M \in \mathsf{A}$, i.i.d distributed from p, such that:

$$\left\| \mathcal{F}(u) - \sum_{i=1}^{M} w_i \varphi\left(\boldsymbol{\alpha}_i \cdot \boldsymbol{u}(\boldsymbol{x})\right) \right\|_\infty < \epsilon, \quad \forall u \in U. \tag{37}$$

RandONet: Shallow-Networks with Random Projections for learning linear and nonlinear operators

https://arxiv.org/abs/2406.05470

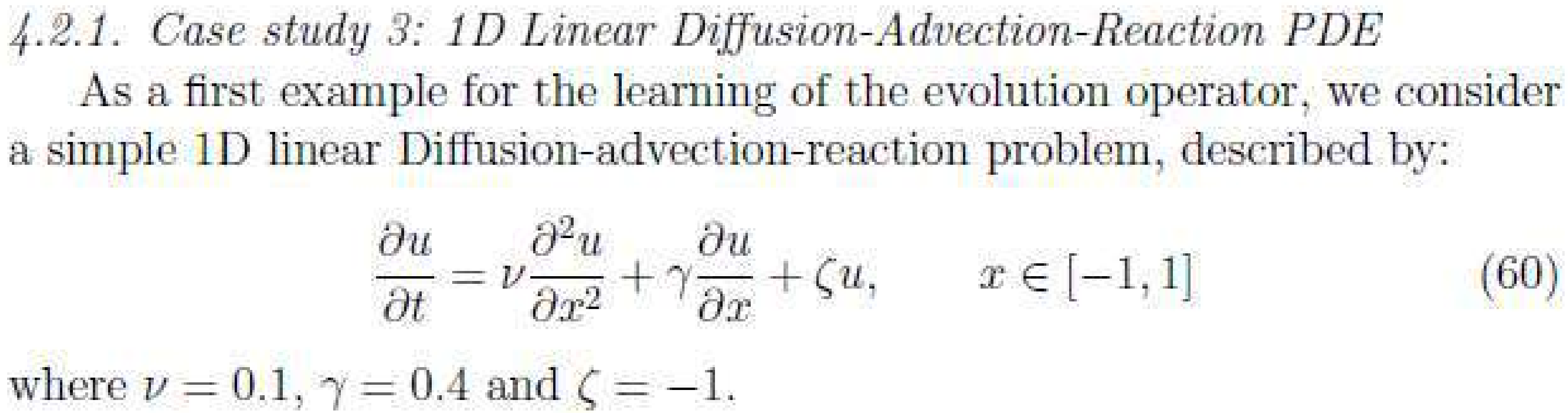


Theorem 3.5 (RandONet universal approximation for Operators). Adopting the framework of propositions 1,2 and the notation of Theorem 3.2, and additionally, let: X be a Banach Space, and $\mathsf{K}_1 \subset \mathsf{X}$, $\mathsf{K}_2 \subset \mathbb{R}^d, \mathsf{U} \subset C(\mathsf{K}_1)$ be compact sets, and $\mathcal{F} : \mathsf{U} \to \mathsf{C}(\mathsf{K}_2)$ be a continuous (in the general case nonlinear) operator. Then, with high probability w.r.t. p, for any $\epsilon > 0$, there exist positive integers $M, N, m \in \mathbb{N}$, and network (hyper)parameters $\boldsymbol{\alpha}_1^{br,tr}, \ldots, \boldsymbol{\alpha}_N^{br,tr} \in \mathsf{A}^{br,tr}$, i.i.d distributed from $p_{\boldsymbol{\alpha}}$ such that:

$$\left\| \mathcal{F}(u)(\boldsymbol{y}) - \sum_{k=1}^{N} \sum_{i=1}^{M} w_{ki} \varphi^{br} \left(\boldsymbol{\alpha}_i^{br} \cdot \boldsymbol{u}(\boldsymbol{x}) \right) \varphi^{tr} (\boldsymbol{\alpha}_k^{tr} \cdot \boldsymbol{y}) \right\|_{\infty} < \epsilon, \quad \forall u \in U, \boldsymbol{y} \in \mathsf{K}_2, \tag{38}$$

where the superscripts br, tr correspond to branch and trunk networks and can be chosen in generally independently.

RandONets (Embedded Shallow NNs vs DeepOnets (Deep Learning)

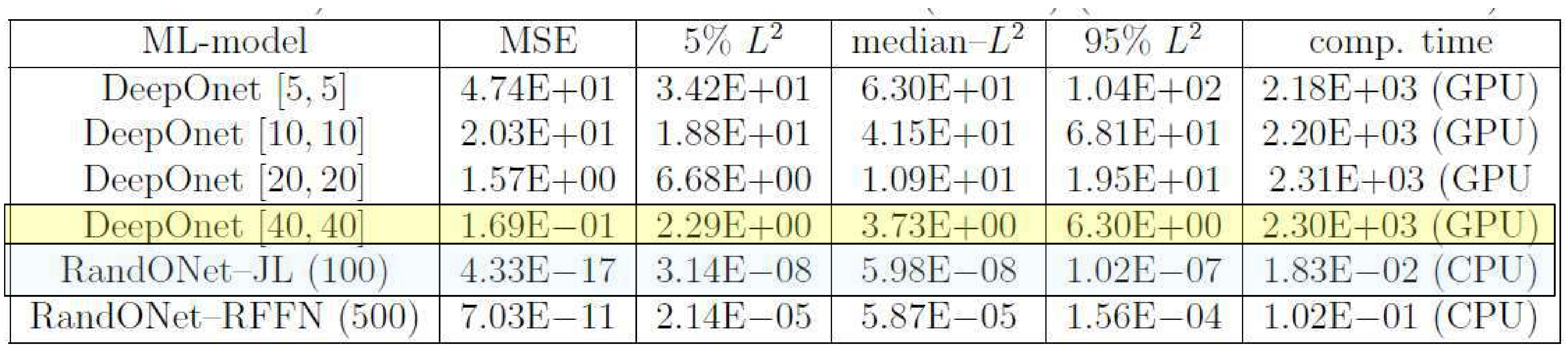


4.2.1. Case study 3: 1D Linear Diffusion-Advection-Reaction PDE

As a first example for the learning of the evolution operator, we consider a simple 1D linear Diffusion-advection-reaction problem, described by:

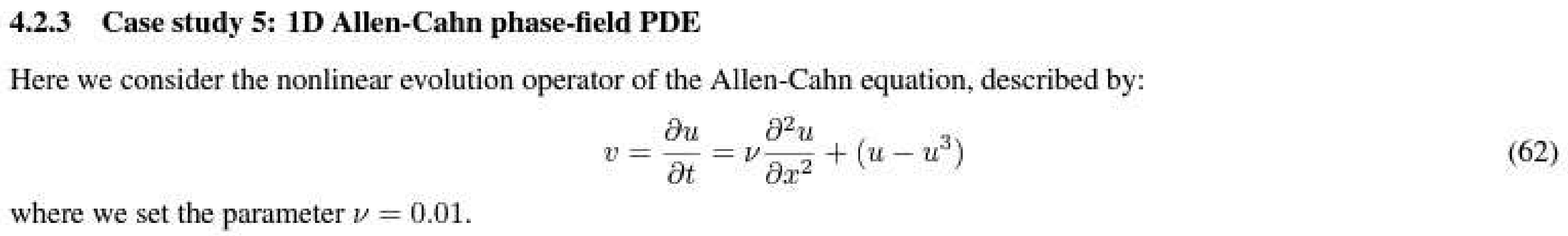
$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} + \gamma \frac{\partial u}{\partial x} + \zeta u, \qquad x \in [-1, 1] \tag{60}$$

where $\nu = 0.1$, $\gamma = 0.4$ and $\zeta = -1$.



ML-model	MSE	5% L^2	median–L^2	95% L^2	comp. time
DeepOnet $[5, 5]$	4.74E+01	3.42E+01	6.30E+01	1.04E+02	2.18E+03 (GPU)
DeepOnet $[10, 10]$	2.03E+01	1.88E+01	4.15E+01	6.81E+01	2.20E+03 (GPU)
DeepOnet $[20, 20]$	1.57E+00	6.68E+00	1.09E+01	1.95E+01	2.31E+03 (GPU
DeepOnet $[40, 40]$	1.69E−01	2.29E+00	3.73E+00	6.30E+00	2.30E+03 (GPU)
RandONet–JL (100)	4.33E−17	3.14E−08	5.98E−08	1.02E−07	1.83E−02 (CPU)
RandONet–RFFN (500)	7.03E−11	2.14E−05	5.87E−05	1.56E−04	1.02E−01 (CPU)

RandONets (Embedded Shallow NNs vs DeepOnets (Deep Learning)



4.2.3 Case study 5: 1D Allen-Cahn phase-field PDE

Here we consider the nonlinear evolution operator of the Allen-Cahn equation, described by:

$$v = \frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} + (u - u^3) \qquad (62)$$

where we set the parameter $\nu = 0.01$.

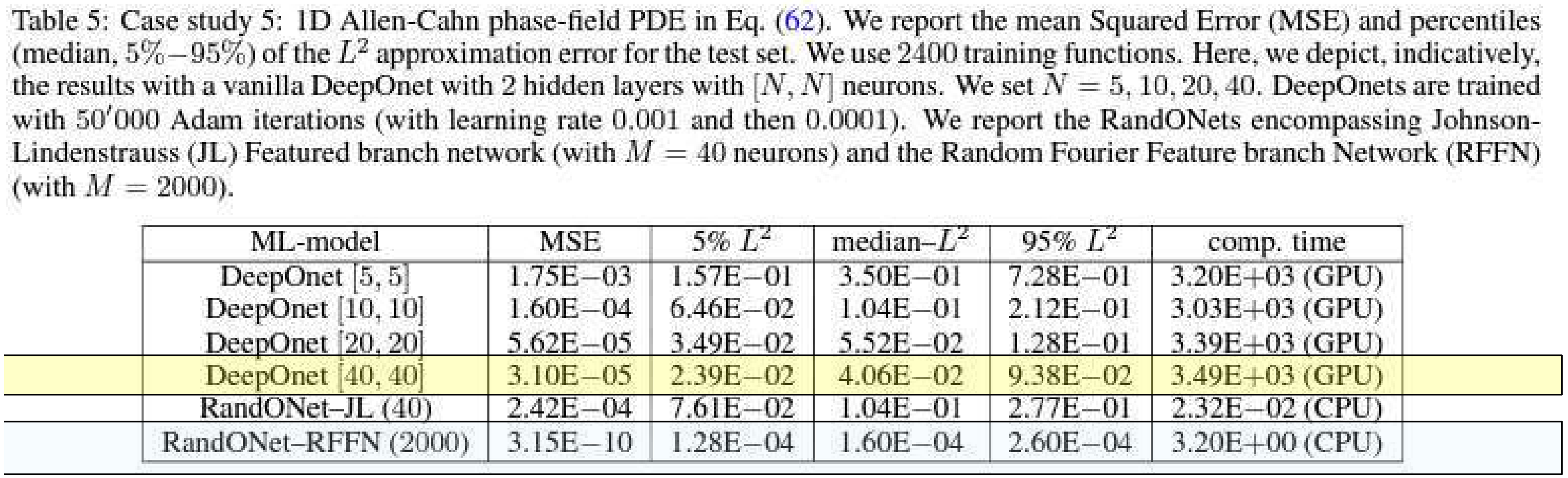
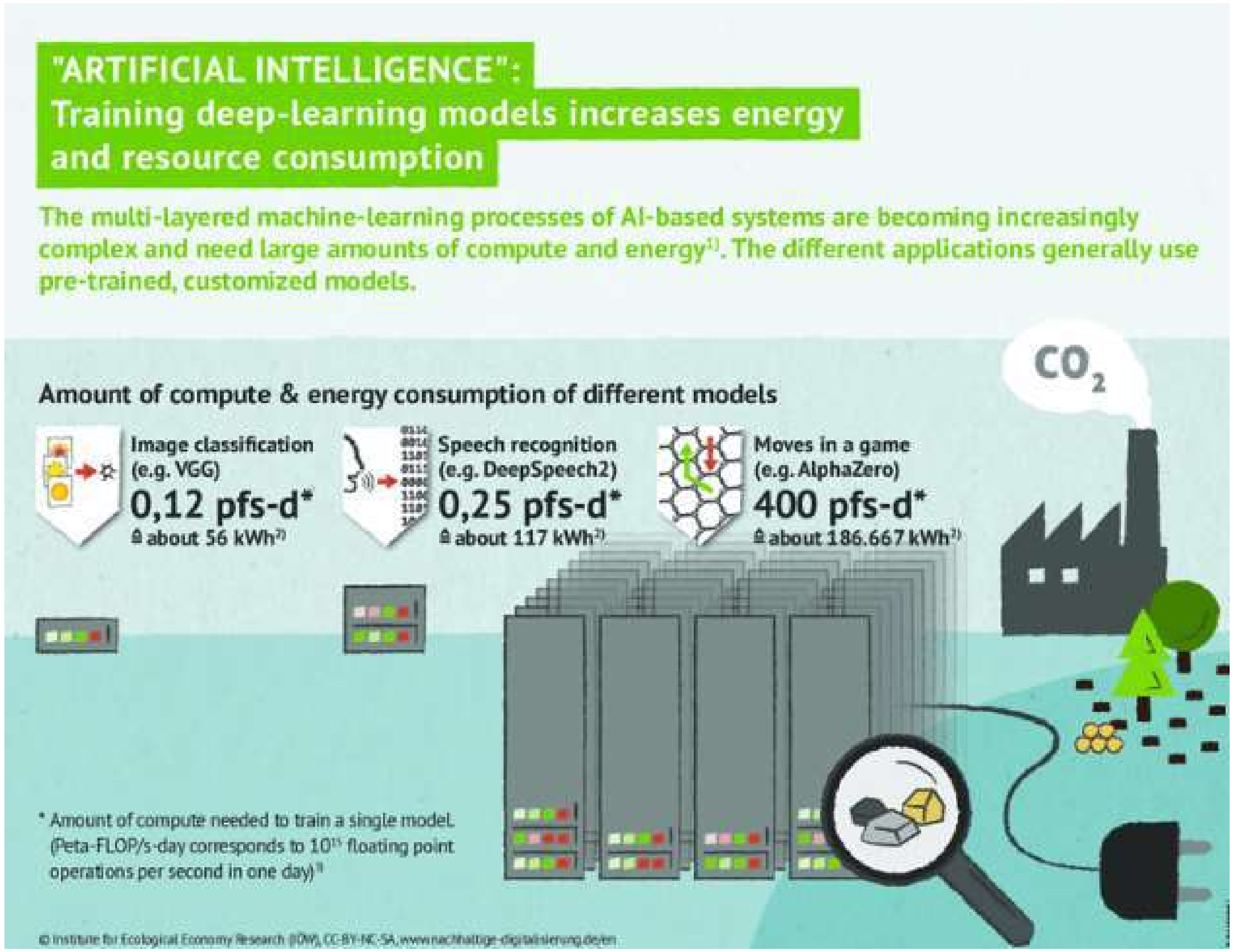


Table 5: Case study 5: 1D Allen-Cahn phase-field PDE in Eq. (62). We report the mean Squared Error (MSE) and percentiles (median, 5%–95%) of the L^2 approximation error for the test set. We use 2400 training functions. Here, we depict, indicatively, the results with a vanilla DeepOnet with 2 hidden layers with $[N, N]$ neurons. We set $N = 5, 10, 20, 40$. DeepOnets are trained with 50′000 Adam iterations (with learning rate 0.001 and then 0.0001). We report the RandONets encompassing Johnson-Lindenstrauss (JL) Featured branch network (with $M = 40$ neurons) and the Random Fourier Feature branch Network (RFFN) (with $M = 2000$).

ML-model	MSE	5% L^2	median–L^2	95% L^2	comp. time
DeepOnet [5, 5]	1.75E−03	1.57E−01	3.50E−01	7.28E−01	3.20E+03 (GPU)
DeepOnet [10, 10]	1.60E−04	6.46E−02	1.04E−01	2.12E−01	3.03E+03 (GPU)
DeepOnet [20, 20]	5.62E−05	3.49E−02	5.52E−02	1.28E−01	3.39E+03 (GPU)
DeepOnet [40, 40]	3.10E−05	2.39E−02	4.06E−02	9.38E−02	3.49E+03 (GPU)
RandONet–JL (40)	2.42E−04	7.61E−02	1.04E−01	2.77E−01	2.32E−02 (CPU)
RandONet–RFFN (2000)	3.15E−10	1.28E−04	1.60E−04	2.60E−04	3.20E+00 (CPU)

ChatGPT consumes as much energy per day as 170.000 homes in the US


"ARTIFICIAL INTELLIGENCE":
Training deep-learning models increases energy and resource consumption

The multi-layered machine-learning processes of AI-based systems are becoming increasingly complex and need large amounts of compute and energy[1]. The different applications generally use pre-trained, customized models.

Amount of compute & energy consumption of different models

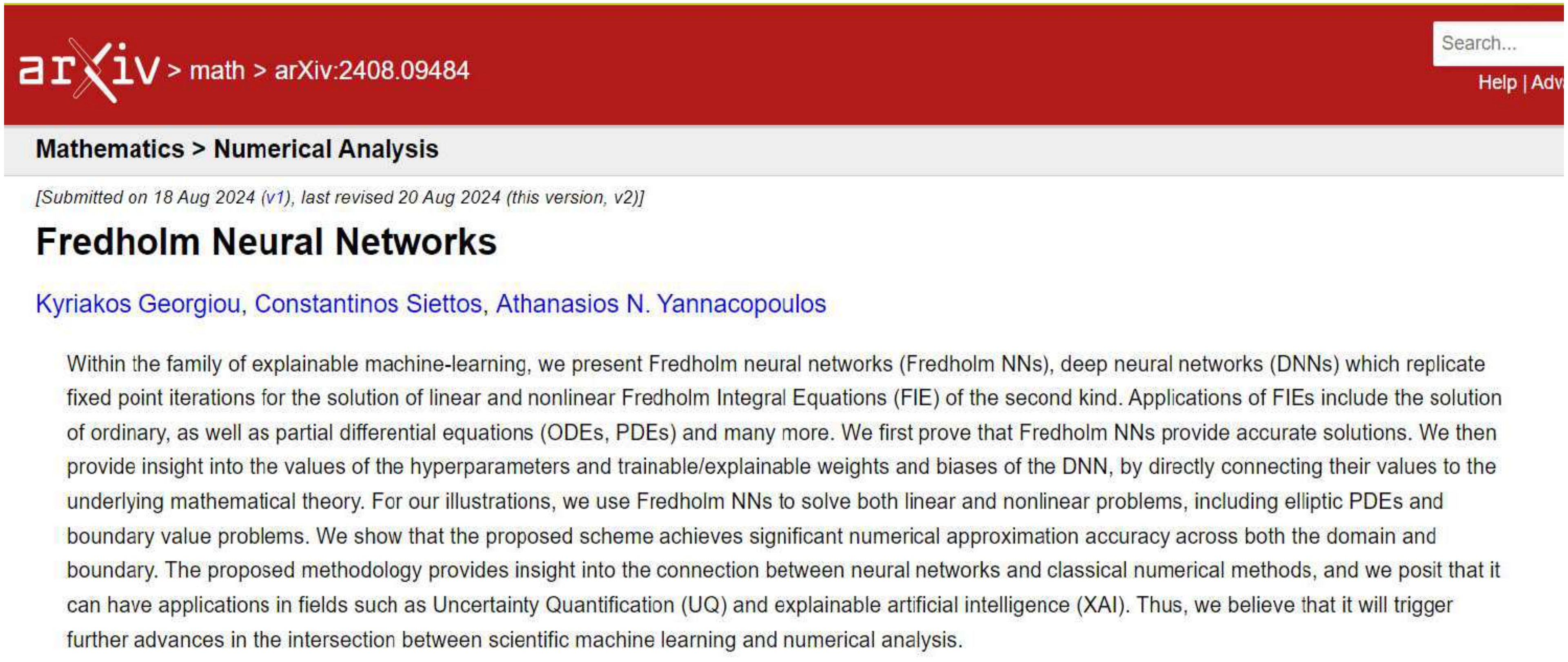
Image classification (e.g. VGG)
0,12 pfs-d*
about 56 kWh[2]

Speech recognition (e.g. DeepSpeech2)
0,25 pfs-d*
about 117 kWh[2]

Moves in a game (e.g. AlphaZero)
400 pfs-d*
about 186.667 kWh[2]

* Amount of compute needed to train a single model. (Peta-FLOP/s-day corresponds to 10^{15} floating point operations per second in one day)[1]

Deep Learning with black-box optimization is NOT appropriate for Accurate Numerical Analysis



Mathematics > Numerical Analysis

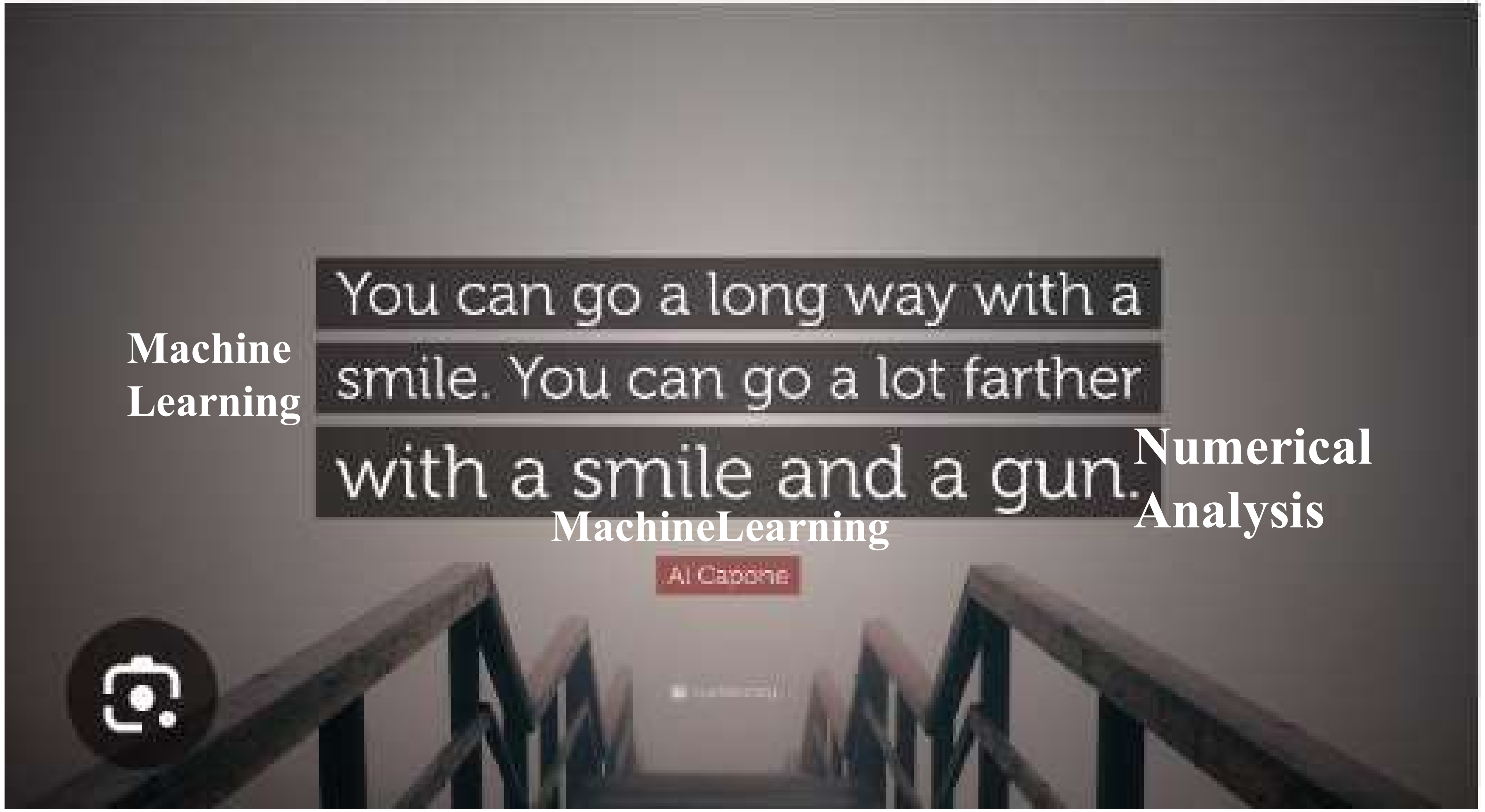
[Submitted on 18 Aug 2024 (v1), last revised 20 Aug 2024 (this version, v2)]

Fredholm Neural Networks

Kyriakos Georgiou, Constantinos Siettos, Athanasios N. Yannacopoulos

Within the family of explainable machine-learning, we present Fredholm neural networks (Fredholm NNs), deep neural networks (DNNs) which replicate fixed point iterations for the solution of linear and nonlinear Fredholm Integral Equations (FIE) of the second kind. Applications of FIEs include the solution of ordinary, as well as partial differential equations (ODEs, PDEs) and many more. We first prove that Fredholm NNs provide accurate solutions. We then provide insight into the values of the hyperparameters and trainable/explainable weights and biases of the DNN, by directly connecting their values to the underlying mathematical theory. For our illustrations, we use Fredholm NNs to solve both linear and nonlinear problems, including elliptic PDEs and boundary value problems. We show that the proposed scheme achieves significant numerical approximation accuracy across both the domain and boundary. The proposed methodology provides insight into the connection between neural networks and classical numerical methods, and we posit that it can have applications in fields such as Uncertainty Quantification (UQ) and explainable artificial intelligence (XAI). Thus, we believe that it will trigger further advances in the intersection between scientific machine learning and numerical analysis.

Within the family of explainable machine-learning, we present Fredholm neural networks (Fredholm NNs), deep neural networks (DNNs) which replicate fixed point iterations for the solution of linear and nonlinear Fredholm Integral Equations (FIE) of the second kind.


You can go a long way with a smile. You can go a lot farther with a smile and a gun.
Al Capone

Machine Learning

MachineLearning

Numerical Analysis

THE GROUP

Dr. Alessandro della Pia

Dr. Kyriakos Georgiou

Dr. Dimitris Patsatzis

Gianluca Fabiani

Hector Vargas Alvarez

Gianmaria Viola

Prof. Yannis Kevrekidis
Dept. of Chem. Eng.,
Dept. of Applied
Mathematics

Johns Hopkins University

Dr. Lucia Russo
Istituto di Scienze e
Tecnologie per l'Energia e la
Mobilità Sostenibili, Consiglio

Prof. Nikos Kazantzis
Dept. of Chemical Engineering
Worcester Polytechnic Institute

Thanos Yannakopoulos
Dept. of Statistics
AUEB, Greece

Felix Dieterich
TUM

Some recent references

Patsatzis, D. G., Russo, L., & Siettos, C., 2024. A physics-informed neural network method for the approximation of slow invariant manifolds for the general class of stiff systems of ODEs. SIAM J. Applied Dynamical Systems, arXiv preprint arXiv:2403.11591

Patsatzis, D.G., Fabiani, G., Russo, L. and Siettos, C., 2024. Slow invariant manifolds of singularly perturbed systems via physics-informed machine learning. *SIAM J. Scientific Computing*, 46 (4), C297-C322

Fabiani, G., Evangelou, N., Cui, T., Bello-Rivas, J.M., Martin-Linares, C.P., Siettos, C. and Kevrekidis, I.G., 2024. Task-oriented machine learning surrogates for tipping points of agent-based models. *Nature Communications*, *15*(1), p.4117. **Featured Article**

Gallos, I.K., Lehmberg, D., Dietrich, F. and Siettos, C., 2024. Data-driven modelling of brain activity using neural networks, diffusion maps, and the Koopman operator. *Chaos*, *34*(1). **Featured Article**

Vargas Alvarez H, Fabiani G, Kazantzis N, Siettos C, Kevrekidis IG, 2023, Discrete-Time Nonlinear Feedback Linearization via Physics-Informed Machine Learning. *Journal of Computational Physics*, 478, 111953.

Patsatzis D, Russo L, Kevrekidis IG, Siettos C, 2023, Data-driven Control of Agent-based Models: an Equation/ Variable-free Machine Learning Approach, *Journal of Computational Physics*, **478**, 111953.

Fabiani G, Galaris E, Russo L, Siettos C, 2023, Parsimonious Physics-Informed Random Projection Neural Networks for Initial Value Problems of ODEs and Index-1 DAEs, *Chaos*, 33, 043128 **Editor’s Pick**

Galaris E, Fabiani G, Gallos I, Kevrekidis I, Siettos C, 2022, Numerical Bifurcation Analysis of PDEs from Lattice Boltzmann Model Simulations: a Parsimonious Machine Learning Approach, *Journal of Scientific Computing*, **92**, 34.

Papaioannou P, Talmon R, Kevrekidis I, Siettos C, 2022, Time-series Forecasting using Manifold learning, Radial Basis Function Interpolation and Geometric Harmonics, *Chaos*, **32**, 083113.