Neural networks and dynamical systems

Sandesh Athni Hiremath

Department of Mechanical and Process Engineering
TU Kaiserslautern

Data Science Applications in Physics,
Balkan School in Tirana
22^{nd} - 26^{th}, January, 2024

Course Outline

Dynamical Systems

- Abstract definition
- Differential equations as dynamical systems

Numerical methods

- Deterministic methods
- Stochastic methods

Statistical methods

- Neural networks
- Approximation theorems

Neural networks and dynamical systems

- Examples

Dynamical system

Definition (Dynamical system)

Let $\mathbb{T} = \mathbb{R}$ or $\mathbb{T} = \mathbb{Z}$, respectively, and let X be a metric space. A dynamical system is a triple $(\Phi, \mathbb{T}, X)$, where the continuous mapping $\Phi : \mathbb{T} \times X \to X$ satisfies the following properties:

- Initial value (identity) condition: $\Phi_0 x = x$ for all $x \in X$
- Group property: $\Phi_{s+t} x = \Phi_s \Phi_t x = \Phi_t \Phi_s x$ for all $x \in X$ and for all $s, t \in \mathbb{T}$.

  - $\mathbb{T}$ is called the parameter space.
  - X is called the phase space.
  - Φ is called the evolution operator.

Dynamical system

- If $\mathbb{T} = \mathbb{R}$, then the dynamical system is called continuous, and in case $\mathbb{T} = \mathbb{Z}$, one has a discrete dynamical system.

- The second property of a dynamical system is called group property because the family of mappings $\{\Phi_t : t \in \mathbb{T}\}$ of X into itself forms a group under composition.

- If the family of mappings $\{\Phi_t : t \in \mathbb{T}_0^+\}$ of X into itself forms a semigroup under composition, then the triple $(\Phi, \mathbb{T}_0^+, X)$ is said to be a semi-dynamical system.

- The parameter space $\mathbb{T}$ can also be multi-dimensional, for e.g. $\mathbb{T} = \mathbb{R}^d$, $d \geq 2$. In this case the evolution operator Φ is a generalized non-autonomous evolution operator.

Example (Shift operator)

Let $\mathbb{T} = \mathbb{R}$, $d \in \mathbb{N}$, let X be the space $C(\mathbb{R}, \mathbb{R}^d)$, of bounded continuous functions $\mathrm{x} : \mathbb{R} \to \mathbb{R}^d$ with the norm $\|\mathrm{x}\|_X = \sup_{t \in \mathbb{T}} \mathrm{x}(t)$ for all $\mathrm{x} \in C(\mathbb{R}, \mathbb{R}^d)$. Let $(\theta_t)_{t \in \mathbb{R}}$ be the family of shift operators on X, defined by $\theta_t \mathrm{x} = \mathrm{x}(t + \cdot)$ for all $t \in \mathbb{R}$. Define Φ on $\mathbb{T} \times X$ as $\Phi_0 \mathrm{x} = \mathrm{x}$ and $\Phi_t \mathrm{x} = \theta_t \mathrm{x}$ for all $t \in \mathbb{R}$. Then $(\Phi, X, \mathbb{T})$ forms a continuous dynamical system.

Example (ODEs)

Let $\mathbb{T} = \mathbb{R}$, $X = \mathbb{R}^d$ and $A \in L(X, X)$ be a bounded linear operator

$$\frac{d}{dt}\mathrm{x}(t) = A\,\mathrm{x}, \quad t > 0$$
$$\mathrm{x}(0) = \mathrm{x}_0, \quad \mathrm{x}_0 \in X$$

Defining $\Phi_t := e^{At}$, then $(\Phi_t, X, \mathbb{T}_0^+)$ forms a semi-dynamical system.

Example (PDEs)

Let $\mathbb{T} = \mathbb{R}$, $\mathfrak{D} \subset \mathbb{R}^d$, $X = L^2(\mathfrak{D})$ and $A \in L(D(A), X)$ be a closed linear operator (e.g. $A := \Delta = \sum_{i=1}^{d} \partial^2_{x_i}$)

$$\frac{d}{dt}\mathrm{x}(t) = A\,\mathrm{x}, \quad t > 0$$
$$\mathrm{x}(0) = \mathrm{x}_0, \quad \mathrm{x}_0 \in X$$

Let $(T_t)_{t\geq 0} : X \to X$ be the operator semi-group generated by A, then defining $\Phi_t := T_t$, we get $(\Phi_t, X, \mathbb{T}_0^+)$ forms a semi-dynamical system.

- Differential equations provide a way to specify dynamical systems.

Differential equations

Let $\mathbb{T} = \mathbb{R}$, $X = \mathbb{R}^d$ and $F : \mathbb{T} \times X \to X$ be a smooth vector field. Consider the initial value problem (IVP)

$$\begin{aligned} \frac{d}{dt}\mathrm{x}(t) &= F(t, \mathrm{x}(t)), \quad \mathbb{T}_0^+ \ni t > 0 \\ \mathrm{x}(0) &= \mathrm{x}_0, \quad \mathrm{x}_0 \in X \end{aligned}$$

Alternatively, the IVP can be written in the following integral form:

$$\mathrm{x}(t) = \mathrm{x}_0 + \int_0^t F(s, \mathrm{x}(s))\, ds$$

- Existence: If $F \in C(\mathbb{T} \times X; X)$ then there exists at least one solution.
- Uniqueness: Additionally, if F is locally Lipschitz continuous, i.e. for any compact set $K \in \mathbb{T} \times X$ there exists a corresponding Lipschitz constant L_K such that:

$$\|F(t, \mathrm{x}_1) - F(t, \mathrm{x}_2)\|_X \leq L_K \|\mathrm{x}_1 - \mathrm{x}_2\|_X, \quad \forall (t, \mathrm{x}_1), (t, \mathrm{x}_2) \in K,$$

then we get global uniqueness of the solution.

Deterministic methods

Based on Taylor series expansion of the solution $\mathrm{x}(t)$ at t_0, for $\tau > 0$ we get

$$\mathrm{x}(t_0 + \tau) = \mathrm{x}(t_0) + \tau\, \mathrm{x}'(t_0) + \mathcal{O}(\tau^2) = \mathrm{x}_0 + \tau\, F(t_0, \mathrm{x}_0) + \mathcal{O}(\tau^2).$$

Letting $t_n = t_0 + n\tau$, $\mathrm{x}_n = \mathrm{x}(t_n)$ we get the following numerical schemes:

- Euler method: $\mathrm{x}_{n+1} = \mathrm{x}_n + \tau F(t_n, \mathrm{x}_n)$,
  $\mathrm{x}_{n+1} = \mathrm{x}_n + \tau F(t_{n+1}, \mathrm{x}_{n+1})$,
- s-stage Runge-Kutta method:

$$k_i = F(t + c_i\tau, \mathrm{x}_n + \tau \sum_{j=1}^{i-1} a_{ij} k_j), \quad i = 1, \ldots, s$$
$$\mathrm{x}_{n+1} = \mathrm{x}_n + \tau \sum_{i=1}^{s} b_i k_i$$

- Collocation method: Find polynomial $g \in \mathbf{P}_s^d$ of degree s such that
  - $g(t_n) = \mathrm{x}_n$
  - $g'(t_n + c_i\tau) = F(t_n + c_i\tau, g(t_n + c_i\tau))$
  - $\mathrm{x}_{n+1} = g(t_n + \tau)$

Deterministic methods

- Collocation method: Let $\{L_1, \ldots, L_s\}$ be the Lagrange basis of the polynomial space $\mathbf{P}^d_{s-1}$ of degree $s-1$ with respect to the nodes $c_1, \ldots, c_s$, with $0 \leq c_1 < \cdots < c_s \leq 1$, then:
  - $k_i = g'(t_n + c_i\tau)$
  - $g'(t_n + \delta\tau) = \sum_{j=1}^{s} k_j L_j(\delta)$
  - $g(t_n + c_i\tau) = \mathrm{x}_n + \tau \int_0^{c_i} g'(t_n + \delta\tau)d\delta = \mathrm{x}_n + \tau \sum_{j=1}^{s} a_{ij}k_j$

    $$a_{ij} = \int_0^{c_i} L_j(\delta)d\delta, \quad \text{for } \; i, j = 1, \ldots, s.$$
  - $k_i = F(t_n + c_i\tau, \mathrm{x}_n + \tau \sum_{j=1}^{s} a_{ij}k_j)$, i = 1,. . . , s.
  - $x_{n+1} = g(t_n + \tau) = \mathrm{x}_n + \tau \int_0^1 g'(t_n + \delta\tau)d\delta = \mathrm{x}_n + \tau \sum_{j=1}^{s} b_j k_j,$

    $$b_j = \int_0^1 L_j(\delta)d\delta, \quad \mathrm{j} = 1, \ldots, \mathrm{s}.$$
- Results in a set of non-linear equations which needs to be solved using for e.g. Newton method.
- Is equivalent to implicit s-stage Runge-Kutta method.

Stochastic methods

- Monte carlo integration: Let $f : \mathbb{R}^d \to \mathbb{R}^d$, we are interested in numerical approximation $\hat{I}$ of the integral $I = \int_{\mathfrak{D}} f(x)dx$, $\mathfrak{D} \subset \mathbb{R}^d$ compact. Let $P = \{x_0, \ldots, x_n\}$ represent the partitioning of the set $\mathfrak{D}$.
- Deterministic approach is to use some type of quadrature rules such as:
  - Midpoint rule: $\hat{I}_n = \sum_{i=1}^n f(m_i)\Delta x_i$, $\Delta x_i := x_i - x_{i-1}$ and $m_i = \frac{x_i + x_{i-1}}{2}$
  - Trapezoidal rule: $\hat{I}_n = \sum_{i=1}^n [f(x_i) + f(x_{i-1})] \frac{\Delta x_i}{2}$,
  - General quadrature rule: $\hat{I}_n = \sum_{i=1}^n w_i f(x_i^*)$, $x_i^* \in [x_{i-1}, x_i]$
- $I = \lim_{n\to\infty} \hat{I}_n$
- Let $\varepsilon \in (0, 1)$. In general, it can be shown that: to achieve a given ε accuracy the minimum number of discretization points $n(\varepsilon)$ required is proportional to $\frac{|\mathfrak{D}|}{\varepsilon^d}$.
- Thus larger the value of d, i.e. the dimension of the integration domain, exponentially higher the number of discretization points needed. This is called the curse of dimensionality.

Stochastic methods

- Monte carlo integration: A simple method to get rid of this curse is to replace the deterministic grid points $\{x_i\}_{i=0}^n$ by random variables $\{X_i\}_{i=0}^n$.
- $I = \int_a^b f(x)dx = \mathbb{E}_{X \in \mathcal{U}(\mathfrak{D})}[f(X)]$
- Consequently, we get that $\hat{I}_n = \frac{|\mathfrak{D}|}{n} \sum_{i=1}^n f(X_i)$.
- It can be shown that the minimal number of samples, $n(\varepsilon)$ required to achieve a given accuracy ε is proportional to $\frac{\sigma^2(f)}{\varepsilon^2}$.

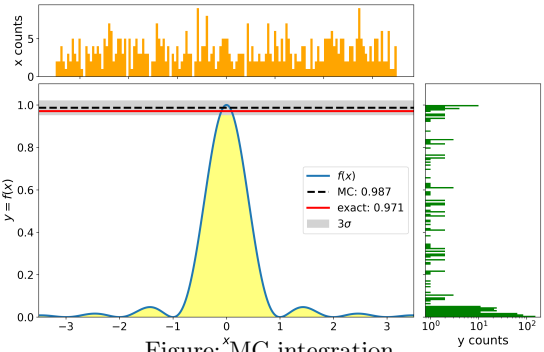

Figure: MC integration

One can now, apply this approach to solve differential equation.

Stochastic methods

Consider the initial value problem (IVP)

$$\frac{d}{dt}\mathrm{x}(t) = F(t), \text{ with } \mathrm{x}(0) = \mathrm{x}_0, \quad \mathrm{x}_0 \in X$$

Its solution can be written as

$$\begin{aligned}\mathrm{x}(t) &= \mathrm{x}_0 + \int_0^t F(s)\, ds = \mathrm{x}_0 + \sum_{i=1}^{N} \int_{t_{i-1}}^{t_i} F(s)\, ds \\ &= \mathrm{x}_0 + \sum_{i=1}^{N} \mathbb{E}_{S\in\mathcal{U}(t_{i-1},t_i)}[F(S)] \\ &= \mathrm{x}_0 + \sum_{i=1}^{N} \Big[\frac{t_{i-1}-t_i}{M} \sum_{k=1}^{M} F(S_k)\Big]\end{aligned}$$

Writing in an iterative manner we get

$$\begin{aligned}\mathrm{x}_{n+1} &= \mathrm{x}_n + \mathbb{E}_{S\in\mathcal{U}(t_n,t_{n+1})}[F(S)] \\ &= \mathrm{x}_n + \frac{t_{n+1}-t_n}{M} \sum_{k=1}^{M} F(S_k).\end{aligned}$$

Similarly, for nonlinear IVP

$$\frac{d}{dt}\mathrm{x}(t) = F(t,\mathrm{x}),\ \text{with } \mathrm{x}(0) = \mathrm{x}_0, \quad \mathrm{x}_0 \in X$$

we get

$$\begin{aligned}\mathrm{x}_{n+1} &= \mathrm{x}_n + \mathbb{E}_{S\in\mathcal{U}(t_n,t_{n+1})}[F(S,\mathrm{x}(S))] \\ &= \mathrm{x}_n + \frac{t_{n+1}-t_n}{M}\sum_{k=1}^{M} F(S_k,\mathrm{x}_n).\end{aligned}$$

Let $F(t,x) := t\sqrt{|x|} + \sin^3(\frac{\pi}{2}t) - 3H(t-2)$, with H being the Heaviside function, then the MC integration based solution is as shown in the Figure on the right.

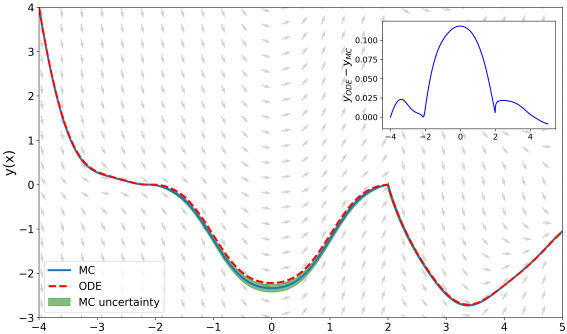

Figure: MC based ODE integration

Statistical problem

- Consider a set of data $\mathbb{D}$ comprising of pairs for inputs and outputs

$$\mathbb{D} := \{(X_1, Y_1), \ldots, (X_N, Y_N)\} \subset \mathbf{X} \times \mathbf{Y}$$

  assumed to be independent and identically distributed observations sampled from an unknown probability distribution $\mathbb{P}$.

- The task is to find a mathematical relation, F, between the input $X \in \mathbf{X}$ and the output $Y \in \mathbf{Y}$. In other words, find a mapping $X \mapsto F(X) =: \hat{Y}$ such that $F : \mathbf{X} \to \mathbf{Y}$ fits (explains) the observed data $\mathbb{D}$ is some optimal sense.

- To measure the quality of the estimated relation F, we need to define a loss functional

$$L : \mathbf{Y} \times \mathbf{Y} \to \mathbb{R}$$

  such that $L(Y, \hat{Y})$ is able to quantify the closeness between the observed output Y and the estimated output $\hat{Y}$.

Typical loss functions

Some common loss functions are

- $L(Y, \hat{Y}) = \|Y - \hat{Y}\|^2_{L^2(\mathbf{Y})}$, squared L^2 norm
- $L(Y, \hat{Y}) = \|Y - \hat{Y}\|^{2p}_{L^p(\mathbf{Y})}$, squared L^p norm with $p \in (0, \infty]$
- $L(Y, \hat{Y}) = -H(Y, \hat{Y}) = \mathbb{E}_Y[\log \hat{Y}]$, cross-entropy of $\hat{Y}$ with respect to Y

The quality of the estimated function F is given by:

$$E(F) = \mathbb{E}_{\mathbb{P}}[L(Y, F(X))] = \int_{\mathbf{X}} L(Y, F(\mathrm{x}))\mathbb{P}(d\mathrm{x})$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} L(Y_i, F(X_i))$$

$\Rightarrow$ F can be estimated by solving the following statistical optimization problem

$$\hat{F} = \underset{F}{\operatorname{argmin}}\ E(F)$$

If $F = F(\cdot; \theta)$ is parameterized by θ, then

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\ E(F(\cdot; \theta)).$$

Typical models for function F

- The type of function models used can be categorized into parametric and non-parametric models.
- Here we consider only parametric models
- Some common parametric function models are:
  - Linear model: $F(X;\theta) = a^T X + b, \theta = (a, b)$
  - Quadratic model: $F(X;\theta) = a^T X^2 + b^T X + c, \theta = (a, b, c)$
  - Polynomial model: $F(X;\theta) = \sum_{i=1}^{n} \theta_i^T X^i, \theta = (\theta_i)$
  - Exponential model: $F(X;\theta) = e^{a^T X + b}, \theta = (a, b)$
  - Perceptron model: $F(X;\theta) = \sigma(\sum_{i=1}^{n} X_i^T w_i + b), \theta = ((w_i), b)$
  - Multi-layer-perceptron model:
    $F(X;\theta) = \sigma\Big(\sum_{k=1}^{n} \sigma\big(\ldots\sigma(\sum_{i=1}^{n} X_i^T w_i + b_1) + b_{m-1}\big) + b_m\Big), \theta = ((w_i), (b_i))$

Feedforward neural network

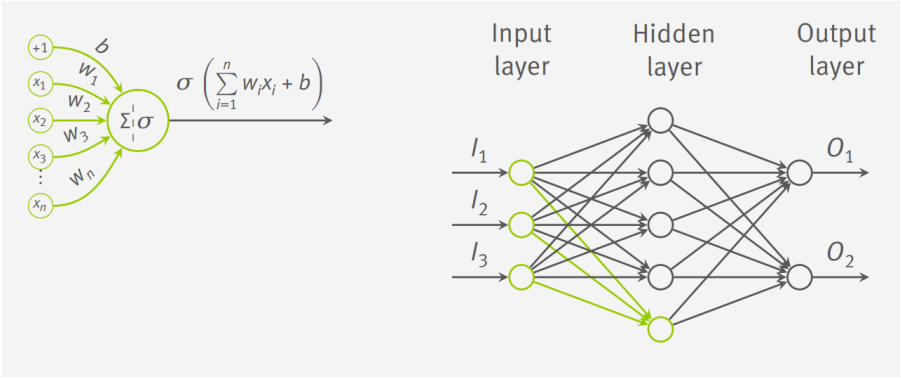

Figure: Example of a feedforward neural network [1]

[1] https://github.com/PetarV-/TikZ/tree/master/Multilayer%20perceptron

Universal approximation theorems

Theorem ([7,8] Single layer and arbitrary width)

Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a continuous activation function other than a polynomial. Let $d, D \in \mathbb{N}$ then for every continuous function $f : \mathbb{R}^d \to \mathbb{R}^D$, every compact subset $K \subset \mathbb{R}^d$ and every $\varepsilon > 0$ there exists a single-layer neural network $\hat{f} = \eta_0 + \sum_{j=1}^{N} \eta_j \sigma(w_j^T x + b_j)$ with $N \in \mathbb{N}$, such that $\sup_{x \in K} |\hat{f}(x) - f(x)| \leq \varepsilon$.

Theorem ([9,10] Constrained width and arbitrary depth)

Let $X \subset \mathbb{R}^d$ be compact. Let $\sigma : \mathbb{R} \to \mathbb{R}$ be any non-affine continuous function that is continuously differentiable with non-zero derivative atleast at one point. Let $\mathcal{N}^{\sigma}_{d,D:d+D+2}$ denote the space of feed forward neural networks with d input neurons, D output neurons, and of arbitrary depth each with $d + D + 2$ neurons, such that hidden neurons have activation function σ and output neurons have identity activation function. Then given any $\varepsilon > 0$ and any $f \in C(X, \mathbb{R}^D)$, there exists $\hat{f} \in \mathcal{N}^{\sigma}_{d,D:d+D+2}$ such that $\sup_{x \in X} |\hat{f}(x) - f| \leq \varepsilon$.

Problem

Let $I \subset (0, \infty)$ be time interval, $\mathfrak{D} \subset \mathbb{R}^d$ be an open bounded domain and $\mathfrak{D}_I = I \times \mathfrak{D}$ denote the open space-time cylinder. Let a semi-dynamical system be specified by a generic differential equation of the form

$$\begin{aligned} F(t, x, y(t,x), \partial_t y(t,x), \nabla_x y(t,x), \nabla_x^2 y(t,x)) &= 0 \quad \text{on } \mathfrak{D}_I, \\ G(t, x, y(t,x), \nabla_x y(t,x)) &= 0 \quad \text{on } \partial\mathfrak{D} \times I, \\ y(0,x) &= y_0(x) \quad \forall x \in \mathfrak{D}, \end{aligned}$$

where $y : \overline{\mathfrak{D}}_I \to \mathbb{R}$, $F : I \times \mathfrak{D} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R}^d \times \mathbb{R}^{d \times d} \to \mathbb{R}$, $G : I \times \partial\mathfrak{D} \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$, $y_0 : \mathfrak{D} \to \mathbb{R}$. The task is to find the solution $y(t,x)$ for all $(t,x) \in \overline{\mathfrak{D}}_I$.

The goal is to computationally solve the above equation using a neural network.

General steps

- Let $y(t,x)$ be given by a surrogate function $\psi(t,x;\theta)$ parameterized by $\theta \in \mathbb{R}^n$. This is to say let $y(t,x) = \psi(t,x;\theta)$.
- $\psi(t,x;\theta)$ represents a neural network with internal parameters θ.
- define a loss function to enable learning of θ parameter.
- possible types of loss functions:
  - Residual loss: $E(\theta) = \int_{\mathfrak{D}_I} F(t,x,\psi_\theta,\partial_t\psi_\theta,\nabla_x\psi_\theta,\nabla_x^2\psi_\theta)^2 dx\,dt$
  - Variational formulation:

$$E(\psi_\theta) = \int_{\mathfrak{D}_I} f(t,x,\psi_\theta,\partial_t\psi_\theta,\nabla_x\psi_\theta,\nabla_x^2\psi_\theta) dx\,dt$$

$$\hat{\psi_\theta} = \underset{\psi_\theta}{\operatorname{argmin}}\ E(\psi_\theta)$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\ E(\psi_\theta)$$

Monte carlo integration

For $(t_i, x_i) =: \xi_i \in \mathcal{U}(\mathfrak{D}_I)$ we can write

$$\begin{aligned} E &= \int_{\mathfrak{D}_I} F(t, x, \psi_\theta, \partial_t \psi_\theta, \nabla_x \psi_\theta, \nabla_x^2 \psi_\theta)^2 dx\, dt \\ &\propto \frac{|\mathfrak{D}_I|}{N} \sum_{i=1}^{N} F(\xi_i, \psi_\theta(\xi_i), \partial_t \psi_\theta(\xi_i), \nabla_x \psi_\theta(\xi_i), \nabla_x^2 \psi_\theta(\xi_i))^2 \\ &= \frac{|\mathfrak{D}_I|}{N} \sum_{i=1}^{N} E_i \end{aligned}$$

Thus $\xi_i \in \mathfrak{D}_I$ can be interpreted as a training sample.

- ξ_i can be some fixed collocation points that may be given as discrete mesh (uniform or triangulated), quasi-uniformly distributed points
- potentially infinite training data

Initial and boundary conditions

Initial and boundary conditions can be enforced in the following manner.

- soft assignment: add penalty term to the loss function

$$\begin{aligned} E(\theta) = &\|F(t, x, \psi_\theta, \partial_t \psi_\theta, \nabla_x \psi_\theta, \nabla_x^2 \psi_\theta)\|^2_{\mathfrak{D}_I} \\ &+ \lambda_1 \|G(t, x, \psi_\theta, \nabla_x \psi_\theta)\|^2_{\partial \mathfrak{D}_I} \\ &+ \lambda_2 \|\psi_\theta(0, x) - y_0(x)\|^2_{\mathfrak{D}} \end{aligned}$$

- hard assignment: use an ansatz function that satisifies the boundary condition by construction.

$$\psi_\theta(t, x) = \alpha(t, x) + \beta(t, x) + \gamma(t, x, \phi_\theta)$$

where α satisfies the initial condition, β satisfies the boundary condition and γ is zero on the boundary and only valid on $\mathfrak{D}_I$.

Example

Consider the 1D linear diffusion equation on $\mathfrak{D} = (0,1)$ and $t > 0$

$$\begin{aligned} \partial_t y(t,x) &= \partial_x^2 y(t,x) + f(t,x), \quad x \in (0,1), t > 0 \\ y(t,0) &= g_0(t), y(t,1) = g_1(t) \\ y(0,x) &= y_0(x), \quad x \in (0,1) \end{aligned}$$

For this case, one could use the following ansatz function:

$$\begin{aligned} \psi_\theta(t,x) = \overbrace{\delta_0(t) y_0(x)}^{\alpha(t,x)} + \overbrace{\delta_{t>0}(t)\delta_{\{0,1\}}(x)[(1-x)g_0(t) + x g_1(t)]}^{\beta(t,x)} \\ + \overbrace{(1 - e^{\frac{t}{1+t}})[x(1-x)\phi_\theta]}^{\gamma(t,x,\phi_\theta)} \end{aligned}$$

$$\Rightarrow \psi_\theta(0,0) = y_0(0),\ \psi_\theta(0,1) = y_0(1),\ \psi_\theta(0,x) = y_0(x)$$

$$\Rightarrow \psi_\theta(t,0) = g_0(t),\ \psi_\theta(t,1) = g_1(t),\ \psi_\theta(t,x) = \gamma(t,x,\phi_\theta)$$

Irregular domain

To deal with complex and irregular domains $\mathfrak{D}$, one can proceed in the following manner:

- define a distance function $D(x)$ which represents the distance to the boundary $\partial\mathfrak{D}$ and use it construct the ansatz function.
- more practical would be to define the ansatz function as:

$$\psi_\theta(t,x) = D(x)P(t)y_0(x) + \beta(t,x) + (1 - P(t))D(x)\phi_\theta$$

and let $D(x)$, $P(t)$ and $\beta(t,x)$ be approximated again by some neural network-

- $D_\eta : \overline{\mathfrak{D}} \to \mathbb{R}$ to approximate D
- $\beta_\kappa : \partial\mathfrak{D} \times I \to \mathbb{R}$ to approximate $\beta(t,x)$ i.e. $y(t,x)$ on $\partial\mathfrak{D} \times I$.
- $P_\rho : \bar{I} \to \mathbb{R}$ to some smooth time dependent function such that $P_\rho(0) = 1$, e.g. $e^{\frac{t}{1+t}}$.

1D advection equation

We want to solve the following PDE:

$$\frac{\partial \phi}{\partial t} + u\frac{\partial \phi}{\partial x} = 0 \quad \text{on } \mathfrak{D}_I$$
$$\phi(0, x) = v \quad \text{on } \mathfrak{D} \text{ for } t = 0.$$

- The independent variables (i.e, x and t) are used as input values for the NN, and the solution (i.e. ϕ) is the output.
- In order to find the solution, at each step the NN outputs are derived w.r.t the inputs.
- The loss function that matches the PDE is built and the weights are updated accordingly. If the loss function goes to zero, we can assume that our NN is indeed the solution to our PDE.
- We can also try to find a general solution for different values of u, v and also $\mathfrak{D}$ so it will be set also as an input.

1D advection equation

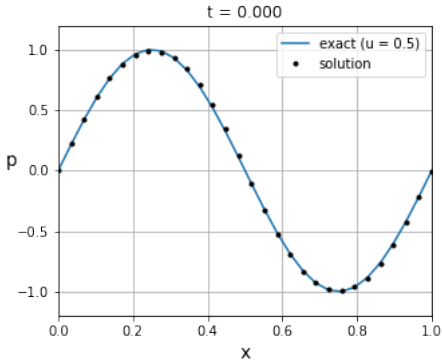

Figure: Simulation of 1D advection equation

Reaction diffusion equation

In this example we are going to solve a reaction diffusion equation.

$$
\begin{aligned}
u_t + \nabla \cdot (\sigma \nabla u) &= \rho u(1-u) \quad \text{on } \mathfrak{D}_I \\
\nabla_n u &= 0 \quad \text{on } \partial\mathfrak{D}_I \\
u_0 &= v \quad \text{on } \mathfrak{D} \text{ for } t = 0.
\end{aligned}
$$

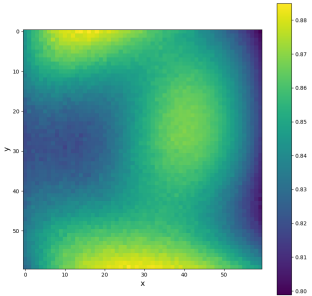

Figure: Simulation of 1D advection equation

Vortex transport equation

In this example we are going to solve the Euler equations for an isentropic two-dimensional vortex in a full-periodic square domain.

$$\rho_t + \nabla \cdot (\rho u) = 0 \text{ on } \mathfrak{D}_I$$
$$(\rho \mathbf{u})_t + (\mathbf{u} \cdot \nabla)(\rho \mathbf{u}) + \nabla p = 0 \text{ on } \mathfrak{D}_I$$
$$\mathbf{u} = \mathbf{v}, \quad \rho = r \text{ on } \partial\mathfrak{D}_I$$
$$\mathbf{u}_0 = u_0, \quad \rho_0 = r_0 \text{ on } \mathfrak{D} \text{ for } t = 0.$$

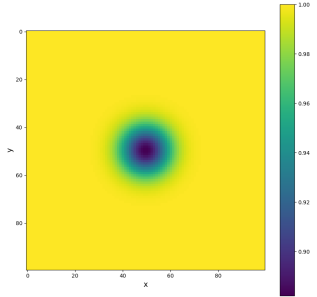

Figure: Simulation of 1D advection equation

Lotka-Volterra equation

We want to solve the following ODE:

$$\frac{dx(t)}{dt} = \alpha x(t) - \beta x(t) y(t) \quad \text{for } t > 0$$
$$\frac{dy(t)}{dt} = -\delta y(t) + \gamma x(t) y(t) \quad \text{for } t > 0$$
$$x(0) = x_0, \quad y(0) = y_0 \quad \text{for } t = 0.$$

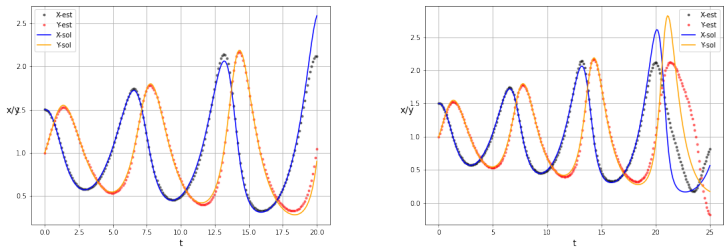

Figure: Simulation of Lotka-Volterra equation

References I

Foundational:

1. Deuflhard, P. and Bornemann, P. 2002. Scientific Computing with Ordinary Differential Equations. Springer-Verlag, Berlin, Heidelberg.

2. Kloeden, P. and Rasmussen, M. (2011). Nonautonomous Dynamical Systems.

Advanced:

3. Akhtar, M. et al., Solving Initial Value Ordinary Differential Equations By Monte Carlo Method, Proc. IAM, 4, 2, 2015, p 149–184,

4. Lagaris, I.E., Likas, A. and Fotiadis, D.I. "Artificial neural networks for solving ordinary and partial differential equations". en. In: IEEE Transactions on Neural Networks 9.5 (Sept. 1998). 00403, pp. 987–1000.

5. Berg, J. and Nyström, Kaj. "A unified deep artificial neural network approach to partial differential equations in complex geometries". In: Neurocomputing 317 (Nov. 2018). 00011, pp. 28–41.

6. Sirignano, J. and Spiliopoulos, J. "DGM: A deep learning algorithm for solving partial differential equations". en. In: Journal of Computational Physics 375 (Dec. 2018).pp. 1339–1364.

References II

7. Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function". Mathematics of Control, Signals, and Systems. 2 (4): 303–314. CiteSeerX 10.1.1.441.7873.

8. Hornik, K. (1991). "Approximation capabilities of multilayer feedforward networks". Neural Networks. 4 (2): 251–257.

9. Kidger, P. and Lyons, T. (July 2020). Universal Approximation with Deep Narrow Networks. Conference on Learning Theory.

10. Park, Y. , et al. "Minimum Width for Universal Approximation". (Sept 2020). ICLR.

11. Weinan, E. and Bing, Yu. "The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems". en. In: Communications in Mathematics and Statistics 6.1 (Mar. 2018). 00006, pp. 1–12

12. Neurodiffeq library: https://github.com/NeuroDiffGym/neurodiffeq

13. Physics Informed Neural network for Advanced modeling (PINA): https://mathlab.github.io/PINA/