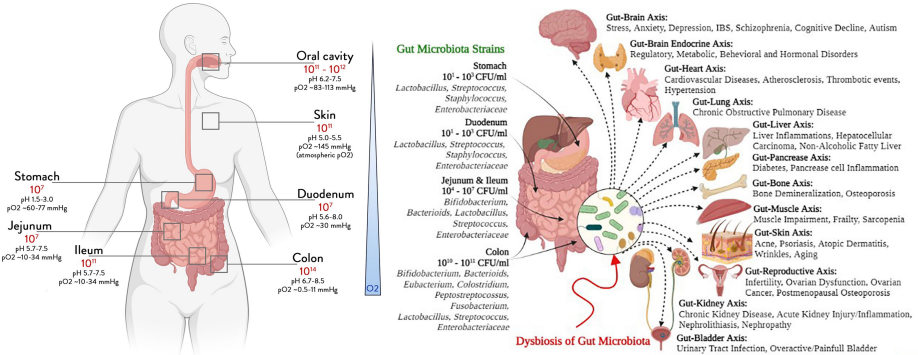
Estimation of interactions in microbial communities via a neural network-based generalized smoothing algorithm

Nicolas Brunel, Paguiel Javan Hossie, Béatrice Laroche, Lucas Perrin,

Thibault Malou, Thomas Saigre, Lorenzo Sala

12 October 2023

Our body's equilibrium is partly ensured by billions of bacteria that form assemblages called microbiota in different sites.



(a) Microbiota repartition in different sites (from *de Vos et. al.*)

(b) Gut microbial strains and negative health outcomes of gut microbial dysbiosis (from *Afzaal et. al.*)

- Microbiota plays a crucial role in various aspects of our well-being.
- A better understanding of the interactions between bacteria is needed to understand the role of microbiota in our health.
- It is composed of many species of bacteria.

Main objectives: comprehend the interactions between these bacteria, their relationship with pathogens, and their functions within the ecosystem.

Motivations

- Generalized Lokta-Volterra to model microbial interaction.
- From a data set obtained through experiments, we want to **estimate the parameters** involved in a model.
- In a previous work[1], this was done using the **Generalized Smoothing Algorithm** with **splines** as data interpolation.

[1] B. Laroche et al. "Parameter estimation for dynamical systems using an FDA approach". In: *11th International Conference of the ERCIM WG on Computational and Methodological Statistics (CMStatistics 2018)*. Pise, Italy, Dec. 2018.

Motivations

- Generalized Lokta-Volterra to model microbial interaction.
- From a data set obtained through experiments, we want to **estimate the parameters** involved in a model.
- In a previous work[1], this was done using the **Generalized Smoothing Algorithm** with **splines** as data interpolation.

Disadvantage: data interpolation with **splines** is the **costly** part of the estimation process.

- Main objective of the project: neural network to replace the spline smoothing.
- As this part tries to fit data points **and** a differential equation: investigate a **Physics-Informed Neural Network** approach.

[1] B. Laroche et al. "Parameter estimation for dynamical systems using an FDA approach". In: *11th International Conference of the ERCIM WG on Computational and Methodological Statistics (CMStatistics 2018)*. Pise, Italy, Dec. 2018.

Table of contents

Modeling biological data

- Models are not exact,
- Several experiments, different initial values, and conditions,
- Sparse and irregular sampling, depending on the experiment,
- Noise and missing data.

Modeling biological data

- Models are not exact,
- Several experiments, different initial values, and conditions,
- Sparse and irregular sampling, depending on the experiment,
- Noise and missing data.

Notations:

- N_s: number of species studied,
- N_{exp}: number of experiments conducted,
- N^e_{obs}: number of observations of the bacterial population of species i at times $t_k^{(e)}$ for $e \in [\![1, N_{\text{exp}}]\!]$,
- $\boldsymbol{U}_{i,k}^{(e)}$ data measured for the experiment e, on species i, at time $t_k^{(e)}$.

Generalized Lotka–Volterra Model[2] (GLV)

- For $i \in [\![1, N_s]\!]$, $x_i(t)$ represent the quantity of bacteria of population i.
- This quantity follows the ODE:

$$\frac{\partial}{\partial t} x_i(t) = \mu_i x_i(t) + \sum_{j=1}^{N_s} a_{ij} x_i(t) x_j(t), \quad t \in [0, t_{\max}]$$

where:

- μ_i represents the **intrinsic growth** rate of the bacterial population in the absence of interaction with other bacterial populations,
- a_{ij} describes the **interaction** coefficient representing the direct effect of species j on the species i.

[2] V. Volterra and M. Brelot. *Leçons sur la théorie mathématique de la lutte pour la vie*. eng. Paris : Gauthier-Villars, 1931.

Generalized Lotka–Volterra Model[2]

Setting $\boldsymbol{\mu} = [\mu_1, \cdots, \mu_{N_s}]^T$, $\boldsymbol{A} = (a_{ij})_{1 \leq i,j \leq N_s}$ and $u_i = \log(x_i)$, the GLV model can be written under the matrix form:

$$\frac{\partial}{\partial t} \begin{bmatrix} u_1(t) \\ \vdots \\ u_{N_s}(t) \end{bmatrix} = \boldsymbol{\mu} + \boldsymbol{A} \cdot \exp\left(\begin{bmatrix} u_1(t) \\ \vdots \\ u_{N_s}(t) \end{bmatrix} \right) \tag{GLV}$$

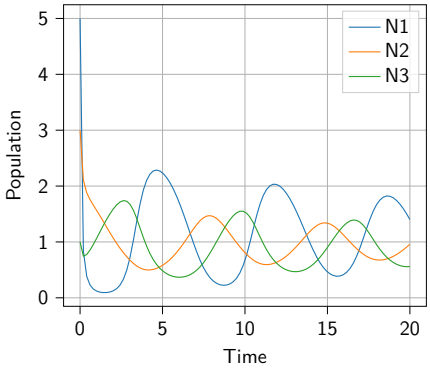
The elements of $\boldsymbol{\mu}$ and $\boldsymbol{A}$ are gathered in a matrix $\boldsymbol{\theta}$ of size $(N_s, N_s + 1)$:

$$\boldsymbol{\theta} = \begin{bmatrix} \mu_1 & a_{11} & \cdots & a_{1,N_s} \\ \vdots & \vdots & \ddots & \vdots \\ \mu_{Ns} & a_{N_s 1} & \cdots & a_{N_s N_s} \end{bmatrix}$$

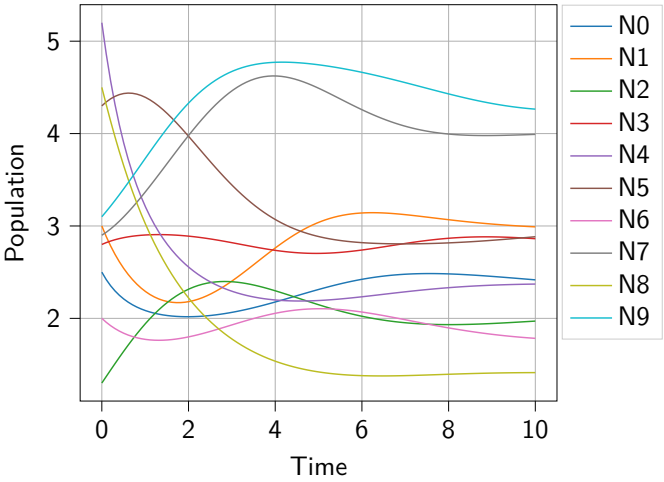
[2] V. Volterra and M. Brelot. *Leçons sur la théorie mathématique de la lutte pour la vie*. eng. Paris : Gauthier-Villars, 1931.

Example with $N_s = 3$

We set $\boldsymbol{A} = \begin{bmatrix} -2 & -5 & -0.5 \\ -0.5 & -1 & -1.2 \\ -1 & -0.5 & -1 \end{bmatrix}$, from an initial population of $\boldsymbol{u}_0 = [5, 3, 1]^T$ and the intrinsic growth rate $\boldsymbol{\mu} = [7.5, 2.6, 2.5]^T$



Example with $N_s = 10$



Framework of the study

Main objective: **determine the optimal parameters** a_{ij} and μ_i of (GLV) **from observed data** throughout multiple experiments.

Challenge:

- bacterial data has a significantly lower sample number than bacterial species
- Direct estimation of GLV model parameters, such as maximum likelihood estimation with smoothing of observation, Bayesian estimation with smoothing of observation, ..., even genetic algorithm is not easy (local minima, instability of the system in certain parameter regions).

Here we present the ***Generalised Smoothing PINN algorithm****: a mixture algorithm between* ***PINN*** *and* ***Generalised Smoothing Algorithm****.*

Generalized Smoothing Algorithm

- Introduced by Ramsay and co-authors[3]
- Method to estimate parameters $\boldsymbol{\theta}$ in a nonlinear differential equation of the form

$$\partial_t \boldsymbol{u}(t) = \boldsymbol{f}(\boldsymbol{u}, t; \boldsymbol{\theta}).$$

[3] J. O. Ramsay et al. "Parameter estimation for differential equations: a generalized smoothing approach". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 69.5 (2007), pp. 741–796.

[4] B. Laroche et al. "Parameter estimation for dynamical systems using an FDA approach". In: *11th International Conference of the ERCIM WG on Computational and Methodological Statistics (CMStatistics 2018)*. Pise, Italy, Dec. 2018.

Generalized Smoothing Algorithm

- Introduced by Ramsay and co-authors[3]
- Method to estimate parameters $\boldsymbol{\theta}$ in a nonlinear differential equation of the form

$$\partial_t \boldsymbol{u}(t) = \boldsymbol{f}(\boldsymbol{u}, t; \boldsymbol{\theta}).$$

- Previously used for the GLV model[4]

[3] J. O. Ramsay et al. "Parameter estimation for differential equations: a generalized smoothing approach". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 69.5 (2007), pp. 741–796.

[4] B. Laroche et al. "Parameter estimation for dynamical systems using an FDA approach". In: *11th International Conference of the ERCIM WG on Computational and Methodological Statistics (CMStatistics 2018)*. Pise, Italy, Dec. 2018.

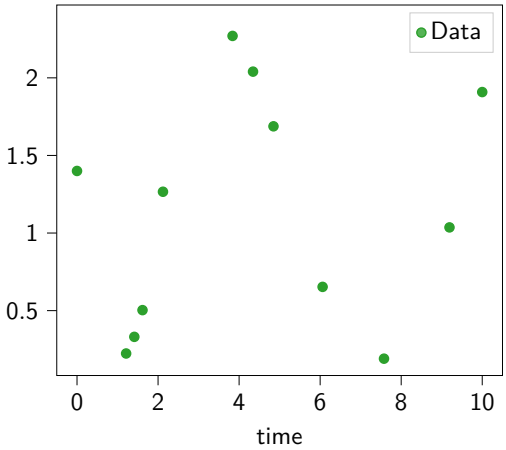
Generalized Smoothing Algorithm - Least Squares (GSA-LS)

Step 0 **Spline smoothing of the data**. The coefficients of the spline function fitting the data are stored in a matrix $\boldsymbol{C}$.

Step 1 Estimate of $\boldsymbol{\theta}$ with the proximal gradient descent technique.

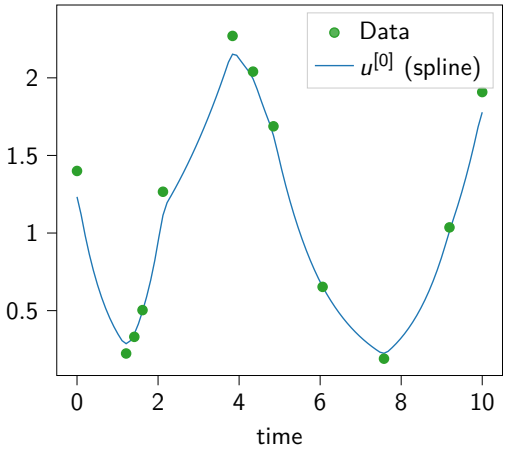
Step 2 New coefficients of the spline $\boldsymbol{C}$ basis are computed using a **least squares minimization approach**.

Generalized Smoothing Algorithm - Least Squares (GSA-LS)

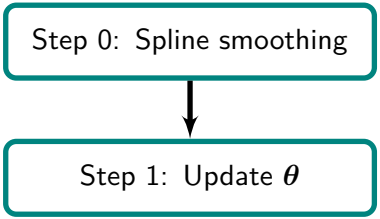


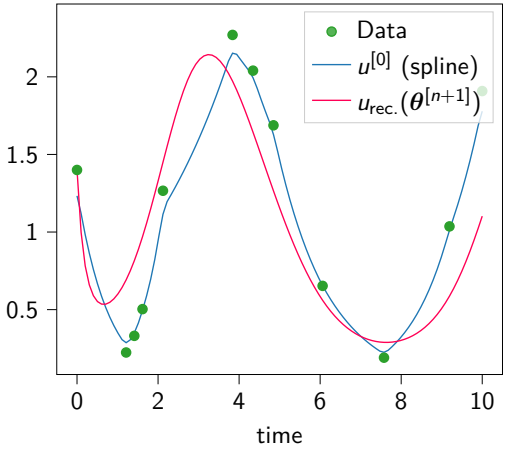
Generalized Smoothing Algorithm - Least Squares (GSA-LS)

Step 0: Spline smoothing

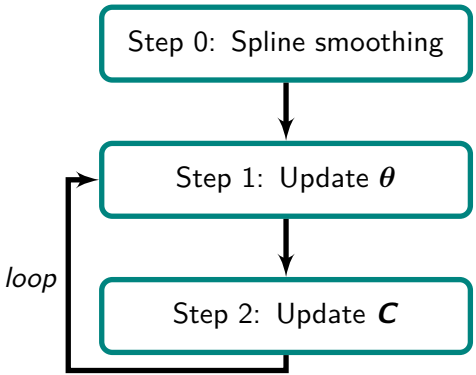


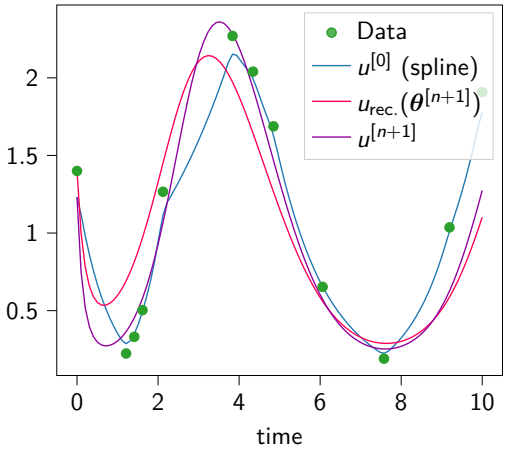
Generalized Smoothing Algorithm - Least Squares (GSA-LS)



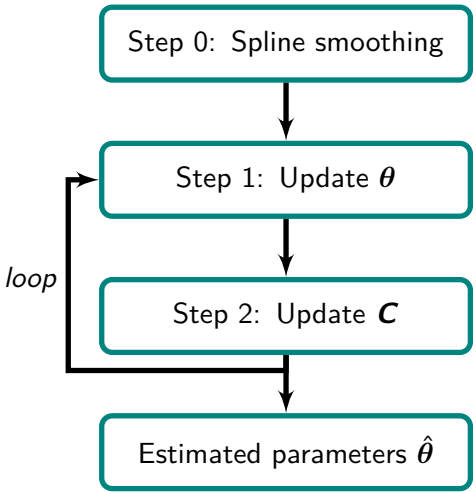


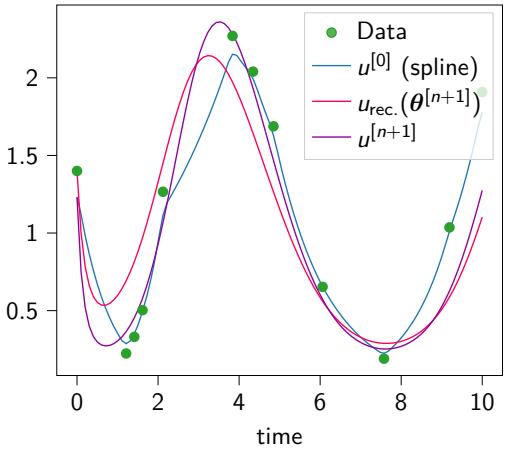
Generalized Smoothing Algorithm - Least Squares (GSA-LS)



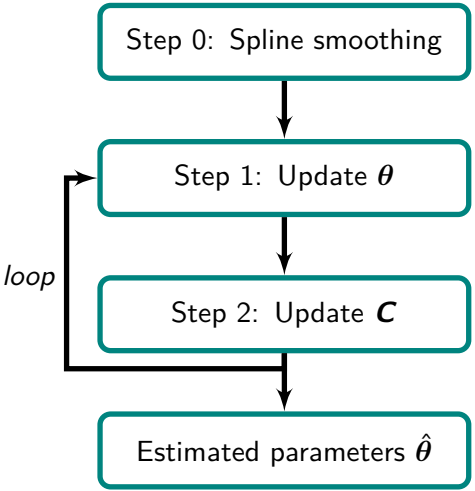


Generalized Smoothing Algorithm - Least Squares (GSA-LS)





Generalized Smoothing Algorithm - Least Squares (GSA-LS)



$$\boldsymbol{\theta}^{[n+1]} \leftarrow \arg\min_{\boldsymbol{\theta}} \left(\underbrace{J_2(\boldsymbol{C}^{[n]}, \boldsymbol{\theta})}_{\text{model}} + \lambda_1 \underbrace{J_3(\boldsymbol{\theta})}_{\text{pen.}} \right)$$

$$\boldsymbol{C}^{[n+1]} \leftarrow \arg\min_{C} \left(\underbrace{J_1(\boldsymbol{C})}_{\text{data}} + \lambda_2^{\text{LS}} \underbrace{J_2(\boldsymbol{C}, \boldsymbol{\theta}^{[n+1]})}_{\text{model}} \right)$$

- **Data:** we want to fit the data points

$$J_1(\boldsymbol{C}) = \sum_{e=1}^{N_{\text{exp}}} \sum_{k=1}^{N_{\text{obs}}^{(e)}} \sum_{i=1}^{N_s} \left| \widehat{u}_i^{(e)}(t_k^{(e)}) - \boldsymbol{U}_{i,k}^{(e)} \right|^2$$

where $\widehat{u}_i^{(e)}(t) = C^{(e)}\boldsymbol{\Phi}(t)$ is the spline reconstructed solution for the species i, and the experiment e.

- **Data:** we want to fit the data points

$$J_1(\boldsymbol{C}) = \sum_{e=1}^{N_{\text{exp}}} \sum_{k=1}^{N_{\text{obs}}^{(e)}} \sum_{i=1}^{N_s} \left| \widehat{u}_i^{(e)}(t_k^{(e)}) - \boldsymbol{U}_{i,k}^{(e)} \right|^2$$

where $\widehat{u}_i^{(e)}(t) = C^{(e)}\boldsymbol{\Phi}(t)$ is the spline reconstructed solution for the species i, and the experiment e.

- **Model:** we want to fit the dynamic

$$J_2(\boldsymbol{C}, \boldsymbol{\theta}) = \sum_{e=1}^{N_{\text{exp}}} \frac{1}{N_f} \sum_{j=1}^{N_f} \left\| \partial_t \widehat{\boldsymbol{u}}^{(e)}(t_j) - \boldsymbol{f}(\widehat{\boldsymbol{u}}^{(e)}(t_j), t_j, \boldsymbol{\theta}) \right\|_2^2$$

where $(t_j)_{j=1}^{N_f}$ is a familly of collocation points, equi-distribued over $[0, 1]$.

- **Data:** we want to fit the data points

$$J_1(\boldsymbol{C}) = \sum_{e=1}^{N_{\text{exp}}} \sum_{k=1}^{N_{\text{obs}}^{(e)}} \sum_{i=1}^{N_s} \left| \widehat{u}_i^{(e)}(t_k^{(e)}) - \boldsymbol{U}_{i,k}^{(e)} \right|^2$$

where $\widehat{u}_i^{(e)}(t) = C^{(e)}\boldsymbol{\Phi}(t)$ is the spline reconstructed solution for the species i, and the experiment e.

- **Model:** we want to fit the dynamic

$$J_2(\boldsymbol{C}, \boldsymbol{\theta}) = \sum_{e=1}^{N_{\text{exp}}} \frac{1}{N_f} \sum_{j=1}^{N_f} \left\| \partial_t \widehat{\boldsymbol{u}}^{(e)}(t_j) - \boldsymbol{f}(\widehat{\boldsymbol{u}}^{(e)}(t_j), t_j, \boldsymbol{\theta}) \right\|_2^2$$

where $(t_j)_{j=1}^{N_f}$ is a familly of collocation points, equi-distribued over $[0, 1]$.

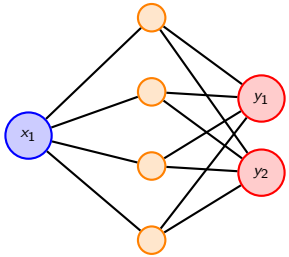
- **Penalization on $\boldsymbol{\theta}$:** we want our parameters matrix to be sparse

$$J_3(\boldsymbol{\theta}) = \text{Pen}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1$$

Advantages of GSA

- **Flexibility**: can handle functional data with complex structures,
- **Nonparametric**: no assumptions about the underlying distribution,
- Effective for **denoising** and recovering underlying patterns in data,
- Provides **interpretable and smooth estimates**,
- **Widely applicable** in diverse fields for analyzing complex functional data.

Neural Networks



- Neural Network: $NN\colon \boldsymbol{x} \in \mathbb{R}^p \mapsto \boldsymbol{y} \in \mathbb{R}^q$
- $NN(\boldsymbol{x}) = f_p \circ \sigma \circ f_{p-1} \circ \sigma \circ \cdots \circ f_1(\boldsymbol{x})$, where:
  - f_i are affine functions $f_i(\boldsymbol{x}) = \boldsymbol{W}_i \boldsymbol{x} + \boldsymbol{b}_i$
  - σ is a *non-linear activation* function (*e.g.* sigmoid, ReLu...).
- $\boldsymbol{\Theta} = (\boldsymbol{W}_1, \boldsymbol{b}_1, \ldots, \boldsymbol{W}_p, \boldsymbol{b}_p)$.

Density of neural networks[5]

The space of neural newtorks funcitons with 1 hidden layer ($p = 1$) is dense in the space of continuous functions on a compact set, for the norm $\|\cdot\|_\infty$.

[5] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314

Neural Networks: supervised learning

The Neural Network is trained with a set of **labeled data**.

(c) Cats

(d) Dogs

Neural Networks: supervised learning

The Neural Network is trained with a set of **labeled data**.

NN 

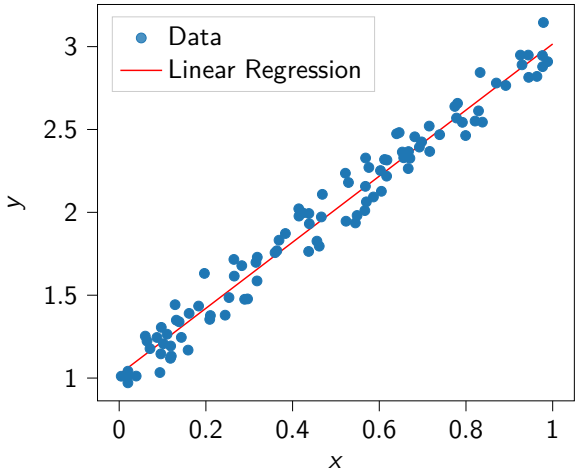
Neural Networks: supervised learning

The Neural Network is trained with a set of **labeled data**.

$$NN\left(\text{  }\right) = \text{Dog}$$

Neural Networks: unsupervised learning

We have **unlabeled data**, and we want to find a structure in it.

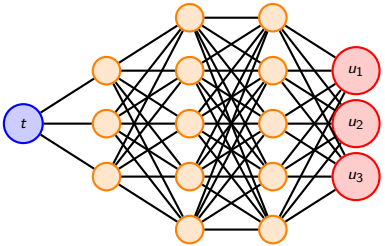


Neural Networks: training

- Set of data: $D = \{(\boldsymbol{x}_j, \boldsymbol{y}_j)\}_{j=1}^{N}$
- Loss function: $\text{Loss}(\boldsymbol{\Theta}) = \sum_{(\boldsymbol{x},\boldsymbol{y})\in D} |NN_{\boldsymbol{\Theta}}(\boldsymbol{x}) - \boldsymbol{y}|^2$
- Optimization: look for $\boldsymbol{\Theta}^\star = \arg\min_{\boldsymbol{\Theta}} \text{Loss}(\boldsymbol{\Theta})$
- **Least square theorem:** The solution exists. It is unique if the data is linearly independent.
- To « find » the solution, we use an **optimizer** like Adam.

Physics-Informed Neural Networks[6]

- Combines both unsupervised and supervised learning.
- Trained to solve learning tasks while respecting a law given here by the ODE / PDE **and** provided data.
- « Loss = $\text{Loss}_{\text{model}} + \text{Loss}_{\text{data}}$ »



Input size = 3 size = 5 size = 5 Output

[6] M. Raissi, P. Perdikaris, and G.E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

PINN to solve the GLV model for a given parameter and initial condition

We ultimately want to replace the previous *Step 2* with a **Physics-Informed Neural Network**, as it minimises proximity to data and proximity to the model.

PINN to solve the GLV model for a given parameter and initial condition

We ultimately want to replace the previous *Step 2* with a **Physics-Informed Neural Network**, as it minimises proximity to data and proximity to the model.

Let us consider a normalized version of (GLV) written as

$$\frac{\partial}{\partial t}\boldsymbol{u}(t) = t_{\max}(\boldsymbol{\mu} + \boldsymbol{A} \cdot \exp(\boldsymbol{u}(t))) \quad \text{for } t \in [0,1]. \qquad \text{(GLV-norm)}$$

PINN to solve the GLV model for a given parameter and initial condition

We ultimately want to replace the previous *Step 2* with a **Physics-Informed Neural Network**, as it minimises proximity to data and proximity to the model.

Let us consider a normalized version of (GLV) written as

$$\frac{\partial}{\partial t}\boldsymbol{u}(t) = t_{\max}(\boldsymbol{\mu} + \boldsymbol{A} \cdot \exp(\boldsymbol{u}(t))) \quad \text{for } t \in [0,1]. \qquad \text{(GLV-norm)}$$

Objective: construct a neural network approximation $\hat{\boldsymbol{u}}(t)$ of the solution $\boldsymbol{u}(t)$ of (GLV-norm) **given some parameters $\boldsymbol{\mu}$ and $\boldsymbol{A}$** (and some data points).

We will have $\hat{\boldsymbol{u}}\colon [0,1] \to \mathbb{R}^{N_s}$, meaning one neural network for each experiment.

PINN to solve the GLV model for a given parameter and initial condition

We ultimately want to replace the previous *Step 2* with a **Physics-Informed Neural Network**, as it minimises proximity to data and proximity to the model.

Let us consider a normalized version of (GLV) written as

$$\frac{\partial}{\partial t}\boldsymbol{u}(t) = t_{\max}(\boldsymbol{\mu} + \boldsymbol{A} \cdot \exp(\boldsymbol{u}(t))) \quad \text{for } t \in [0,1]. \qquad \text{(GLV-norm)}$$

Objective: construct a neural network approximation $\hat{\boldsymbol{u}}(t)$ of the solution $\boldsymbol{u}(t)$ of (GLV-norm) **given some parameters $\boldsymbol{\mu}$ and $\boldsymbol{A}$** (and some data points).

We will have $\hat{\boldsymbol{u}}\colon [0,1] \to \mathbb{R}^{N_s}$, meaning one neural network for each experiment.
Let $\mathcal{L}$ be the *residual* of the prediction $\hat{\boldsymbol{u}}(t)$ defined as:

$$\mathcal{L}(t) := \partial_t \hat{\boldsymbol{u}}(t) - t_{\max}\left(\boldsymbol{\mu} + \boldsymbol{A}\exp(\hat{\boldsymbol{u}}(t))\right) \quad \forall t \in [0,1].$$

Loss function

We introduce 2 types of errors:

- The mean squared misfit by the data:

$$MSE_{\text{data}}\left(t^{(e)}\right) = \frac{1}{N_s N_{\text{obs}}^e} \sum_{i=1}^{N_s} \sum_{k=1}^{N_{\text{obs}}^e} \left\| \hat{\boldsymbol{u}}^i(t_k^{(e)}) - \boldsymbol{U}_{i,k}^{(e)} \right\|^2$$

- The mean squared residual, with collocation points $t^{(\text{col})} = \{t_j\}_{j=1}^{N_f} \subset [0,1]$:

$$MSE_{\mathcal{L}}(t^{(\text{col})}) = \frac{1}{N_s N_f} \sum_{i=1}^{N_s} \sum_{j=1}^{N_f} \|\mathcal{L}_i(t_j)\|^2$$

Loss function

We introduce 2 types of errors:

- The mean squared misfit by the data:

$$MSE_{\text{data}}\left(t^{(e)}\right) = \frac{1}{N_s N_{\text{obs}}^e} \sum_{i=1}^{N_s} \sum_{k=1}^{N_{\text{obs}}^e} \left\| \hat{\boldsymbol{u}}^i(t_k^{(e)}) - \boldsymbol{U}_{i,k}^{(e)} \right\|^2$$

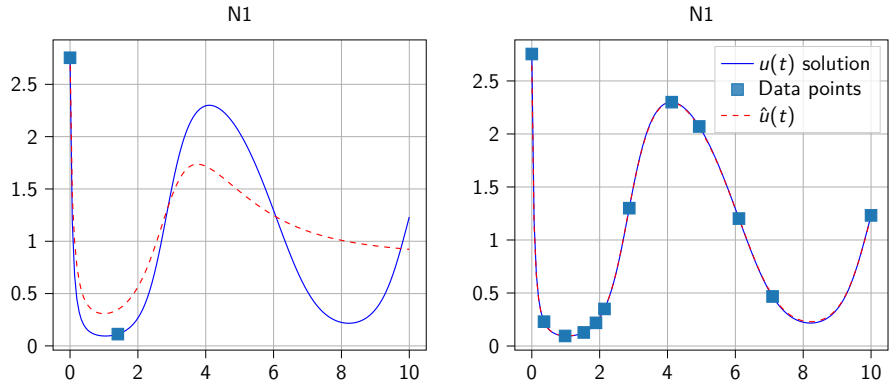
- The mean squared residual, with collocation points $t^{(\text{col})} = \{t_j\}_{j=1}^{N_f} \subset [0,1]$:

$$MSE_{\mathcal{L}}(t^{(\text{col})}) = \frac{1}{N_s N_f} \sum_{i=1}^{N_s} \sum_{j=1}^{N_f} \|\mathcal{L}_i(t_j)\|^2$$

Target loss to be minimized, involving hyper-parameters $\lambda_2^{\text{PINN}} > 0$:

$$\boxed{\text{Loss} = MSE_{\text{data}}(t^{(e)}) + \lambda_2^{\text{PINN}} MSE_{\mathcal{L}}(t_r)}$$

PINN prediction

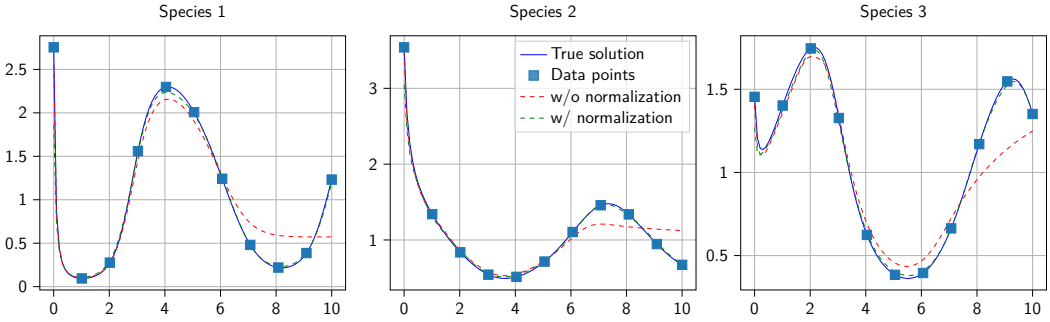


(e) With 2 data points

(f) With 12 data points

Figure 1: Prediction of the PINN with various numbers of points used for the training set.

Normalisation of the time



Tuning the hyperparameters

Hyperparameters of interest:

- λ_2^{PINN}
- architecture (number of layers, size of layers)

Tuning the hyperparameters

Hyperparameters of interest:

- λ_2^{PINN}
- architecture (number of layers, size of layers)

Tuning with Optuna: an open source hyperparameter optimization framework, with the objective of minimizing:

$$E_{\text{PINN}} = \frac{1}{N_s} \sum_{j=1}^{N_s} \frac{\|\hat{\boldsymbol{u}}^j - \boldsymbol{u}^j_{\text{truth}}\|^2_{L^2([0,t_{\max}])}}{\|\boldsymbol{u}^j_{\text{truth}}\|^2_{L^2([0,t_{\max}])}}$$

Tuning the hyperparameters

Hyperparameters of interest:

- λ_2^{PINN}
- architecture (number of layers, size of layers)

Tuning with Optuna: an open source hyperparameter optimization framework, with the objective of minimizing:

$$E_{\text{PINN}} = \frac{1}{N_s} \sum_{j=1}^{N_s} \frac{\|\hat{\boldsymbol{u}}^j - \boldsymbol{u}^j_{\text{truth}}\|^2_{L^2([0,t_{\max}])}}{\|\boldsymbol{u}^j_{\text{truth}}\|^2_{L^2([0,t_{\max}])}}$$

- $\lambda_2^{\text{PINN}} = 10^{-3}$
- best architecture is $[1, N_s, 7 \cdot N_s, 7 \cdot N_s, N_s]$

Regarding the architecture:

- compromise between speed of training and precision

GSA with a PINN

Step 0: $\boldsymbol{u}^{[0]}$ $\leftarrow$ spline smoothing of data
$+$ « loop 0 »: First training of the PINN

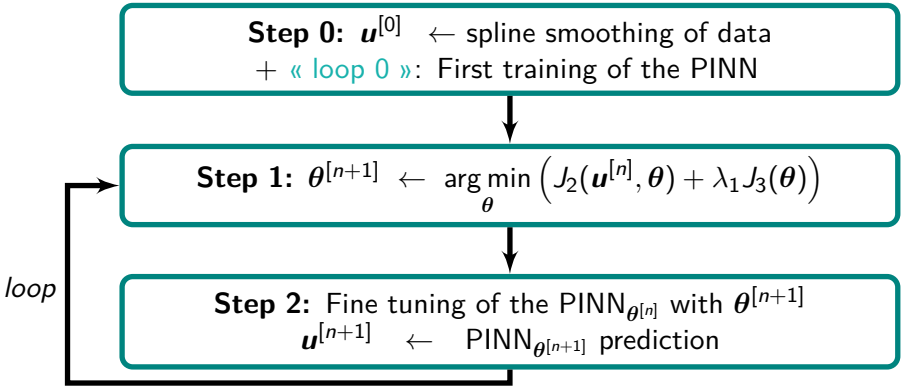
GSA with a PINN

Step 0: $\boldsymbol{u}^{[0]} \leftarrow$ spline smoothing of data
$+$ « loop 0 »: First training of the PINN

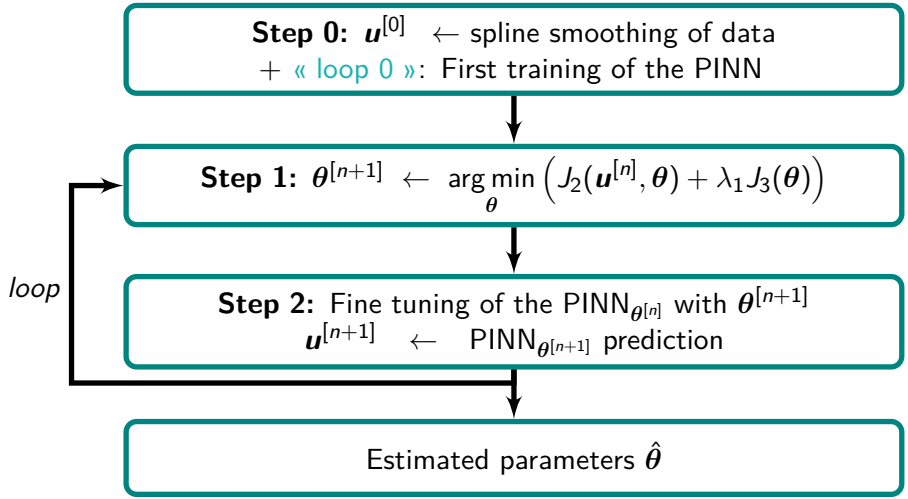
$\downarrow$

Step 1: $\boldsymbol{\theta}^{[n+1]} \leftarrow \underset{\boldsymbol{\theta}}{\arg\min} \left(J_2(\boldsymbol{u}^{[n]}, \boldsymbol{\theta}) + \lambda_1 J_3(\boldsymbol{\theta}) \right)$

GSA with a PINN



GSA with a PINN



Epoch management

How many epochs should we do for the PINN?
We have to do a trade-off between precision and computation time.
Hyperparameters tuning methods no so helpful as they appeared to be very problem dependent.

Chose an **adaptative method**: do fewer epochs if there is a smaller change in the estimated parameters.

- do k epochs, with $k = \min\left\{1 + \left\lfloor 10^3 \cdot \frac{\|\boldsymbol{\theta}^{[n]} - \boldsymbol{\theta}^{[n+1]}\|_F}{\|\boldsymbol{\theta}^{[n]}\|_F} \right\rfloor, 200\right\}$
- stop if Loss $\leq 10^{-3}$

Stop criterion

- We use a relative error between two consecutive iterations to stop the algorithm:

$$err^{[n]} = \frac{\|\boldsymbol{u}^{[n]} - \boldsymbol{u}^{[n+1]}\|_{L^2[0,t_{\max}]}}{\|\boldsymbol{u}^{[n]}\|_{L^2[0,t_{\max}]}} + \frac{\|\boldsymbol{\theta}^{[n]} - \boldsymbol{\theta}^{[n+1]}\|_{\mathrm{F}}}{\|\boldsymbol{\theta}^{[n]}\|_{\mathrm{F}}}$$

- We stop when $err^{[n]}$ reaches a given tolerance `errMax`,
- We also stop if the number of iterations reaches a maximal number of iterations `maxIter`.

Stop criterion

- We use a relative error between two consecutive iterations to stop the algorithm:

$$err^{[n]} = \frac{\|\boldsymbol{u}^{[n]} - \boldsymbol{u}^{[n+1]}\|_{L^2[0,t_{\max}]}}{\|\boldsymbol{u}^{[n]}\|_{L^2[0,t_{\max}]}} + \frac{\|\boldsymbol{\theta}^{[n]} - \boldsymbol{\theta}^{[n+1]}\|_{\mathrm{F}}}{\|\boldsymbol{\theta}^{[n]}\|_{\mathrm{F}}}$$

- We stop when $err^{[n]}$ reaches a given tolerance `errMax`,
- We also stop if the number of iterations reaches a maximal number of iterations `maxIter`.
- But $err^{[n]}$ decreasing really slowly,
- Adaptative stop criterion: every 30 steps, if we have not improved the minimal error, we multiply the tolerance by 10.

Stop criterion

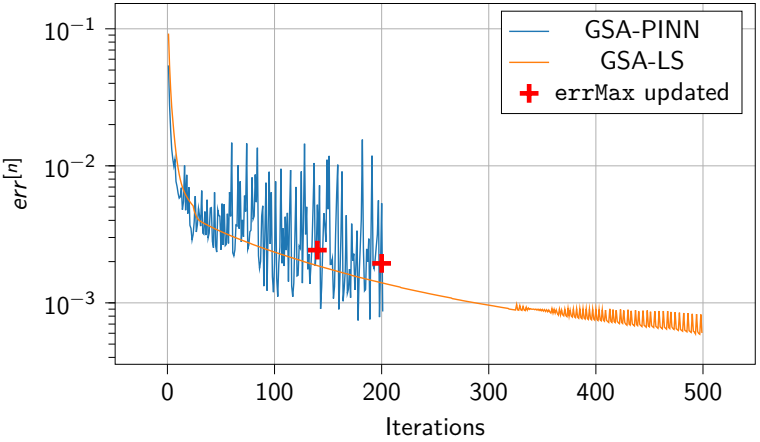
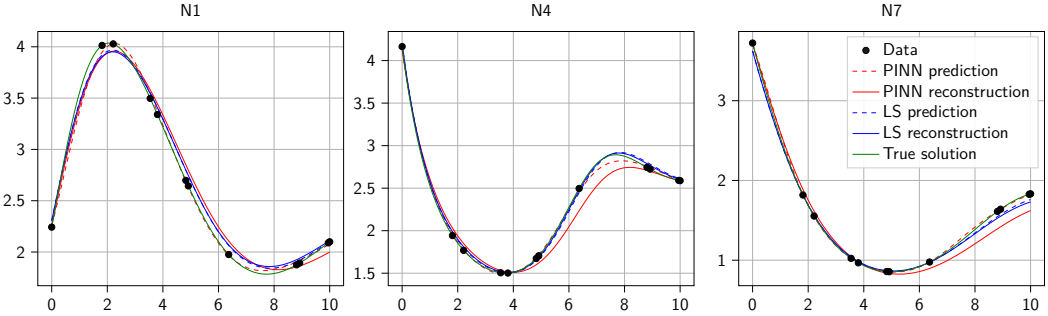


Figure 2: Evolution of the error in the GSA-PINN algorithm, test case with 3 populations.

First comparison: 10 experiments for 10 species are performed

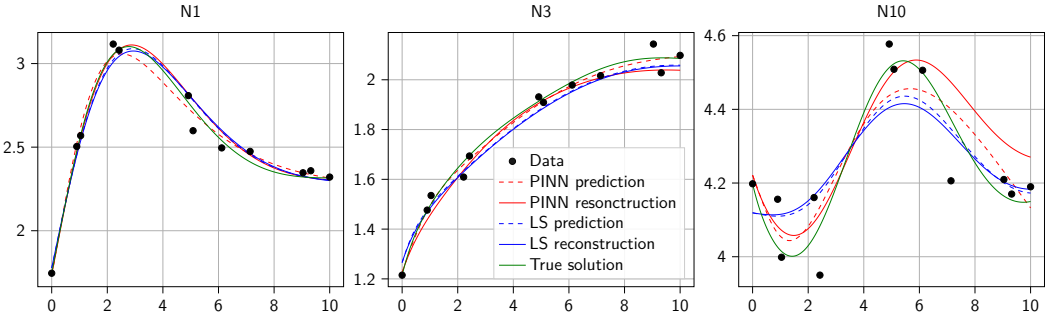
We use data manually generated from a known set of parameters.



(a) Data without noise

First comparison: 10 experiments for 10 species are performed

We use data manually generated from a known set of parameters.



(b) Noisy data

First comparison: comparison between the two algorithms

- $Err_{\theta,1} := \dfrac{\|\hat{\theta} - \theta_{\text{truth}}\|_F}{\|\theta_{\text{truth}}\|_F}$,
- $Err_{\theta,2}$ defined as the number of coefficients where $\hat{\theta}$ and θ_{truth} have the same sign, divided by the number of coefficients of the matrices.
- $Err_{u,1} := \dfrac{1}{N_s N_{\text{exp}}} \sum_{i=1}^{N_s} \sum_{e=1}^{N_{\text{exp}}} \dfrac{\|\hat{u}_j^{(e)} - u_j^{(e)}(\theta_{\text{truth}})\|_2^2}{\|u_j^{(e)}(\theta_{\text{truth}})\|_2^2}$,
- $Err_{u,2} := \dfrac{1}{N_s N_{\text{exp}}} \sum_{i=1}^{N_s} \sum_{e=1}^{N_{\text{exp}}} \dfrac{\|u_j^{(e)}(\hat{\theta}) - u_j^{(e)}(\theta_{\text{truth}})\|_2^2}{\|u_j^{(e)}(\theta_{\text{truth}})\|_2^2}$,

Comparison between Matlab original code and our Python code

Uniformly distributed data (non-random), 10 data, 10 species, 1 experiment, no noise

Algo.	Mean $Err_{\boldsymbol{\theta},1}$	Mean $Err_{\boldsymbol{\theta},2}$	Mean $Err_{\boldsymbol{u},1}$	Mean $Err_{\boldsymbol{u},2}$	Elapsed time
GSA-LS	1.02	0.69	$4.73 \cdot 10^{-2}$	$4.94 \cdot 10^{-2}$	5.54 sec
GSA-PINN	1.06	0.6	$1.62 \cdot 10^{-2}$	$2.2 \cdot 10^{-2}$	4.46 sec

Comparison between Matlab original code and our Python code

Uniformly distributed data (non-random), 10 data, 20 species, 1 experiment, no noise

Algo.	Mean $Err_{\boldsymbol{\theta},1}$	Mean $Err_{\boldsymbol{\theta},2}$	Mean $Err_{\boldsymbol{u},1}$	Mean $Err_{\boldsymbol{u},2}$	Elapsed time
GSA-LS	1.09	0.8	$3.4 \cdot 10^{-2}$	$3.53 \cdot 10^{-2}$	8.75 sec
GSA-PINN	0.98	0.74	$1.35 \cdot 10^{-2}$	$1.92 \cdot 10^{-2}$	10.48 sec

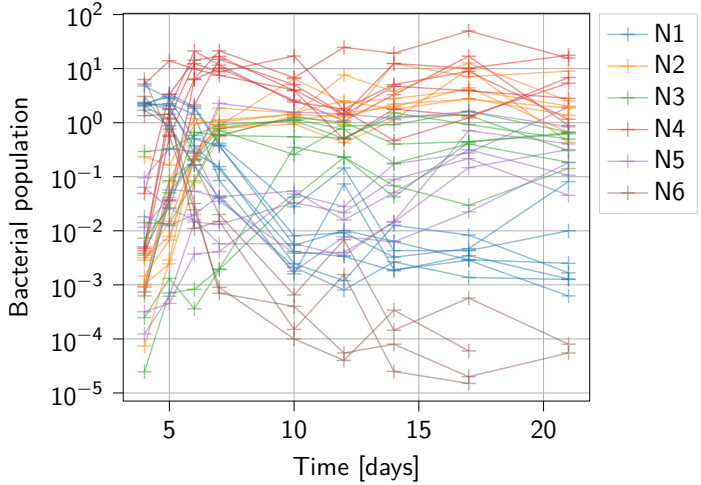
Comparison between Matlab original code and our Python code

Uniformly distributed data (non-random), 10 data, 10 species, 10 experiments, no noise

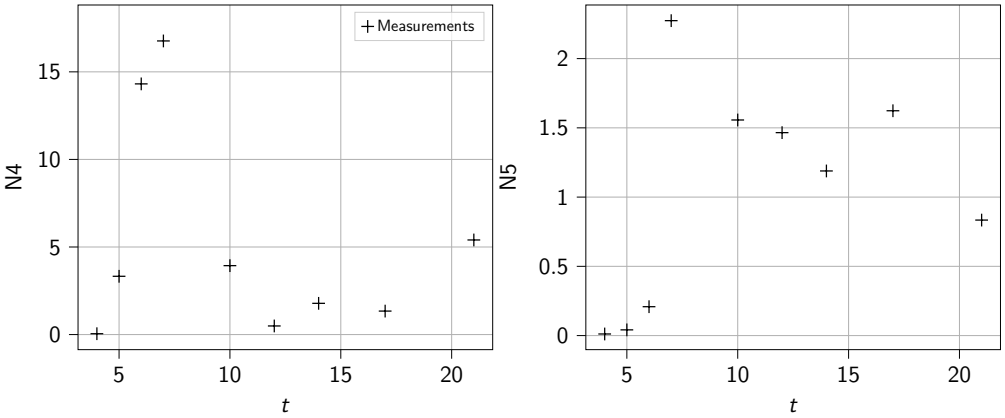
Algo.	Mean $Err_{\boldsymbol{\theta},1}$	Mean $Err_{\boldsymbol{\theta},2}$	Mean $Err_{\boldsymbol{u},1}$	Mean $Err_{\boldsymbol{u},2}$	Elapsed time
GSA-LS	0.19	0.21	0.32	0.26	16.81 sec
GSA-PINN	0.22	0.18	$1.5 \cdot 10^{-2}$	$2.18 \cdot 10^{-2}$	35.53 sec

Bacterial population in mice guts

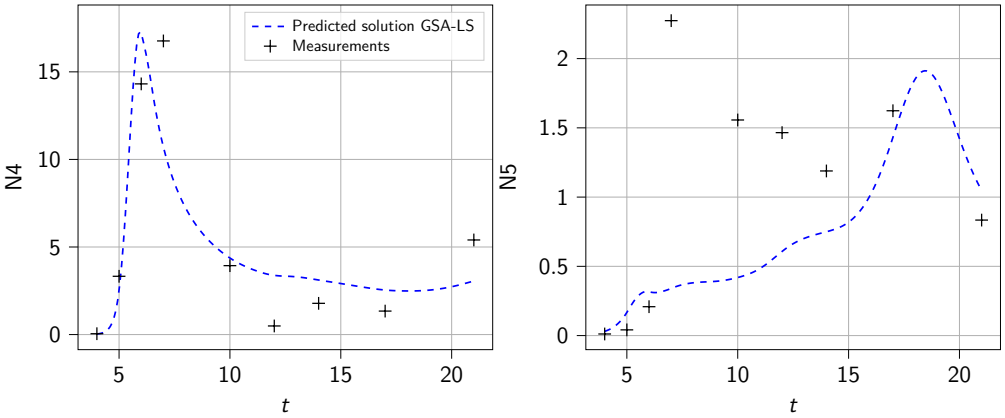
7 experiments were performed, to measure 6 various species



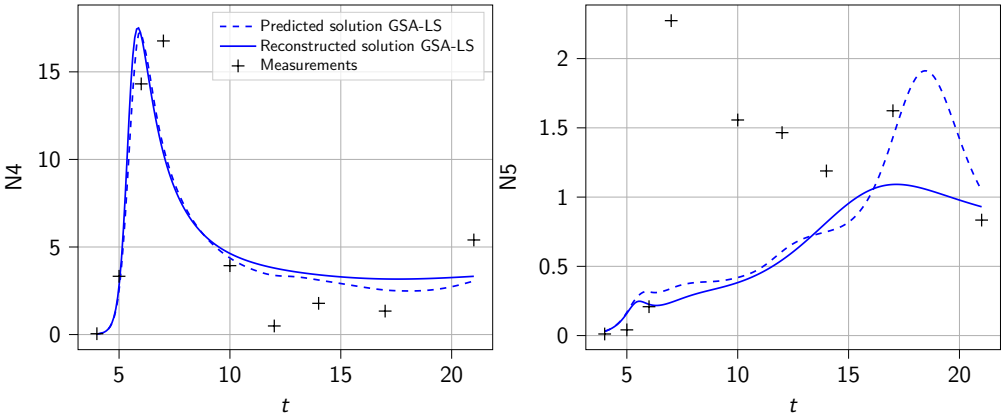
Results of the GSA algorithms



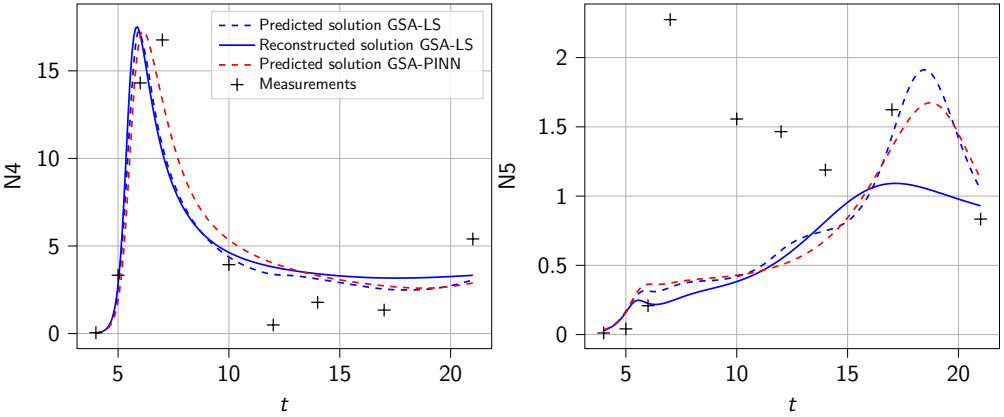
Results of the GSA algorithms



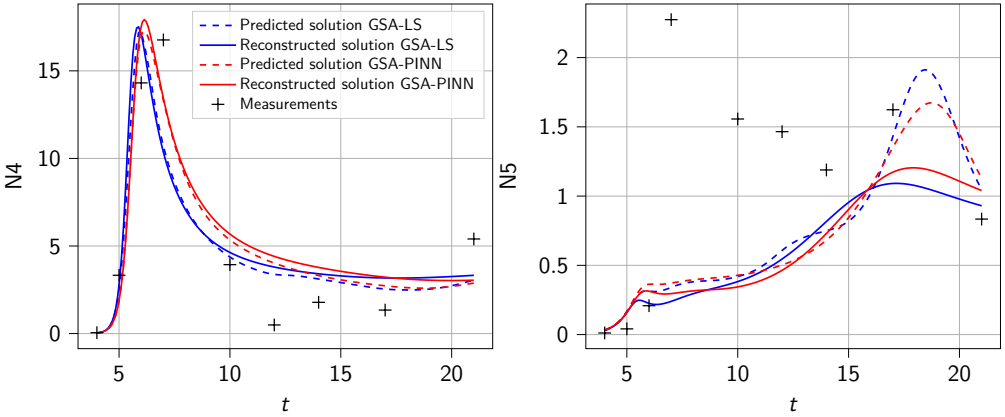
Results of the GSA algorithms



Results of the GSA algorithms



Results of the GSA algorithms



Conclusion and outlooks

- Inferring interaction coefficients from noisy data for the GLV model is a difficult question.
- Our approach gives similar results as the previous one, but it can be quicker in certain cases.

Conclusion and outlooks

- Inferring interaction coefficients from noisy data for the GLV model is a difficult question.
- Our approach gives similar results as the previous one, but it can be quicker in certain cases.

Outlooks:

- Other approaches of the Machine-Learning:
  - Have a unique PINN for all experiments ($\star$)
  - ($\star$) + trained offline so it only has to predict during the alternate minimization
  - Study a PINN for the first step or « Last-step PINN »
- Tests on « almost real » simulated data

Conclusion and outlooks

- Inferring interaction coefficients from noisy data for the GLV model is a difficult question.
- Our approach gives similar results as the previous one, but it can be quicker in certain cases.

Outlooks:

- Other approaches of the Machine-Learning:
  - Have a unique PINN for all experiments ($\star$)
  - ($\star$) + trained offline so it only has to predict during the alternate minimization
  - Study a PINN for the first step or « Last-step PINN »
- Tests on « almost real » simulated data

Thanks for your attention!

References I

[Aki+19] Takuya Akiba et al. *Optuna: A Next-generation Hyperparameter Optimization Framework*. 2019.

[Cyb89] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314.

[Lar+18] B. Laroche et al. "Parameter estimation for dynamical systems using an FDA approach". In: *11th International Conference of the ERCIM WG on Computational and Methodological Statistics (CMStatistics 2018)*. Pise, Italy, Dec. 2018.

[Ram+07] J. O. Ramsay et al. "Parameter estimation for differential equations: a generalized smoothing approach". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 69.5 (2007), pp. 741–796.

References II

[RPK19] M. Raissi, P. Perdikaris, and G.E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

[VB31] V. Volterra and M. Brelot. *Leçons sur la théorie mathématique de la lutte pour la vie*. eng. Paris : Gauthier-Villars, 1931.