Solving the Bateman Equation using Physics Informed Neural Networks (PINNs): an introduction to building an elementary PINN to solve decay chains in nuclear physics

by

Genevieve Alpar

University of Dallas, 2025

I. ABSTRACT

In nuclear physics, accurately modeling the time evolution of isotope abundances is essential for applications ranging from reactor design to medical imaging and astrophysics. The Bateman equations—linear first-order ordinary differential equations—describe these dynamics based on known decay rates. However, solving these equations can be computationally intensive, especially for long or stiff decay chains. In this study, we implement a physics-informed neural network (PINN) using PyTorch to model the decay chain of $^{88}\mathrm{Y} \rightarrow^{88} \mathrm{Sr}$. The PINN architecture consists of five hidden layers with 64 neurons per layer and employs a combination of linear and `tanh` activation functions. It takes time as input and outputs the predicted abundances of ^{88}Y and ^{88}Sr. The model is trained using a loss function that incorporates the differential equations themselves, ensuring physical consistency. We compare the PINN's output to the analytical solution and demonstrate that the network achieves high accuracy, highlighting the potential of PINNs as efficient and reliable tools for solving stiff systems in nuclear decay modeling.

II. INTRODUCTION

A. What is the Bateman Equation

The Bateman equation—formulated by Ernest Rutherford in 1905 and solution provided by Harry Bateman in 1910—is a mathematical model in nuclear physics describing the abundances and activities in a decay chain as a function of time, based primarily on the decay rates and initial abundances. In mathematical terms, the Bateman equations are formed in its simplest form by

$$\frac{dN_1(t)}{dt} = -\lambda_1 N_1(t) \tag{1}$$

$$\frac{dN_i(t)}{dt} = -\lambda_i N_i(t) + \lambda_{i-1} N_{i-1}(t) \tag{2}$$

$$\frac{dN_k(t)}{dt} = \lambda_{k-1} N_{k-1}(t) \tag{3}$$

where λ is the decay constant of the respective isotope, related to its half life by

$$T_{1/2} = \frac{ln(2)}{\lambda_k} \tag{4}$$

and $N_i(t)$ is the number of atoms of the isotope i at time t that decays into isotope $i + 1$ [1]. This common formula can be adapted to handle decay branches and applied to more complex decay chains. For the sake of simplicity, we will be modeling the decay chain of $^{88}\mathrm{Y} \rightarrow ^{88}\mathrm{Sr}$, governed by the system of ODES

$$\frac{dN_Y}{dt} = -\lambda_Y N_Y \tag{5}$$

$$\frac{dN_{Sr}}{dt} = \lambda_Y N_Y \tag{6}$$

where λ_Y is the decay rate of ^{88}Y and is a known value of 0.0065 days^{-1}.

B. What are Neural Networks?

Neural networks are computational models inspired by the structure and functions of biological neural networks used to solve complex systems. Within a neural network, each neuron receives an input, processes it through mathematical functions, and passes the output to the next layer. Below in Figure 1, observe a simple structure for a basic neural network.

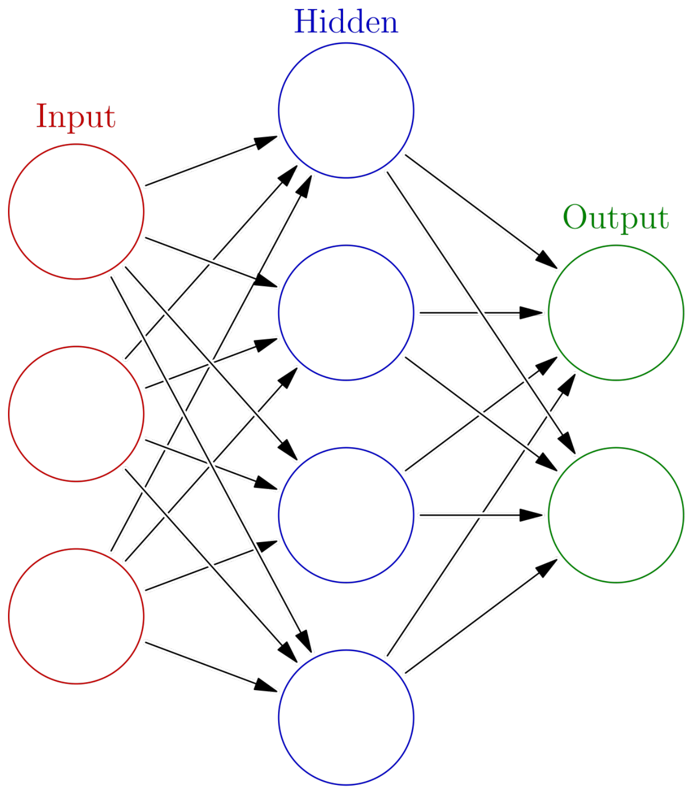

FIG. 1. An artifical neural network is an interconnected group of nodes, inspired by a simplification of neurons in a brain. Each circular node represents an artificial neuron and an arrow represents a connection from the output of one artificial neuron to the input of another.[2]

A neural network consists of nodes called artificial neurons, which loosely model the neurons in the brain. These nodes are connected by edges, which try to model the synapses in the brain. Each artificial neuron will receive signals from the connected neurons and then process them and sens a signal to other connected neurons. The signal is a real number and the output of each neuron can be computed by some non-linear function of the sum of the signal's inputs. This is known as the activation function. The neurons are made into layers, which perform different transformations on their inputs. Signals pass from the input layer to the output layer, and any hidden layers in between [2]. In the case of this paper, a neural network is used for predictive modeling for the Bateman equations.

More specifically, physics-informed neural networks (PINNs) are universal function approximators that embed knowledge of physical laws that govern a given data-set in the learning process, and are described by PDEs. The prior general physical laws help to train the neural networks to increase the accuracy of the function approximation [3]. In this paper, the system of bateman equations are used to train the NN.

III. SOLVING THE DECAY CHAIN OF $^{88}Y \to {}^{88}Sr$ - AN EXAMPLE IN NUCLEAR PHYSICS

Let us observe the decay chain of $^{88}Y \to^{88} Sr$. We start with with Bateman equation ODEs that govern the decay from ^{88}Y into ^{88}Sr from Equations 5 and 6. To define a neural network, we first begin by defining the input, which will be time $t \in [0, T]$ in days, and the output as the expected abundances of ^{88}Y and ^{88}Sr $\hat{N}_Y(t)$ and $\hat{N}_{Sr}(t)$ respectively. We then build the architecture of the neural network, where the input has one neuron and the output has two neurons. Then we build five layers. The first layer is a fully connect (dense) layer that takes 1 input neuron of time and outputs 64 neurons. Think of this as projecting a scalar of time into a 64-dimensional space. Then there is a tanh layer that applies the activation function element-wise to the 64 outputs, which introduces non-linearity to help the network model complex relationships. We then do this two more times to get another linear layer, another tanh layer, then a final linear layer that outputs a singular scalar value (the prediction) from the 64-neuron input.

We then define the physics residuals, which is what separates PINNs from NNs. These physics residuals of the Bateman equations will be trained by the network to be as close to zero as possible. These residuals measure how well the neural network satisfies the Bateman equations at each point in time, and this value is defined by the equations

$$\mathcal{R}_Y(t) = \frac{d\hat{N}_Y}{dt} + \lambda_Y \hat{N}_Y(t) \tag{7}$$

$$\mathcal{R}_{Sr}(t) = \frac{d\hat{N}_{Sr}}{dt} - \lambda_Y \hat{N}_Y(t) \tag{8}$$

The residuals are then used to define the loss function, which combines the residual loss to enforce the Bateman equations and the initial condition loss to match the known starting values. For simplicity, we start with 1 mole of ^{88}Y and 0 moles of ^{88}Sr. Our loss function approximates the total accuracy of the PINN to solve the Bateman equations, and is described by the equation

$$\mathcal{L} = \text{MSE}_{\text{residual}} + \text{MSE}_{\text{initial}} \tag{9}$$

where the residual mean square error is from the physics and the initial mean square error is from the data. These residuals are defined as

$$\text{MSE}_{\text{residual}} = \frac{1}{N}\sum_{i=1}^{N}\left(\mathcal{R}_Y(t_i)^2 + \mathcal{R}_{Sr}(t_i)^2\right) \tag{10}$$

$$\text{MSE}_{\text{initial}} = \left(\hat{N}_Y(0) - 1\right)^2 + \left(\hat{N}_{Sr}(0) - 0\right)^2 \tag{11}$$

The complex part of the PINN is the training, where different steps are taken to train the NN to accurately produce the solutions to the ODEs in the Bateman equations. The first step is to learn the decay rate. We first give the PINN a value 0.05 days^{-1} and inform it that the real decay rate of ^{88}Y is 0.0065 days^{-1}. The model learns the correct decay rate by minimizing error between the predicted decay and accumulation of the true data. The PINN is then trained using the loss function defined above, which is the mean square error between predictions and actual values that guides the model in adjusting its parameters. For example, it will take into account the loss function to accurately learn the decay rate, which it will eventually decay to. The next step that occurs is gradients are used to update parameters through backpropagation, helping the model to minimize error. Finally, the PINN takes into account an optimizer (like Adam, AdamW, etc.) to help in adjusting the model's parameters efficiently using adaptive learning rates. Changing the learning rates can be sensitive to the system, so this is a parameter in the PINN's learning that may need to be adjusted. See the python code in Section A for specifics on the coding aspect of the PINN.

After training, the neural network approximates the abundances of ^{88}Y and ^{88}Sr and produces the following plot from the data in section A

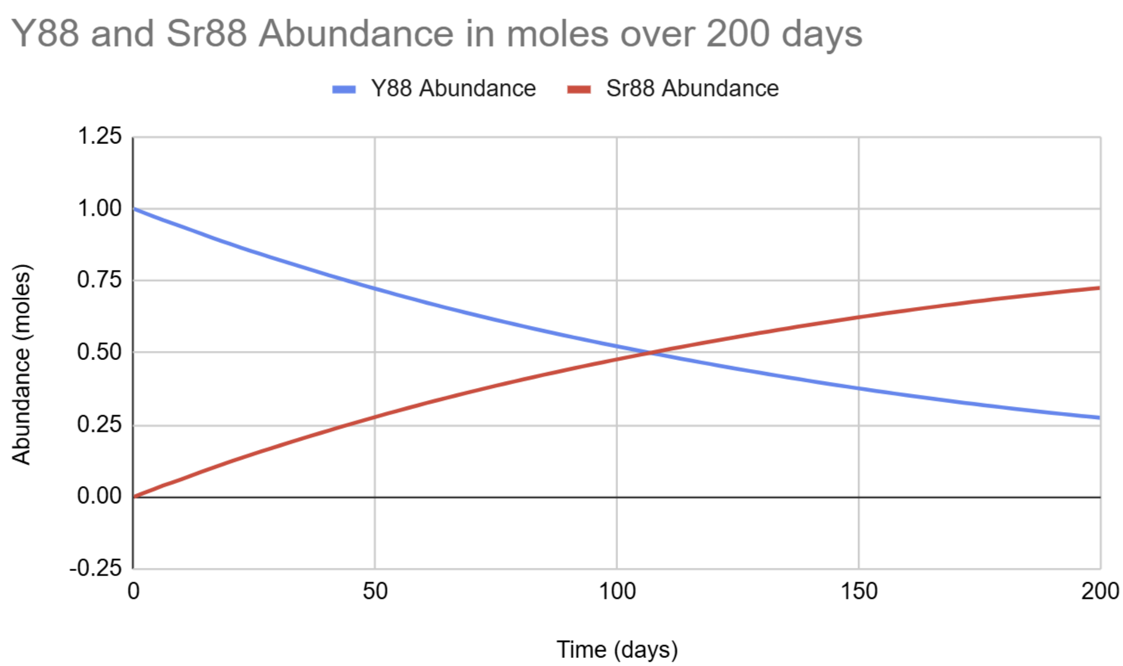

FIG. 2. Abundance of ^{88}Y and ^{88}Sr over time plotted from data in Section A.

IV. CONCLUSION

Overall, the PINN appears to approximate the abundances of ^{88}Y and ^{88}Sr very well. We can fact check this by validating the abundances at the half-life, 106.6 days. From the graph, we can see that the lines from ^{88}Y and ^{88}Sr intersect at 106.6 days and 0.50 abundances, which verifies that our PINN model works very well to approximate the abundances of ^{88}Y and ^{88}Sr over time. The PINN takes into account the decay rate of ^{88}Y and trains the model using the Bateman equation residuals, the loss function, and an optimizer.

Appendix A: Example Python Code

INPUT:

```
import torch

import torch.nn as nn

import torch.optim as optim

import matplotlib.pyplot as plt


# Define the PINN model

class PINN_Y88(nn.Module):

    def __init__(self):

        super(PINN_Y88, self).__init__()

        self.net = nn.Sequential(

            nn.Linear(1, 64),

            nn.Tanh(),

            nn.Linear(64, 64),

            nn.Tanh(),

            nn.Linear(64, 1)

        )

        self.lambda_Y = nn.Parameter(torch.tensor(0.02, dtype=torch.float32))


    def forward(self, t):

        return self.net(t)


# Physics-based loss: dY/dt = -lambda * Y

def physics_loss(model, t):

    t.requires_grad = True

    Y = model(t)

    dY_dt = torch.autograd.grad(
```

```
        outputs=Y,

        inputs=t,

        grad_outputs=torch.ones_like(Y),

        create_graph=True

    )[0]

    lambda_Y = model.lambda_Y

    return torch.mean((dY_dt + lambda_Y * Y)**2)


# Training data: time values and known Y(t)

lambda_true = 0.0065

t_train = torch.linspace(0, 200, 200).view(-1, 1).float()

Y_true = torch.exp(-lambda_true * t_train)


# Initialize model and optimizer

model = PINN_Y88()

optimizer = optim.Adam(model.parameters(), lr=1e-3)


# Training loop

epochs = 5000

for epoch in range(epochs):

    model.train()

    optimizer.zero_grad()


    Y_pred = model(t_train)


    # Data loss: match known decay curve

    data_loss = torch.mean((Y_pred - Y_true)**2)


    # Physics loss: match the ODE
```

```
    p_loss = physics_loss(model, t_train)

    # Total loss
    loss = data_loss + p_loss

    loss.backward()
    optimizer.step()

    if epoch % 500 == 0:
        print(f"Epoch {epoch}, Total Loss: {loss.item():.6f}, Lambda_Y: {model.lambda_Y.item():.5f}")

# Evaluate and plot
model.eval()
t_test = torch.linspace(0, 200, 100).view(-1, 1).float()
with torch.no_grad():
    Y_test = model(t_test).cpu().numpy()
    Sr_test = 1 - Y_test

# Print predictions
print("Predictions (Y-88, Sr-88):")
for i, time in enumerate(t_test.numpy().flatten()):
    print(f"Time: {time:.2f} days, Y-88: {Y_test[i, 0]:.4f}, Sr-88: {Sr_test[i, 0]:.4f}")

# t_np = t_test.numpy().flatten()
# plt.plot(t_np, Y_test[:, 0], label="Y-88 (PINN)")
# plt.plot(t_np, Sr_test[:, 0], label="Sr-88 (PINN)")
# plt.xlabel("Time (days)")
# plt.ylabel("Amount (mol)")
# plt.title("Y-88 Decay and Sr-88 Accumulation (PINN)")
```

```
# plt.legend()

# plt.grid(True)

# plt.show()
```

OUTPUT:

```
Epoch 0, Total Loss: 0.502827, Lambda_Y: 0.04900

Epoch 500, Total Loss: 0.000011, Lambda_Y: 0.00616

Epoch 1000, Total Loss: 0.000008, Lambda_Y: 0.00640

Epoch 1500, Total Loss: 0.000001, Lambda_Y: 0.00643

Epoch 2000, Total Loss: 0.000001, Lambda_Y: 0.00643

Epoch 2500, Total Loss: 0.000004, Lambda_Y: 0.00646

Epoch 3000, Total Loss: 0.000006, Lambda_Y: 0.00647

Epoch 3500, Total Loss: 0.000000, Lambda_Y: 0.00648

Epoch 4000, Total Loss: 0.000001, Lambda_Y: 0.00648

Epoch 4500, Total Loss: 0.000038, Lambda_Y: 0.00638

Predictions (Y-88, Sr-88):

Time: 0.00 days, Y-88: 1.0002, Sr-88: -0.0002

Time: 2.02 days, Y-88: 0.9868, Sr-88: 0.0132

Time: 4.04 days, Y-88: 0.9739, Sr-88: 0.0261

Time: 6.06 days, Y-88: 0.9608, Sr-88: 0.0392

Time: 8.08 days, Y-88: 0.9493, Sr-88: 0.0507

Time: 10.10 days, Y-88: 0.9373, Sr-88: 0.0627

Time: 12.12 days, Y-88: 0.9248, Sr-88: 0.0752

Time: 14.14 days, Y-88: 0.9123, Sr-88: 0.0877

Time: 16.16 days, Y-88: 0.9000, Sr-88: 0.1000

Time: 18.18 days, Y-88: 0.8880, Sr-88: 0.1120

Time: 20.20 days, Y-88: 0.8764, Sr-88: 0.1236

Time: 22.22 days, Y-88: 0.8650, Sr-88: 0.1350

Time: 24.24 days, Y-88: 0.8538, Sr-88: 0.1462

Time: 26.26 days, Y-88: 0.8428, Sr-88: 0.1572
```

```
Time: 28.28 days, Y-88: 0.8320, Sr-88: 0.1680
Time: 30.30 days, Y-88: 0.8213, Sr-88: 0.1787
Time: 32.32 days, Y-88: 0.8107, Sr-88: 0.1893
Time: 34.34 days, Y-88: 0.8002, Sr-88: 0.1998
Time: 36.36 days, Y-88: 0.7898, Sr-88: 0.2102
Time: 38.38 days, Y-88: 0.7795, Sr-88: 0.2205
Time: 40.40 days, Y-88: 0.7693, Sr-88: 0.2307
Time: 42.42 days, Y-88: 0.7592, Sr-88: 0.2408
Time: 44.44 days, Y-88: 0.7493, Sr-88: 0.2507
Time: 46.46 days, Y-88: 0.7394, Sr-88: 0.2606
Time: 48.48 days, Y-88: 0.7297, Sr-88: 0.2703
Time: 50.51 days, Y-88: 0.7201, Sr-88: 0.2799
Time: 52.53 days, Y-88: 0.7106, Sr-88: 0.2894
Time: 54.55 days, Y-88: 0.7013, Sr-88: 0.2987
Time: 56.57 days, Y-88: 0.6921, Sr-88: 0.3079
Time: 58.59 days, Y-88: 0.6830, Sr-88: 0.3170
Time: 60.61 days, Y-88: 0.6740, Sr-88: 0.3260
Time: 62.63 days, Y-88: 0.6652, Sr-88: 0.3348
Time: 64.65 days, Y-88: 0.6565, Sr-88: 0.3435
Time: 66.67 days, Y-88: 0.6479, Sr-88: 0.3521
Time: 68.69 days, Y-88: 0.6395, Sr-88: 0.3605
Time: 70.71 days, Y-88: 0.6312, Sr-88: 0.3688
Time: 72.73 days, Y-88: 0.6230, Sr-88: 0.3770
Time: 74.75 days, Y-88: 0.6149, Sr-88: 0.3851
Time: 76.77 days, Y-88: 0.6069, Sr-88: 0.3931
Time: 78.79 days, Y-88: 0.5991, Sr-88: 0.4009
Time: 80.81 days, Y-88: 0.5913, Sr-88: 0.4087
Time: 82.83 days, Y-88: 0.5837, Sr-88: 0.4163
Time: 84.85 days, Y-88: 0.5761, Sr-88: 0.4239
```

Time: 86.87 days, Y-88: 0.5687, Sr-88: 0.4313

Time: 88.89 days, Y-88: 0.5613, Sr-88: 0.4387

Time: 90.91 days, Y-88: 0.5541, Sr-88: 0.4459

Time: 92.93 days, Y-88: 0.5469, Sr-88: 0.4531

Time: 94.95 days, Y-88: 0.5398, Sr-88: 0.4602

Time: 96.97 days, Y-88: 0.5329, Sr-88: 0.4671

Time: 98.99 days, Y-88: 0.5260, Sr-88: 0.4740

Time: 101.01 days, Y-88: 0.5191, Sr-88: 0.4809

Time: 103.03 days, Y-88: 0.5124, Sr-88: 0.4876

Time: 105.05 days, Y-88: 0.5057, Sr-88: 0.4943

Time: 107.07 days, Y-88: 0.4992, Sr-88: 0.5008

Time: 109.09 days, Y-88: 0.4927, Sr-88: 0.5073

Time: 111.11 days, Y-88: 0.4862, Sr-88: 0.5138

Time: 113.13 days, Y-88: 0.4799, Sr-88: 0.5201

Time: 115.15 days, Y-88: 0.4736, Sr-88: 0.5264

Time: 117.17 days, Y-88: 0.4674, Sr-88: 0.5326

Time: 119.19 days, Y-88: 0.4613, Sr-88: 0.5387

Time: 121.21 days, Y-88: 0.4552, Sr-88: 0.5448

Time: 123.23 days, Y-88: 0.4492, Sr-88: 0.5508

Time: 125.25 days, Y-88: 0.4433, Sr-88: 0.5567

Time: 127.27 days, Y-88: 0.4375, Sr-88: 0.5625

Time: 129.29 days, Y-88: 0.4317, Sr-88: 0.5683

Time: 131.31 days, Y-88: 0.4260, Sr-88: 0.5740

Time: 133.33 days, Y-88: 0.4204, Sr-88: 0.5796

Time: 135.35 days, Y-88: 0.4148, Sr-88: 0.5852

Time: 137.37 days, Y-88: 0.4093, Sr-88: 0.5907

Time: 139.39 days, Y-88: 0.4039, Sr-88: 0.5961

Time: 141.41 days, Y-88: 0.3985, Sr-88: 0.6015

Time: 143.43 days, Y-88: 0.3932, Sr-88: 0.6068

```
Time: 145.45 days, Y-88: 0.3880, Sr-88: 0.6120

Time: 147.47 days, Y-88: 0.3829, Sr-88: 0.6171

Time: 149.49 days, Y-88: 0.3778, Sr-88: 0.6222

Time: 151.52 days, Y-88: 0.3728, Sr-88: 0.6272

Time: 153.54 days, Y-88: 0.3679, Sr-88: 0.6321

Time: 155.56 days, Y-88: 0.3630, Sr-88: 0.6370

Time: 157.58 days, Y-88: 0.3583, Sr-88: 0.6417

Time: 159.60 days, Y-88: 0.3536, Sr-88: 0.6464

Time: 161.62 days, Y-88: 0.3489, Sr-88: 0.6511

Time: 163.64 days, Y-88: 0.3444, Sr-88: 0.6556

Time: 165.66 days, Y-88: 0.3399, Sr-88: 0.6601

Time: 167.68 days, Y-88: 0.3355, Sr-88: 0.6645

Time: 169.70 days, Y-88: 0.3311, Sr-88: 0.6689

Time: 171.72 days, Y-88: 0.3268, Sr-88: 0.6732

Time: 173.74 days, Y-88: 0.3226, Sr-88: 0.6774

Time: 175.76 days, Y-88: 0.3185, Sr-88: 0.6815

Time: 177.78 days, Y-88: 0.3145, Sr-88: 0.6855

Time: 179.80 days, Y-88: 0.3105, Sr-88: 0.6895

Time: 181.82 days, Y-88: 0.3066, Sr-88: 0.6934

Time: 183.84 days, Y-88: 0.3027, Sr-88: 0.6973

Time: 185.86 days, Y-88: 0.2990, Sr-88: 0.7010

Time: 187.88 days, Y-88: 0.2953, Sr-88: 0.7047

Time: 189.90 days, Y-88: 0.2917, Sr-88: 0.7083

Time: 191.92 days, Y-88: 0.2881, Sr-88: 0.7119

Time: 193.94 days, Y-88: 0.2846, Sr-88: 0.7154

Time: 195.96 days, Y-88: 0.2812, Sr-88: 0.7188

Time: 197.98 days, Y-88: 0.2779, Sr-88: 0.7221

Time: 200.00 days, Y-88: 0.2746, Sr-88: 0.7254
```

[1] Wikipedia contributors, "Bateman equation," https://en.wikipedia.org/wiki/Bateman_equation (n.d.), accessed: 2025-05-10.

[2] Wikipedia contributors, "Neural network (machine learning)," https://en.wikipedia.org/wiki/Neural_network_(machine_learning) (n.d.), accessed: 2025-05-10.

[3] Wikipedia contributors, "Physics-informed neural networks," https://en.wikipedia.org/wiki/Physics-informed_neural_networks (n.d.), accessed: 2025-05-10.