PAUL SCHERRER INSTITUT

PSI

Romana Boiger :: Laboratory of Waste Management :: Paul Scherrer Institut

Solving the Bateman Equation using Physics Informed Neural Networks

Guglielmo Pacifico, Arnau Alba, Andreas Adelmann

PINN-PAD: Physics Informed Neural Networks in PADova, February 2024

Bateman Equation

$$\frac{d\vec{N}(t)}{dt} = \mathbb{A}\vec{N}(t), \quad \text{with} \quad \vec{N}(t=0) = \overrightarrow{N_0}$$

Where:

- $\vec{N}(t) \in \mathbb{R}^n$ is the nuclide concentration vector
- $\mathbb{A} \in \mathbb{R}^{n \times n}$ is the transmutation matrix
- $n \in \mathbb{N}^+$ is the number of nuclides
- $t \in \mathbb{R}$ the time

Mathematical model, that describes the
abundances and **activities** in **decay** chains of **radioactive isotopes**

Applications: Bateman Equation

$$\frac{d\vec{N}(t)}{dt} = \mathbb{A}\vec{N}(t), \quad \text{with} \quad \vec{N}(t=0) = \overrightarrow{N_0}$$

- Nuclear Physics: **nuclear depletion codes** – predict the **behaviour** of isotopes during reactor operation and fuel depletion
- Radiochemistry: study the **kinetics** of radioactive decay
- Nuclear Medicine: **medical imaging** and **therapy** using **radioactive isotopes** – model the decay of isotopes injected into the body and predict concentration at specific times
- Radiation Protection and Environmental Monitoring: predict the **behaviour** of **radioactive isotopes released** into the environment from nuclear accidents, nuclear waste disposal sites, or industrial processes

Bateman Equation

$$\frac{d\vec{N}(t)}{dt} = \mathbb{A}\vec{N}(t), \quad \text{with} \quad \vec{N}(t=0) = \overrightarrow{N_0}$$

- $\vec{N}(t) \in \mathbb{R}^n$ is the nuclide concentration vector
- $\mathbb{A} \in \mathbb{R}^{n \times n}$ is the transmutation matrix
- $n \in \mathbb{N}^+$ is the number of nuclides
- $t \in \mathbb{R}$ the time

- Formulated by Ernest Rutherford in 1905
- Analytic solution by Harry Bateman (involving Laplace transform) in 1910:

$$\vec{N}_n(t) = \vec{N}_1(0) \times \left(\prod_{i=1}^{n-1} \lambda_i\right) \times \sum_{i=1}^{n} \frac{e^{-\lambda_i t}}{\prod_{j=1, j\neq i}^{n} (\lambda_j - \lambda_i)}$$

- **Computational errors, slow if n gets bigger** => numerical methods for general case

Bateman Equation

$$\frac{d\vec{N}(t)}{dt} = \mathbb{A}\vec{N}(t)\,, \quad \text{with} \quad \vec{N}(t=0) = \overrightarrow{N_0}$$

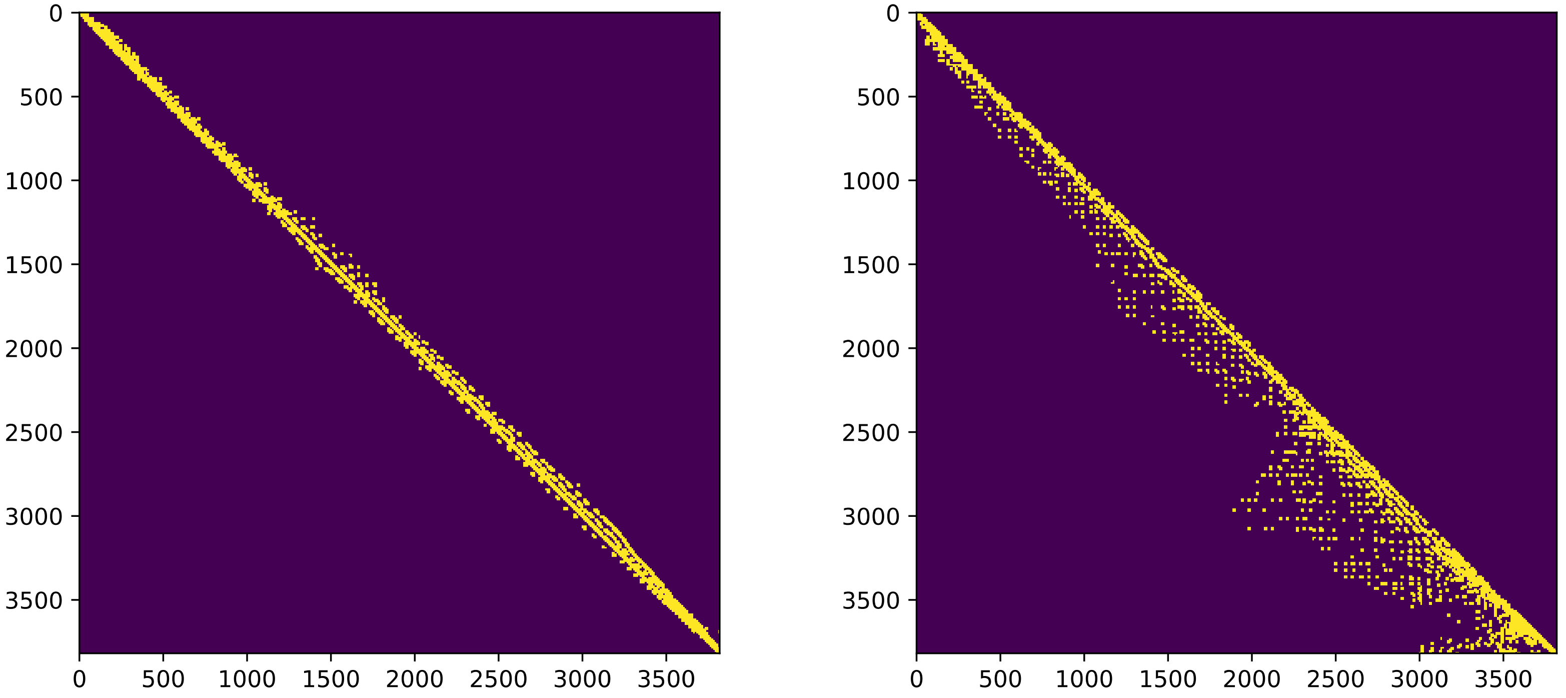
Transmutation matrix $\mathbb{A}$:

- Is **sparse**: most of its elements are zero
- Is **stiff** where: $\kappa(\mathbb{A}) := \frac{|Re(\lambda_{max})|}{|Re(\lambda_{min})|}$, where $\lambda_{max} \geq \lambda_i \geq \lambda_{min}$
- The order of rows and columns is arbitrary, given a permutation matrix P

$$\frac{d\vec{N}(t)}{dt} = (P\mathbb{A}P)\vec{N}\,, \quad \text{with} \quad \vec{N}(t=0) = P\overrightarrow{N_0}$$

Bateman Equation

Decay matrix → can be permuted into a lower triangular matrix



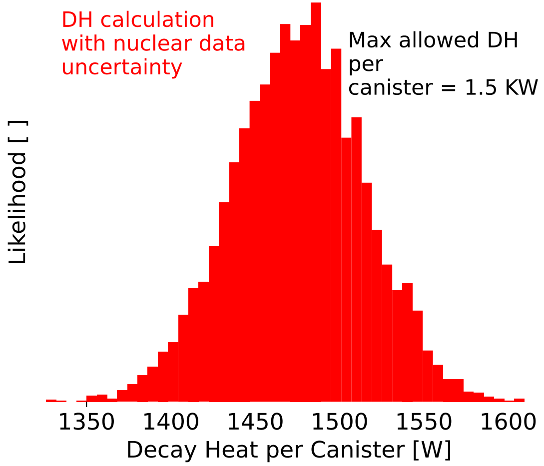
Uncertainty Quantification

- Both $\overrightarrow{N_0}$ & $\mathbb{A}$ have intrinsic uncertainties
- We want to evaluate the propagation of the uncertainties

$$\overrightarrow{N_0} \pm \Delta\overrightarrow{N_0} \to \vec{N} \pm \Delta\vec{N}$$

$$\mathbb{A} \pm \Delta\mathbb{A} \to \vec{N} \pm \Delta\vec{N}$$

- Use Monte Carlo method & transfer learning



Solving the Bateman Equation

$$\frac{d\vec{N}(t)}{dt} = \mathbb{A}\vec{N}(t), \quad \text{with} \quad \vec{N}(t=0) = \overrightarrow{N_0}$$

$$\vec{N}(t) = e^{\mathbb{A}t}\,\overrightarrow{N_0}$$

Analytic Solution of Linear Chains

Evaluating the Matrix Exponentials

- Padé approximation
  $R(x) = \frac{\sum_{j=0}^{m} a_j x^j}{1+\sum_{k=1}^{n} b_k x^k}$
- Chebyshev Rational Approximation Method (CRAM) $r_{k,k}(x) = \frac{p_k(x)}{q_k(x)}$, satisfying $\inf_{r_{k,k}\in\pi_{k,k}}\{|r_{k,k}(x) - e^x|\}$

Numerical Methods for ODEs

- e.g. Runge-Kutta

=> PINNs (physics informed neural networks) for solving the Bateman Equation

Physics Informed Neural Networks

Theorem:[1] *Let $\vec{N}\,(t)$ be a continuous function, $NN(t;\ \theta)$ a neural network parameterized by θ, and ϵ a fixed error greater than zero, then:*

$$\forall \vec{N} \in C^0, \forall \epsilon > 0, \exists \theta\colon \left\| \vec{N}(t) - NN(t;\theta) \right\| < \epsilon$$

- Physics Informed Neural Networks (PINNs) are NNs designed to solve differential equations.

$$\frac{d\vec{N}(t)}{dt} = \mathbb{A}\vec{N}(t)\,, \quad \text{with} \quad \vec{N}(t=0) = \overrightarrow{N_0}$$

- To achieve this, we embed the Bateman equation in the loss function

- $\mathcal{L}(\theta) = \mathcal{L}_{physics}(\theta) + \mathcal{L}_{initial}(\theta) = \frac{1}{T}\sum_{i=0}^{T} \left\| \frac{dNN(t_i;\theta)}{dt} - \mathbb{A}NN(t_i;\theta) \right\|_2^2 + \left\| NN(t=0;\theta) - \vec{N}_0 \right\|_2^2$

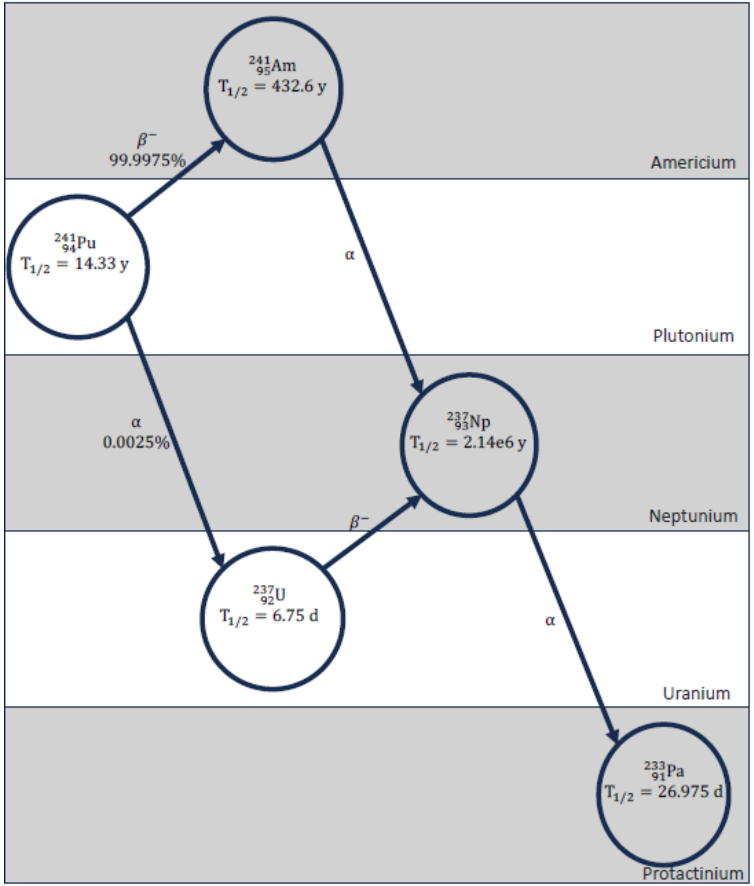
1: Hornik, Stinchcombe, White 1998

Plutonium Decay

- $\mathbb{A} = 5 \times 5$ matrix for the decay for Plutonium-241
- $\kappa(\mathbb{A}) = 9 \times 10^9$
- Solve it for 128 days
- Compare the solution with CRAM over 10^4 time steps
- Use L_2 as our metric: $L_2 = \frac{1}{5}\sum_{i=1}^{n} L_2^{N_i}$

$$L_2^{N_i} = \frac{1}{10^4}\sum_{j=1}^{10^4}\left[N_i^{CRAM}(t_j) - N_i^{PINN}(t_j)\right]^2 \times 100$$

$$\mathbb{A} = \begin{pmatrix} -2.2e{-9} & 0 & 0 & 0 & 0 \\ 5.4e{-14} & -1.7e{-6} & 0 & 0 & 0 \\ 2.2e{-9} & 0 & -7.3e{-11} & 0 & 0 \\ 0 & 1.7e{-6} & 7.3e{-11} & -1.5e{-14} & 0 \\ 0 & 0 & 0 & 1.5e{-14} & -4.3e{-7} \end{pmatrix}$$



PAUL SCHERRER INSTITUT PSI

PINN Approach: Vanilla Method

- Basic implementation of PINNs[1]

Key concept: loss function (weighted sum of the two loss terms)

$$\mathcal{L}(\theta) = \frac{w_{physics}\mathcal{L}_{physics}(\theta) + w_{initial}\mathcal{L}_{initial}(\theta)}{w_{physics} + w_{initial}}$$

- The loss scales with the weights → transformation to simplify the tuning of the weights can be used

$$\mathcal{L}(\theta) = \frac{\mathcal{L}_{physics}(\theta) + w\mathcal{L}_{initial}(\theta)}{1 + w} \quad \text{where} \quad w = \frac{w_{initial}}{w_{physics}}$$

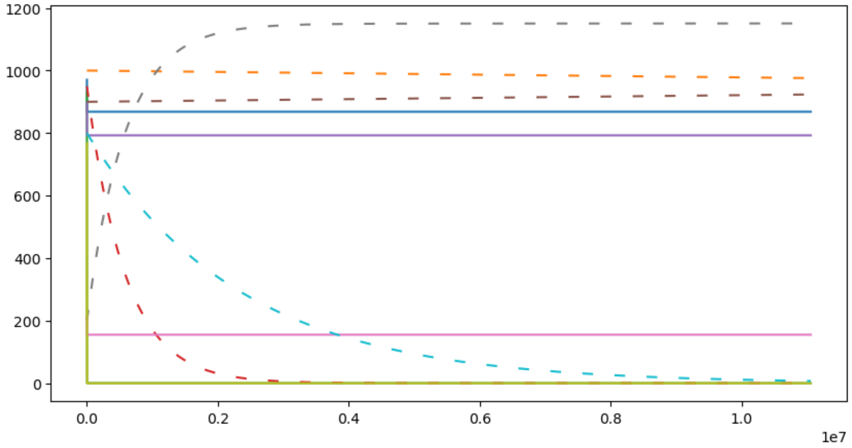
1: Raissi, Perdikaris, Karniadakis 2019

Comparison of the PINN Methods

Use Optuna to do the hyperparameters search

	Training time (s)	$L_2(\%)$
Vanilla	1528.28	62.23

High training time, high L2 error and manual weight tuning!



CRAM – takes 13 seconds for 10000 time steps for 128 days

Hard Boundary Method

Problem with Vanilla loss: manual weight tuning

Solution: Rewrite constrained problem as un unconstrained one

Two different ansatz to unconstrain the problem:

Defined by Lagaris, Likas, Fotiadis 1998:

$$\vec{\psi}(t) \coloneqq \overrightarrow{N_0} + t\overrightarrow{NN}(t;\theta)$$

Theory of Functional Connections (TFC) by Mortari 2017, Mortari, Johnston and Smith 2019

$$\vec{\psi} \coloneqq \overrightarrow{NN}(t;\theta) + \overrightarrow{N_0} - \overrightarrow{NN}(t=0;\theta)$$

$$\Rightarrow \frac{d\vec{\psi}(t)}{dt} = \mathbb{A}\vec{\psi}(t)$$

$$\Rightarrow \mathcal{L} = \mathcal{L}_{physics} = \frac{1}{T}\sum_{i=0}^{T}\left\|\frac{d\vec{\psi}(t_i;\theta)}{dt} - \mathbb{A}\vec{\psi}(t_i;\theta)\right\|_2^2$$

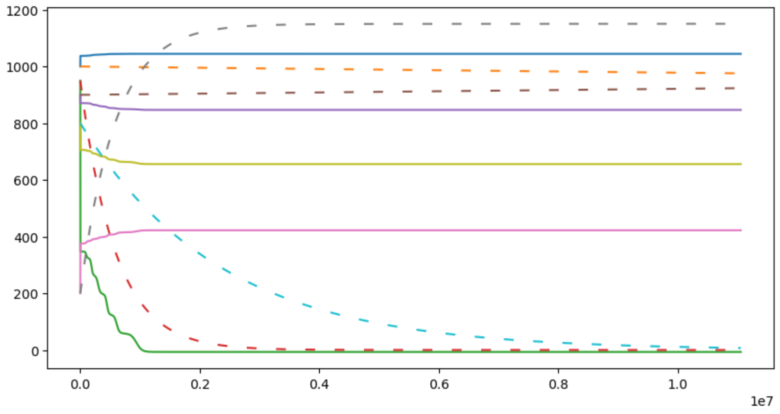
Comparison of the PINN Methods

Use Optuna to do the hyperparameters search

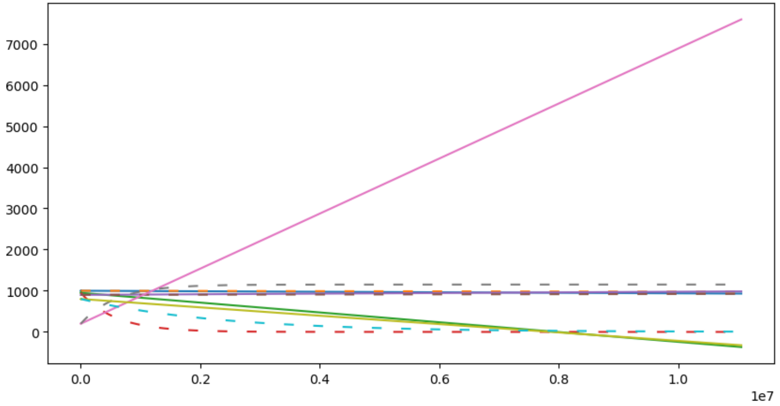
	Training time (s)	$L_2(\%)$
Vanilla	1528.28	62.23
HB-Lagaris	5084.54	131.67
HB-TFC	3218.89	68.52

High training time, high L2 error!

HB-TFC



HB-Lagaris



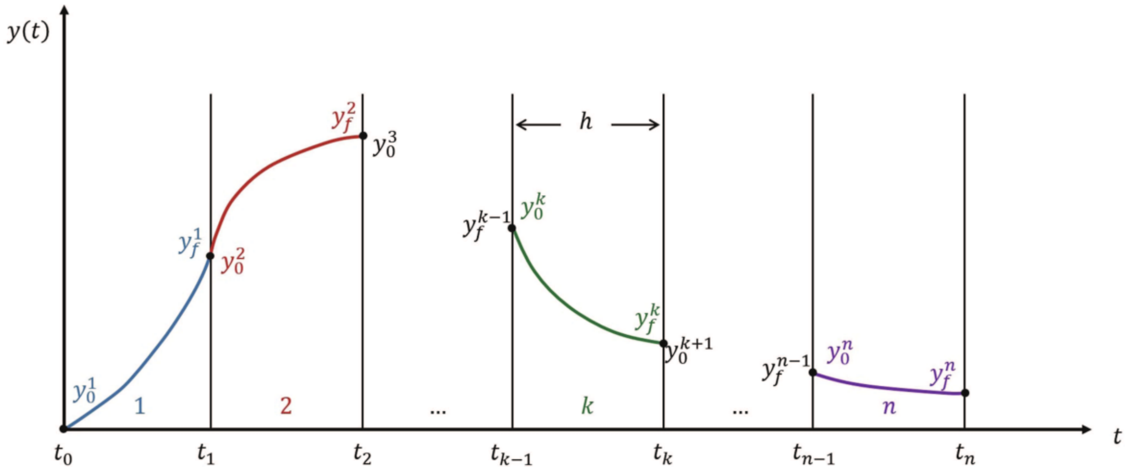
CRAM – takes 13 seconds for 10000 time steps for 128 days

Domain Decomposition Method

Problem: Long timescale – stiffness?

Key idea:

- Divide and conquer, like in Moseley, Markham, Nissen-Meyer 2023
- Use the output of the previous sub-domain as the initial condition for the next one, DeFlorio, Schiassi, Furfaro 2022
- Use transfer-learning from the previous sub-domain to the next one



Extending the Limit with Domain Decomposition

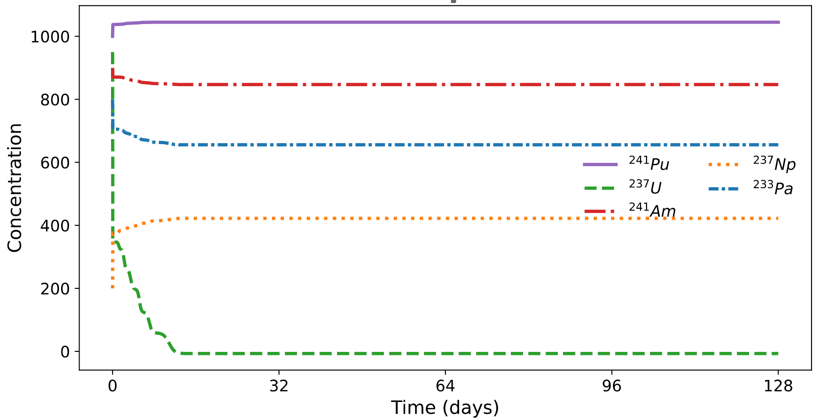
- Combine the HB-TFC with Domain Decomposition to solve the 128 days problem

Hyperparameters	
Num of sub-domains	1106
Length sub-domain	2hr 46min
Neurons per sub-domain	144
Time steps per sub-domain	75552

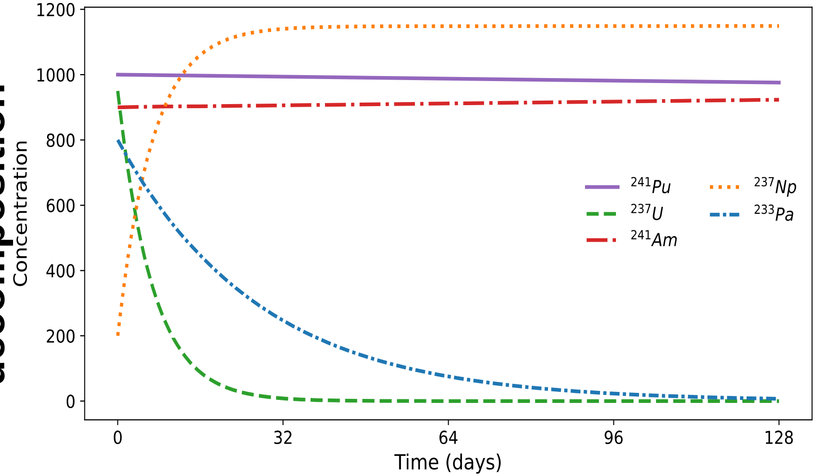
Results	
training time 1° sub-domain	39min 6s
average training time [2, 1106] sub-domains	15.6 *s*
L_2	0.1945 %

5hr

HB-TFC



HB-TFC with domain decomposition



PAUL SCHERRER INSTITUT PSI

Extreme Learning Machine Method

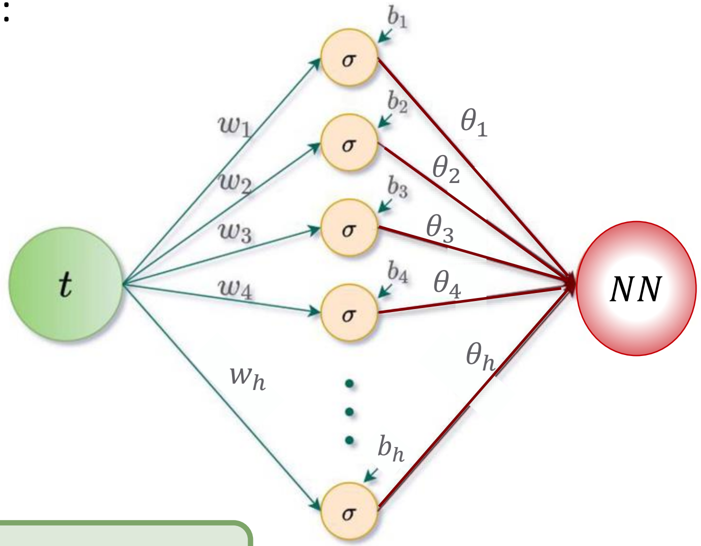
Problem with Vanilla loss and Hard boundary loss: NN training challenge and effort

Solution: Huang, Zhu, Siew 2006: Extreme learning machine:

- Use single layer NN with random input weights and biases
- Compute output weights from closed-form solution

$$w_j \in U(-a\,,a),\ \ b_j \in U(-c,c)$$

$$for\ \ j = 1,2,\ldots,h\ given\ a,c \in \mathbb{R}^+$$



$$\Rightarrow \overrightarrow{NN}(t;\vec{\theta}) = \sum_{j=1}^{h} \theta_j \sigma(w_j t + b_j) = \vec{\sigma} \cdot \vec{\theta}$$

$$\boxed{\mathcal{L}(\vec{\theta}) = \frac{1}{T}\sum_{i=0}^{T} \left\| \frac{d\vec{\sigma}(t_i)}{dt} - \mathbb{A}\left[\vec{\sigma}(t_i)\cdot\vec{\theta} - \vec{\sigma}(t=0)\cdot\vec{\theta} + \overrightarrow{N_0}\right] \right\|_2^2}$$

Solve the linear system $\Longrightarrow \mathcal{J}\vec{\theta} = -\vec{\mathcal{L}}(\vec{0})$

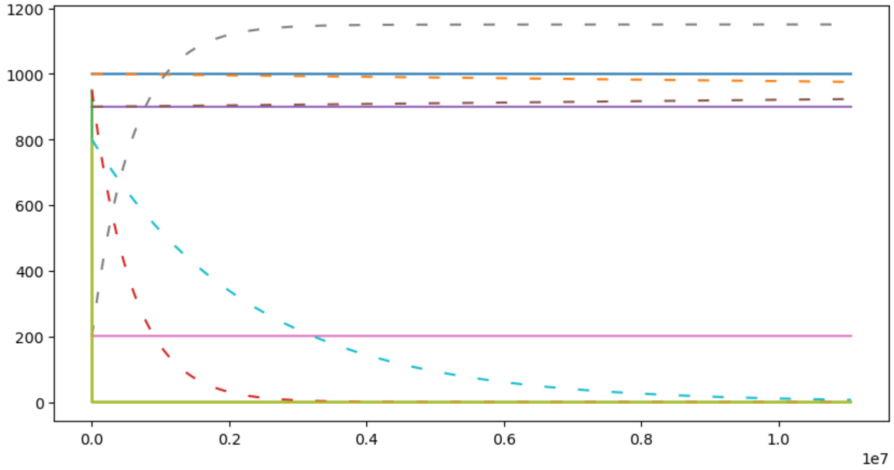
Comparison of the PINN Methods

Use Optuna to do the hyperparameters search

	Training time (s)	L_2(%)
Vanilla	1528.28	62.23
HB-Lagaris	5084.54	131.67
HB-TFC	3218.89	68.52
ELM	15.47	56.94
ELM-LBFGS	5.14	22.25

High L2 error!

ELM-LBFGS



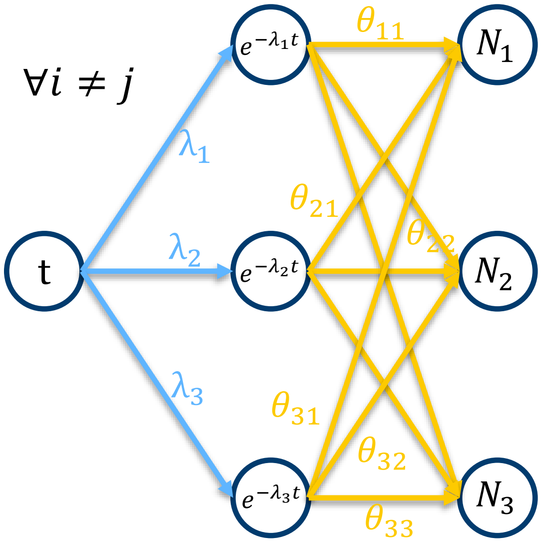
CRAM – takes 13 seconds for 10000 time steps for 128 days

Exponential Method

Assumption 1: if $\lambda_i \neq \lambda_j$ for $\lambda_i \in eigenvalues(\mathbb{A})$ $\forall i \neq j$

Assumption 2: $\mathbb{A}$ is a decay matrix

$$\Longrightarrow \vec{N}(t) = \sum_{i=1}^{n} \vec{a_i}\, e^{\lambda_i t}$$



Key ideas:

- Use a single layer NN
- Use as many neurons as nuclides (i.e. $n = h$)
- Use the exponential activation function (i.e. $\sigma = exp$)
- Use the eigenvalues as the input weights (i.e. $w_i = \lambda_i$)
- Freeze the output weights such that they form a lower triangular matrix
- Train only the output weights θ

$$\Longrightarrow \quad \overrightarrow{NN}(t; \vec{\theta}) = \sum_{i=1}^{n} \vec{\theta_i}\, e^{\lambda_i t}$$

1: Pusa, Leppänen 2010

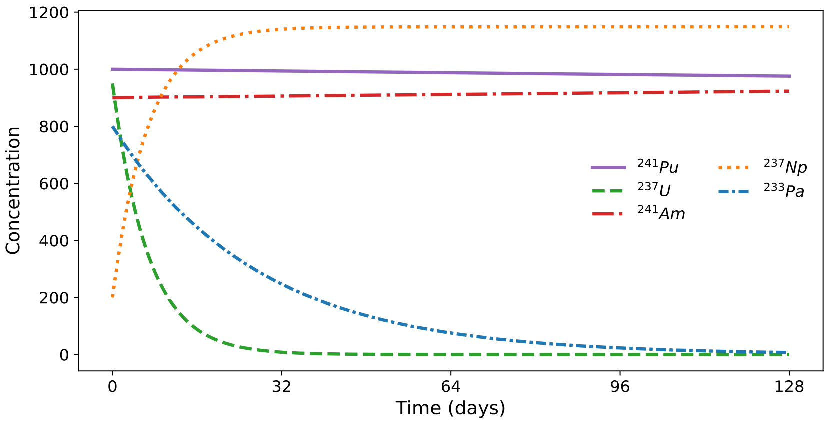
Comparison of the PINN Methods

Use Optuna to do the hyperparameters search

	Training time (s)	$L_2(\%)$
Vanilla	1528.28	62.23
HB-Lagaris	5084.54	131.67
HB-TFC	3218.89	68.52
ELM	15.47	56.94
ELM-LBFGS	5.14	184.93
Exp-Vanilla	19.72	0.000035
Exp-HB-TFC	21.18	0.0014

High training time!

Exp-HB-TFC

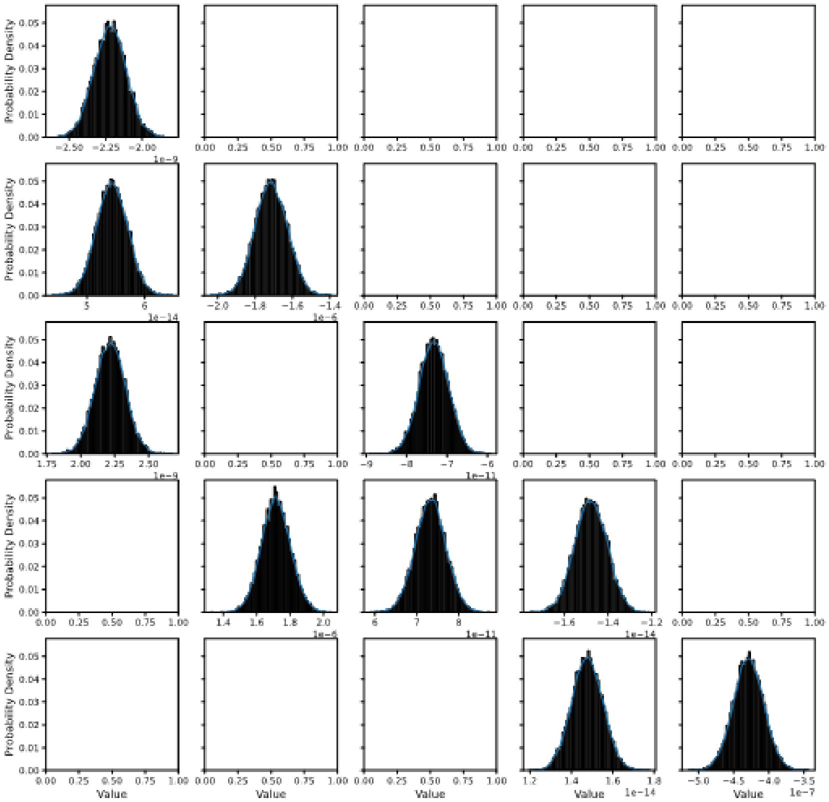


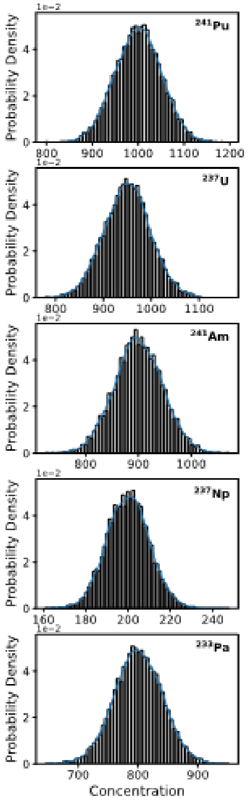
CRAM – takes 13 seconds for 10000 time steps for 128 days

PAUL SCHERRER INSTITUT

Uncertainty Quantification

Use Monte Carlo method → 10^4 samples sampled from a Gaussian distribution with $\mathrm{std} = 5\%$



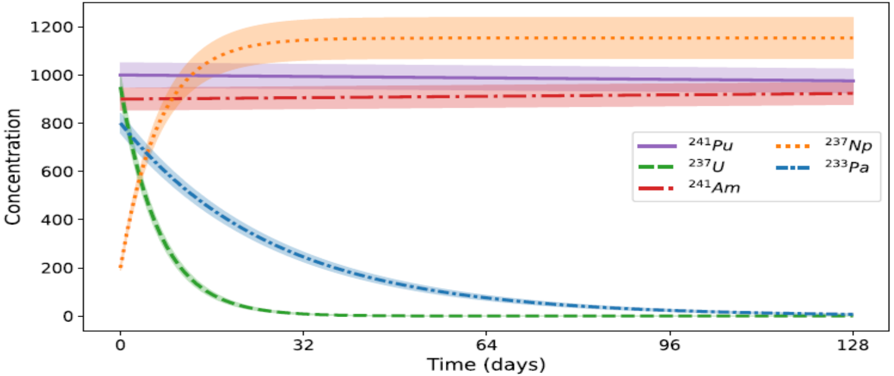


Uncertainty Quantification

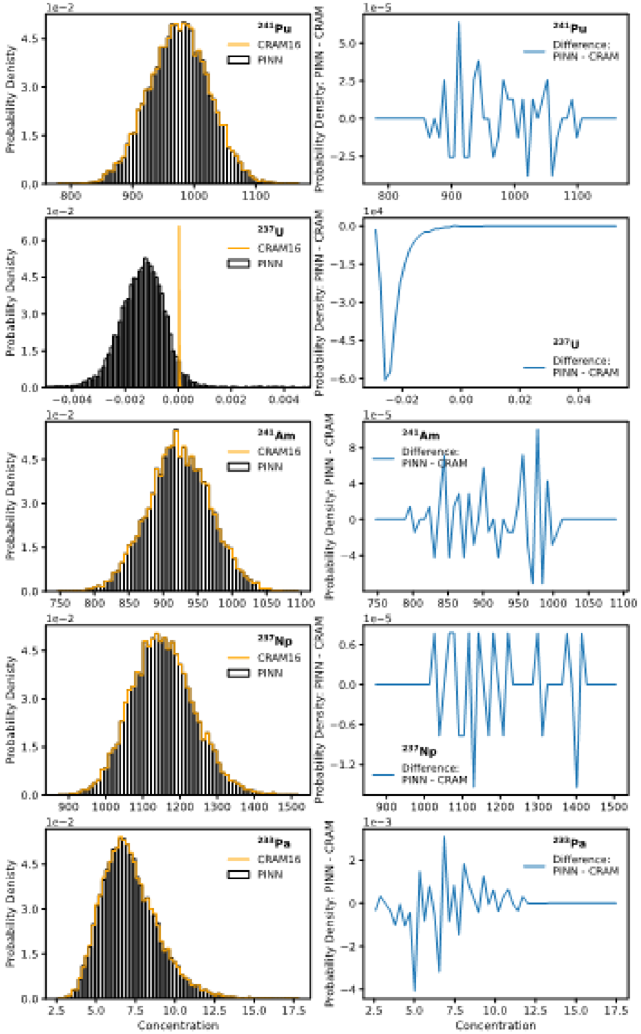
- Not all final distributions are Gaussian

  $\Longrightarrow$ linear error propagation would fail

- Thanks to transfer learning we had a speed up of over 90%:
  - 1^{st} sample $\rightarrow$ training time $= 21.7$ s
  - $2^{\text{nd}} - 1000^{\text{th}}$ sample $\rightarrow$ training time $= 2.1$ s



CRAM – takes 1 second for 128 time steps for 128 days per sample



Conclusions and Future Work

Conclusions:

- We implemented and tested PINN methods to solve the $^{241}\mathrm{Pu}$ decay for 128 days,
  - HB-TFC combined with Domain Decomposition, with and $\mathrm{L}_2 = 0.19\%$ using 1106 sub-domains solved the task, but was very slow
  - Exp-HB-TFC, Exp-Vanilla with an $\mathrm{L}_2 = 0.0014\%$, L2 = 0.000035 solved the task successfully, but still a bit slower than CRAM
- We performed UQ making use of transfer learning
  - speeding up the training time by over 90% for each sample compared to train 1000 PINNs from scratch
  - results are comparable to CRAM
  - PINN method is still slower than CRAM

Future work:

- Test the PINN methods with a larger matrix and a burnup matrix
- Alternative methods: Adaptive Weights, Use some known points as measurements points

PAUL SCHERRER INSTITUT

We create knowledge – today for tomorrow

Thanks!