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

\[ \frac{d\vec{N}(t)}{dt} = A\vec{N}(t), \quad \text{with} \quad \vec{N}(t = 0) = \vec{N}_0 \]

Where:
- \( \vec{N}(t) \in \mathbb{R}^n \) is the nuclide concentration vector
- \( A \in \mathbb{R}^{n \times n} \) 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

\[ \frac{d\vec{N}(t)}{dt} = A\vec{N}(t), \quad \text{with} \quad \vec{N}(t = 0) = \vec{N}_0 \]

- **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

\[ \frac{d\vec{N}(t)}{dt} = \mathbb{A}\vec{N}(t), \quad \text{with} \quad \vec{N}(t = 0) = \vec{N}_0 \]

- \( \vec{N}(t) \in \mathbb{R}^n \) 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:
  \[ \vec{N}_n(t) = \vec{N}_1(0) \times (\prod_{i=1}^{n-1} \lambda_i) \times \sum_{i=1}^n \frac{e^{-\lambda_i t}}{\prod_{j=1, j \neq i}^n (\lambda_j - \lambda_i)} \]

- **Computational errors, slow if \( n \) gets bigger** => numerical methods for general case
Bateman Equation

\[ \frac{d\vec{N}(t)}{dt} = A\vec{N}(t), \quad \text{with} \quad \vec{N}(t = 0) = \vec{N}_0 \]

Transmutation matrix $A$:

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

\[ \frac{d\vec{N}(t)}{dt} = (PAP)\vec{N}, \quad \text{with} \quad \vec{N}(t = 0) = P\vec{N}_0 \]
Bateman Equation

Decay matrix $\rightarrow$ can be permuted into a lower triangular matrix
• Both $\vec{N}_0$ & $\vec{A}$ have intrinsic uncertainties
• We want to evaluate the propagation of the uncertainties

$$\vec{N}_0 \pm \Delta \vec{N}_0 \rightarrow \vec{N} \pm \Delta \vec{N}$$

$$\vec{A} \pm \Delta \vec{A} \rightarrow \vec{N} \pm \Delta \vec{N}$$

• Use Monte Carlo method & transfer learning
Solving the Bateman Equation

\[ \frac{d\vec{N}(t)}{dt} = \mathbb{A}\vec{N}(t), \quad \text{with} \quad \vec{N}(t = 0) = \vec{N}_0 \]

\[ \vec{N}(t) = e^{\mathbb{A}t} \vec{N}_0 \]

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 \( \inf_{r_{k,k} \in \pi_{k,k}} \{ |r_{k,k}(x) - e^x| \} \)

Analytic Solution of Linear Chains

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 \( \theta \), and \( \epsilon \) a fixed error greater than zero, then:

\[
\forall \vec{N} \in C^0, \forall \epsilon > 0, \exists \theta : \| \vec{N}(t) - NN(t; \theta) \| < \epsilon
\]

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

\[
\frac{d\vec{N}(t)}{dt} = \mathbb{A}\vec{N}(t), \quad \text{with} \quad \vec{N}(t = 0) = \vec{N}_0
\]

- 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

• \( \mathbf{A} = 5 \times 5 \) matrix for the decay for Plutonium-241

• \( \kappa(\mathbf{A}) = 9 \times 10^9 \)

• Solve it for 128 days

• Compare the solution with CRAM over \( 10^4 \) time steps

• Use \( L_2 \) as our metric:

\[
L_2 = \frac{1}{5} \sum_{i=1}^{n} L_{N_i}^2
\]

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

\[
\mathbf{A} = \begin{pmatrix}
-2.2e-9 & 0 & 0 & 0 & 0 \\
5.4e-14 & -1.7e-6 & 0 & 0 & 0 \\
2.2e-9 & 0 & -7.3e-11 & 0 & 0 \\
0 & 1.7e-6 & 7.3e-11 & -1.5e-14 & 0 \\
0 & 0 & 0 & 1.5e-14 & -4.3e-7
\end{pmatrix}
\]
PINN Approach: Vanilla Method

• Basic implementation of PINNs\(^1\)

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

\[
\mathcal{L}(\theta) = \frac{w_{\text{physics}} \mathcal{L}_{\text{physics}}(\theta) + w_{\text{initial}} \mathcal{L}_{\text{initial}}(\theta)}{w_{\text{physics}} + w_{\text{initial}}}
\]

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

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

1: Raissi, Perdikaris, Karniadakis 2019
Comparison of the PINN Methods

Use Optuna to do the hyperparameters search

<table>
<thead>
<tr>
<th>Method</th>
<th>Training time (s)</th>
<th>$L_2$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>1528.28</td>
<td>62.23</td>
</tr>
</tbody>
</table>

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

CRAM – takes 13 seconds for 10000 time steps for 128 days
Problem with Vanilla loss: manual weight tuning

Solution: Rewrite constrained problem as unconstrained one

Two different ansatz to unconstrain the problem:

- Defined by Lagaris, Likas, Fotiadis 1998:
  \[ \vec{\psi}(t) := \vec{N}_0 + t\vec{NN}(t; \theta) \]

- Theory of Functional Connections (TFC) by Mortari 2017, Mortari, Johnston and Smith 2019
  \[ \vec{\psi} := \vec{NN}(t; \theta) + \vec{N}_0 - \vec{NN}(t = 0; \theta) \]

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

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

Use Optuna to do the hyperparameters search

<table>
<thead>
<tr>
<th>Method</th>
<th>Training time (s)</th>
<th>$L_2$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>1528.28</td>
<td>62.23</td>
</tr>
<tr>
<td>HB-Lagaris</td>
<td>5084.54</td>
<td>131.67</td>
</tr>
<tr>
<td>HB-TFC</td>
<td>3218.89</td>
<td>68.52</td>
</tr>
</tbody>
</table>

High training time, high L2 error!

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

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num of sub-domains</td>
<td>1106</td>
</tr>
<tr>
<td>Length sub-domain</td>
<td>2hr 46min</td>
</tr>
<tr>
<td>Neurons per sub-domain</td>
<td>144</td>
</tr>
<tr>
<td>Time steps per sub-domain</td>
<td>75552</td>
</tr>
</tbody>
</table>

### Results

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>training time 1° sub-domain</td>
<td>39min 6s</td>
</tr>
<tr>
<td>average training time [2, 1106] sub-domains</td>
<td>15.6 s</td>
</tr>
<tr>
<td>$L_2$</td>
<td>0.1945 %</td>
</tr>
</tbody>
</table>
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

\[ w_j \in U(-a, a), \quad b_j \in U(-c, c) \]

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

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

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

Solve the linear system \( \Rightarrow \nabla \mathcal{L}(\vec{\theta}) = -J \vec{\theta} \)
Comparison of the PINN Methods

Use Optuna to do the hyperparameters search

<table>
<thead>
<tr>
<th>Method</th>
<th>Training time (s)</th>
<th>$L_2$(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>1528.28</td>
<td>62.23</td>
</tr>
<tr>
<td>HB-Lagaris</td>
<td>5084.54</td>
<td>131.67</td>
</tr>
<tr>
<td>HB-TFC</td>
<td>3218.89</td>
<td>68.52</td>
</tr>
<tr>
<td>ELM</td>
<td>15.47</td>
<td>56.94</td>
</tr>
<tr>
<td>ELM-LBFGS</td>
<td>5.14</td>
<td>22.25</td>
</tr>
</tbody>
</table>

High L2 error!

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

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

Assumption 2: $A$ is a decay matrix

\[
\Rightarrow \tilde{N}(t) = \sum_{i=1}^{n} a_i e^{\lambda_i t}
\]

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 $\theta$

\[
\Rightarrow \tilde{NN}(t; \Theta) = \sum_{i=1}^{n} \theta_i e^{\lambda_i t}
\]

1: Pusa, Leppänen 2010
Use Optuna to do the hyperparameters search

<table>
<thead>
<tr>
<th>Method</th>
<th>Training time (s)</th>
<th>$L_2$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>1528.28</td>
<td>62.23</td>
</tr>
<tr>
<td>HB-Lagaris</td>
<td>5084.54</td>
<td>131.67</td>
</tr>
<tr>
<td>HB-TFC</td>
<td>3218.89</td>
<td>68.52</td>
</tr>
<tr>
<td>ELM</td>
<td>15.47</td>
<td>56.94</td>
</tr>
<tr>
<td>ELM-LBFGS</td>
<td>5.14</td>
<td>184.93</td>
</tr>
<tr>
<td>Exp-Vanilla</td>
<td>19.72</td>
<td>0.000035</td>
</tr>
<tr>
<td>Exp-HB-TFC</td>
<td>21.18</td>
<td>0.0014</td>
</tr>
</tbody>
</table>

High training time!

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

Use Monte Carlo method → $10^4$ samples sampled from a Gaussian distribution with std = 5%
Uncertainty Quantification

- Not all final distributions are Gaussian
  \[ \Rightarrow \] linear error propagation would fail
- Thanks to transfer learning we had a speed up of over 90%:
  - 1\textsuperscript{st} sample $\rightarrow$ training time $= 21.7$ s
  - 2\textsuperscript{nd} – 1000\textsuperscript{th} sample $\rightarrow$ training time $= 2.1$ s

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

• We implemented and tested PINN methods to solve the $^{241}$Pu decay for 128 days,
  o HB-TFC combined with Domain Decomposition, with and $L_2 = 0.19\%$ using 1106 sub-domains solved the task, but was very slow
  o Exp-HB-TFC, Exp-Vanilla with an $L_2 = 0.0014\%$, $L_2 = 0.000035$ solved the task successfully, but still a bit slower than CRAM

• We performed UQ making use of transfer learning
  o speeding up the training time by over 90\% for each sample compared to train 1000 PINNs from scratch
  o results are comparable to CRAM
  o 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
We create knowledge – today for tomorrow

Thanks!