The Euler-Bernoulli Beam Equation and Physics Informed Neural Networks (PINN)

Robert Haas

July 4, 2025

Abstract

A numerical study for the 1D Euler-Bernoulli equation for a deflected beam is done here. The Euler-Bernoulli equation will be presented in different forms which are used in the calculations afterwards. For the calculations of the Euler-Bernoulli equation *physics informed neural networks (PINN)* have been used. *First*, the equation itself has been solved using a PINN. The result has been compared with a classical finite element (FE) solution. Since FE solvers are optimized for classical linear engineering problems, the FE approach performs far better than the PINN. *Second*, simulated deflection measurements are used for approximating a solution that provides the data to estimate the unknown elastic modulus This is done with two PINNs . This is a typical data-driven use case for using PINNs and here very satisfactory results can be obtained by a careful adjustment of the calculation parameters.

Contents

1 The Euler-Bernoulli Equation of a Beam

For studying the bending behavior of a beam we consider a beam with the following assumptions:

(A1) The beam under consideration is a cantilever beam: At one end the beam is fixed in that way, that there is no displacement and no rotation allowed. At the other end -the free end- displacement and rotation may occur.

(A2) The beam has a fixed constant cross-section.

(A3) Hooke's law, i.e. the stress σ is proportional to the strain ϵ, is fulfilled everywhere.

The Euler-Bernoulli equation is given by

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}\left(E(x) \cdot I \cdot \frac{\mathrm{d}^2 w(x)}{\mathrm{d}x^2}\right) = q \text{ for } x \in (0, L). \tag{1}$$

With constant elastic modulus E this simplifies to

$$E \cdot I \cdot \frac{\mathrm{d}^4 w(x)}{\mathrm{d}x^4} = q \text{ for } x \in (0, L) \tag{2}$$

subject to the boundary conditions

$$w(0) = \frac{\mathrm{d}w}{\mathrm{d}x}(0) = 0 \text{ , and } \frac{\mathrm{d}^2 w}{\mathrm{d}x^2}(L) = \frac{\mathrm{d}^3 w}{\mathrm{d}x^3}(L) = 0. \tag{3}$$

The physical quantities in equation (1) are described in the following table.

Symbol	**SI Unit**	**Description**	**Typical Value**
L	m	Length	5 m
w	m	Deflection	$w << L$
E	Pa	Elastic modulus (Young's modulus)	210 000 $\frac{\mathrm{N}}{\mathrm{mm}^2}$
I	m^4	Second moment of area	10^{-6} m^4
q	$\frac{\mathrm{N}}{\mathrm{m}}$	Distributed load	1000 $\frac{\mathrm{N}}{\mathrm{m}}$

Table 1: Physical quantities.

Unless otherwise stated we consider the Euler-Bernoulli equation (2), i.e. the variant with constant elastic modulus E. For this case an exact solution is given by

$$w(x) = \frac{q}{24E \cdot I} \cdot x^2 \cdot \left(x^2 - 4L \cdot x + 6L^2\right).$$

With the values given in Table 1 we get

$$w(L) = \frac{q}{24E \cdot I} \cdot L^2 \cdot \left(L^2 - 4L \cdot L + 6L^2\right) = \frac{q \cdot L^4}{8E \cdot I} \approx 0.37202 \text{ m}, \tag{4}$$

which is the deflection of the beam at position $x = L$.

Following [1, pp. 3] a dimensionless form of the equation (1) can be derived to give

$$\frac{\mathrm{d}^4 v(\xi)}{\mathrm{d}\xi^4} = 1 \text{ for } \xi \in (0,1), \tag{5}$$

where

$$\begin{aligned} E(x) &= E_{\text{char}} \cdot e(\xi), \\ w(x) &= w_{\text{char}} \cdot v(\xi), \\ x &= \xi \cdot L, \\ w_{\text{char}} &= \frac{q \cdot L^4}{E \cdot I}, \\ \frac{\mathrm{d}^4 w(x)}{\mathrm{d}x^4} &= \frac{w_{\text{char}}}{L^4} \cdot \frac{\mathrm{d}^4 v(\xi)}{\mathrm{d}\xi^4}. \end{aligned}$$

Here E_{char} may be chosen as

$$E_{\text{char}} = \max_{x\in[0,1]} E(x).$$

The boundary conditions now read as

$$v(0) = \frac{\mathrm{d}v}{\mathrm{d}\xi}(0) = 0 \text{ , and } \frac{\mathrm{d}^2 v}{\mathrm{d}\xi^2}(1) = \frac{\mathrm{d}^3 v}{\mathrm{d}\xi^3}(1) = 0. \tag{6}$$

Again, there exists an exact solution, given by

$$v(\xi) = \frac{1}{24} \cdot \xi^2 \cdot \left(\xi^2 - 4\xi + 6\right).$$

The equation (13) is crucial for the application of some numerical methods, e.g. physics informed neural networks (PINN).

The Finite-Element Method (FEM) for solving the Euler-Bernoulli Equation

The FEM is set up for the the dimensionless problem (13) and the boundary conditions (6). Here, classical 1D beam elements (third order Hermite polynomials) on 99 elements are used to solve this equation. The theory on FEM to solve the Euler-Bernoulli equation can be found in any good text book. The results of the calculation will be shortly presented in Table 6.

2 The Idea of a Physics Informed Neural Network (PINN)

Neural Networks are usually used for either for *classification* or *approximation*. A short introduction to neural networks can be found in [2] and [3, pp. 809] for example. Today the main applications of neural networks contain classification. On the other hand, neural networks can be used to approximate an (unknown) function u. If this function u arises from a physics or engineering context, i.e. from a mathematical equation of physics the equation structure will be incorporated into the neural network or the learning algorithm in some way. In this case the neural network is often called *phyiscs informed neural networks (PINN)*. Here the key task is to construct an approximation $\hat{u}$ for the unknown u. Assuming there is a PINN of n layers then the output $O_{(k)}$of layer k is connected to the output $O_{(k-1)}$ of layer $k-1$

via

$$O_i^{(k)} = f^{(k-1)}\left(b_i^{(k-1)} + \sum_j \theta_{i,j}^{(k-1)} O_j^{(k-1)}\right),$$

where $O_j^{(k-1)}$ is the output of the j-th node in layer $k-1$, $\theta_{i,j}^{(k-1)}$ is the weight of the connection between the j-th node in layer $k-1$ and the i-th node in layer k, $b_i^{(k-1)}$ is the bias neuron of i-th node in layer $k-1$ and $f^{(k-1)}$ is the activation of the nodes of layer $k-1$.

For a function v of one variable and a PINN with only one hidden layer and $f^{(2)} = 1$ the approximation task is to find weights θ and a bias neuron b such that

$$\hat{v}(\xi) = b_1^{(2)} + \sum_j \theta_j^{(2)} f^{(1)}\left(b_j^{(1)} + \theta_1^{(1)}\xi\right).$$

This illustrates that the unknown function v is essentially approximated by a linear combination of the activation functions whereas the numbers $b_j^{(1)}$, $b_1^{(2)}$, $\theta_1^{(1)}$ and $\theta_j^{(2)}$ are adjusted by the backpropagation algorithm to minimise $\sum_{j=1}^{N} \|\hat{v}(\xi_j) - v(\xi_j)\|^2$, for example.

There are two important application cases where PINNs are applied to:

- Solving a set of mathematical equations. This is called the *direct problem*. The set of mathematical equations is usually a set of differential equations and their boundary conditions. These differential equations can be either ordinary or partial differential equations.
- Estimation of an unknown parameter in the set of mathematical equations where its solution is given by simulated measured values for example. This is called the *inverse problem*.

Typically neural networks learn by minimising a loss function. A key concept in PINN is to extend the loss function by a term that penalizes deviations from the governing equations and the boundary conditions.

Mathematical equations in engineering or physics are often of the form

$$\begin{aligned} \mathscr{A}(v) &= 0, \text{ in the domain,} \\ \mathscr{B}(v) &= 0, \text{ on the boundary.} \end{aligned}$$

In most cases, $\mathscr{A}$ is a differential operator either for an ODE or a PDE. Then, $\mathscr{B}$ is the boundary operator. For the direct problem the function typically consists of

$$\text{Physics loss} = \mathscr{L}_{\text{Physics}}(\hat{v}) = \beta \sum_{k=1}^{N} \|\mathscr{A}(\hat{v}(\xi_k))\|^2 \tag{7}$$

$$\text{Boundary loss} = \mathscr{L}_{\text{Boundary}}(\hat{v}) = \beta \sum_{k=1}^{M} \|\mathscr{B}(\hat{v}(\xi_{n_k}))\|^2 \text{ with boundary points } \xi_{n_k} \tag{8}$$

$$\text{Regularisation term} = \beta \sum_{\theta \in \text{PINNWeights}} \|\theta\|^2. \tag{9}$$

Often, neural network libraries use the mean square error as loss function, then $\beta = 1/N$ i.e. the reciprocal of the number of sample points. Otherwise, if the square of the Euclidean distance is used, then β is equal to one. For the inverse problem the PINN approximation an additional loss term,

$$\text{Data loss} = \mathscr{L}_{\text{Data}}(\hat{v}) = \beta \sum_{k=1}^{N} \|\hat{v}(\xi_k) - v_k\|^2,$$

wii be added to the terms (7), (8) and (9). Here v_k are the simulated measured values at position ξ_k.

3 PINNs for solving the Euler-Bernoulli Equation

In case of the equation (1) the term would read $\|E \cdot I \cdot \mathrm{d}^4 w(x)/\mathrm{d}x^4 - q\|^2$. Since neural networks are sensitive to unscaled input data the dimensionless version (13) is used to give

$$\text{Physics loss} = \sum_{\xi \in \{\text{Sample points}\}} \left\| \frac{\mathrm{d}^4 v(\xi)}{\mathrm{d}\xi^4} - 1 \right\|^2 + \text{Loss from boundary conditions.}$$

The loss from the boundary conditions (6) is modeled as follows

$$\text{Boundary loss} = \|v(0)\|^2 + \left\| \frac{\mathrm{d}v}{\mathrm{d}\xi}(0) \right\|^2 + \left\| \frac{\mathrm{d}^2 v}{\mathrm{d}\xi^2}(1) \right\|^2 + \left\| \frac{\mathrm{d}^3 v}{\mathrm{d}\xi^3}(1) \right\|^2.$$

The total loss is then

$$\begin{aligned}
\text{Total loss} &= \text{Physics loss} + \text{Boundary loss} + \text{Regularisation term} && (10)\\
&= \sum_{\xi \in \{\text{Sample points}\}} \left\| \frac{\mathrm{d}^4 v(\xi)}{\mathrm{d}\xi^4} - 1 \right\|^2 + \|v(0)\|^2 + \left\| \frac{\mathrm{d}v}{\mathrm{d}\xi}(0) \right\|^2 + \left\| \frac{\mathrm{d}^2 v}{\mathrm{d}\xi^2}(1) \right\|^2 + \left\| \frac{\mathrm{d}^3 v}{\mathrm{d}\xi^3}(1) \right\|^2 && (11)\\
&+ \lambda \sum_{\theta \in \{\text{PINN weights}\}} \|\theta\|^2. && (12)
\end{aligned}$$

The Setup of the PINN

Here we set up a feed forward neural network of the following structure:

Layer	Nodes	Description
Input Layer	1	The input variable ξ sampled at 100 points between 0 and 1
Hidden Layer	5	Classical hidden layer with tanh activation
Output Layer	1	The output variable $v(\xi)$ at the sample points

Table 2: Architecture of the PINN.

Two solving methods are executed sequentially, i.e.

1. Adaptive Moment estimation (ADAM), and
2. Limited-memory Broyden–Fletcher–Goldfarb–Shanno (LBFGS), afterwards. That is for refinement of the solution.

To train the PINN a set of 100 equidistantly distributed sample points on the interval $[0, 1]$ (including the interval boundaries) is used to propagate forward through the PINN. The setup of the parameters of the ADAM and LBFGS solver is as follows.

Property Name	**Property Value**
Learning rate	0.1
Epochs	200
Loss Function	Mean square error (MSE)
Early stopping	at threshold 10^{-6}
L^2-Regularisation	switched off by default

Table 3: ADAM setup.

Property Name	**Property Value**
Learning rate	0.01
Iterations	300
Loss Function	Mean square error (MSE)
Stopping	10^{-5} for the gradients, 10^{-9} for the MSE
L^2-Regularisation	switched off by default

Table 4: LBFGS setup.

PINN Results

The Euler-Bernoulli equation has solved using the PINN as configured in the section before. The errors -MSE and relative- are moderate and acceptable in some way. Nevertheless, compared to the finite element solution, the errors of the FE method are far below the ones of the PINN method. That confirms just the fact that FEM was designed to solve linear elasticity problems and hence FEM is a very powerful tool tackling such problems. After discussing the results of ODE solving we present a problem where PINN is very useful to use.

Method	**Execution Time (s)**	**RMSE**	**Relative Error**
ADAM	3.2240	8.401149e-02	4.435198e-01
LBFGS	2.7095	9.454326e-06	4.991199e-05
FEM	0.1740	1.863023e-08	9.835413e-08

Table 5: Execution time, RMSE values and relative errors wrt exact solution.

Method	**ADAM**	**LBFGS**	**FEM**	**Exact**
ADAM	-	8.400263e-02	8.401147e-02	8.401149e-02
LBFGS	4.434940e-01	-	9.443193e-06	9.454326e-06
FEM	4.435197e-01	4.985322e-05	-	1.863023e-08
Exact	4.435198e-01	4.991199e-05	9.835413e-08	-

Table 6: Relative errors (lower left) and RMSE values (upper right).

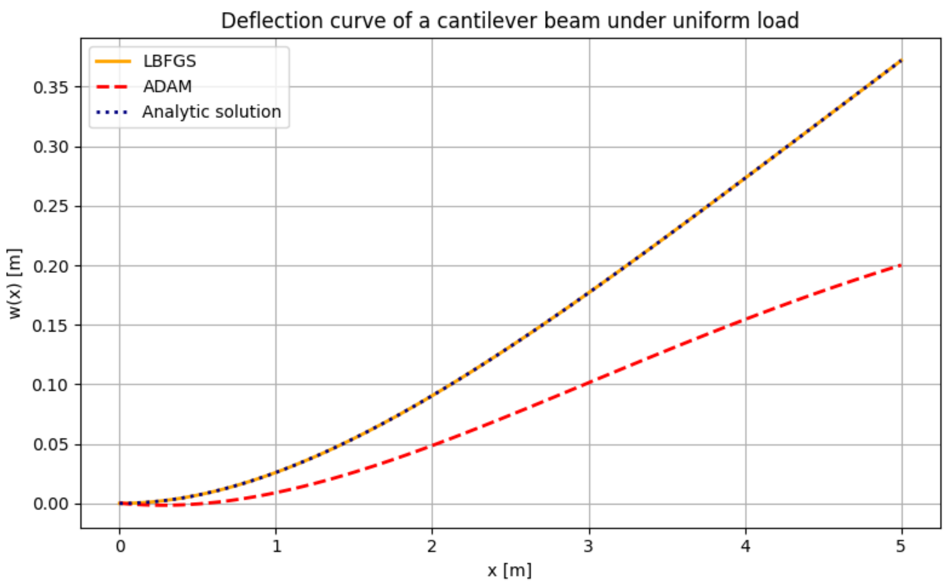

Figure 1: PINN Solution vs exact Solution.

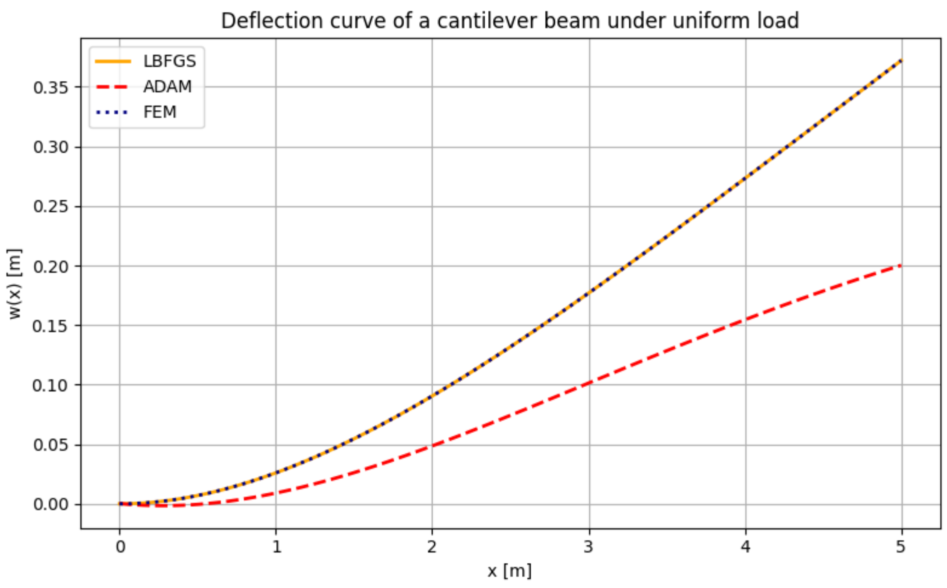

Figure 2: PINN Solution vs FEM Solution.

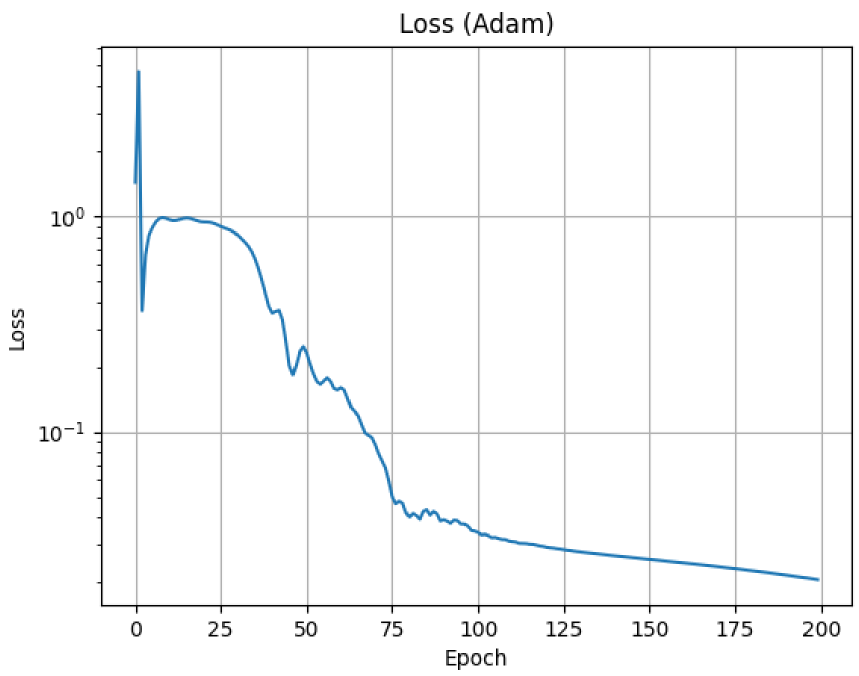

Figure 3: The error of the ADAM calculation vs time/epochs.

4 Parameter Identification for variable Elastic Modulus

PINNs and Parameter Identification

Where PINNs have a poor performance in comparison to FEM in solving linear ODEs they perform very well in many data-driven approaches. One is the prediction of unknown parameters from data, e.g. measurements.

A PINN to predict Elastic Modulus from simulated measurement Data

We now assume that the elastic modulus is an unknown material property due inhomogeneities of the material. Such an inhomogeneity could be rust in an iron beam, for example. On the other hand we assume that the deflection of the beam is known in form of simulated measurements.For this purpose we turn back to the form (1) of the Euler-Bernoulli equation with variable $E = E(x)$:

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}\left(E(x)\cdot I\cdot\frac{\mathrm{d}^2 w(x)}{\mathrm{d}x^2}\right) = q(x) \text{ for } x \in (0, L),$$

with boundary conditions (3) as before. For the numerical solution we use the dimensionless form

$$\frac{\mathrm{d}^2}{\mathrm{d}\xi^2}\left(e(\xi)\frac{\mathrm{d}^2 v(\xi)}{\mathrm{d}\xi^2}\right) = 1 \text{ for } \xi \in (0, 1), \tag{13}$$

subject to the boundary conditions (6) where

$$e(\xi) = \frac{E(x)}{E_{\text{char}}} \text{ and } E_{\text{char}} = \max_{x\in[0,L]} E(x).$$

Following [4] two neural networks are to be trained now:

- A neural network $\mathscr{V}$ to get an approximation $\hat{v}(\xi)$ for the simulated measurements of w.
- A neural network $\mathscr{E}$ to get an estimation $\hat{e}(\xi)$ for the unknown elastic modulus.

These two neural networks will be trained simultaneously. Their architecture is given by the following two tables.

Layer	**Nodes**	**Description**
Input Layer	1	The input variable ξ sampled at 100 points between 0 and 1
Hidden Layer 1	20	Classical hidden layer with tanh activation
Hidden Layer 2	20	Classical hidden layer with tanh activation
Output Layer	1	The output variable $v(\xi)$ at the sample points

Table 7: Architecture of $\mathscr{V}$.

Layer	**Nodes**	**Description**
Input Layer	1	The input variable ξ sampled at 100 points between 0 and 1
Hidden Layer 1	35	Classical hidden layer with tanh activation
Hidden Layer 2	35	Classical hidden layer with tanh activation
Output Layer	1	The output variable $e(\xi)$ at the sample points

Table 8: Architecture of $\mathscr{E}$.

Both networks will be trained using a common loss function. This loss function has a rather complicated structure than (10) for the direct problem:

$$\begin{aligned}\text{Total loss} &= \alpha_1 \cdot \text{Data loss} + \alpha_2 \cdot \text{Physics loss} + \alpha_3 \cdot \text{Boundary loss}\\ &+ \alpha_4 \cdot \text{Loss from gradient terms of } E + \alpha_5 \cdot \text{Regularisation term}\end{aligned}$$

where

$$\begin{aligned}\text{Data loss} &= \sum_{\xi\in\{\text{Sample points}\}} \|\hat{v}(\xi) - v_{\text{measure}}(\xi)\|^2\\ \text{Physics loss} &= \sum_{\xi\in\{\text{Sample points}\}} \left\|\frac{\mathrm{d}^2}{\mathrm{d}\xi^2}\left(\frac{\mathrm{d}^2 v(\xi)}{\mathrm{d}\xi^2}\right) - 1\right\|^2\\ \text{Boundary loss} &= \|v(0)\|^2 + \left\|\frac{\mathrm{d}v}{\mathrm{d}\xi}(0)\right\|^2 + \left\|\frac{\mathrm{d}^2 v}{\mathrm{d}\xi^2}(1)\right\|^2 + \left\|\frac{\mathrm{d}^3 v}{\mathrm{d}\xi^3}(1)\right\|^2\\ \text{Loss from gradient terms of } E &= \sum_{\xi\in\{\text{Sample points}\}} \left\|\frac{\mathrm{d}e(\xi)}{\mathrm{d}\xi}\right\|^2\\ \text{Regularisation term} &= \sum_{\theta\in\{\text{weights of } \mathscr{E}\}} \|\theta\|^2.\end{aligned}$$

For the parameters α_k $k \in \{1,2,3,4,5\}$ the choices are made as in Table refLossFunctionAdjustment.

Again, a combinationof an ADAM solver and a LBFGS solver has been used. The adjustments are given in the tables 10 and 11.

Parameter	**Value**
α_1	100.0
α_2	1.5
α_3	2.0
α_4	1e-5
α_5	1e-3

Table 9: Adjustment values for the loss function.

Property Name	**Property Value**
Learning rate	0.005
Epochs	3000
Loss Function	Mean square error (MSE)
Early stopping	at threshold 10^{-6}

Table 10: ADAM setup.

Property Name	**Property Value**
Learning rate	0.005
Iterations	1500
Loss Function	Mean square error (MSE)
Stopping	10^{-5} for the gradients, 10^{-9} for the MSE

Table 11: LBFGS setup.

With these values the two PINNs $\mathscr{V}$ and $\mathscr{E}$ have been trained using Python and PyTorch. Therefore 100 pairs of random integers - each uniformly distributed between 0 and 100- are generated. Then the first random number of each pair is used to generate a normally distributed noise with zero mean and standard deviation of 0.002506 to be added to the exact solution $v(\xi)$. This will simulate measured data which randomly vary around the exact solution. The second random number sets the seed for the randomly generated initial weights of the two neural networks $\mathscr{V}$ and $\mathscr{E}$. For 99 out of the 100 samples we have the following These results show a very near and acceptable approximation.

Loss	**Value**
Data loss	< 9.27e-6
Physics loss	< 1.42e-5
Boundary loss	< 1.36e-5
Max difference in E	< 16385 Pa
R² value	> 0.993

Table 12: Results for 99 of 100 test samples.

There is one pathological case where the approximation seriously fails: The poor approximation of this case may be easily seen in the following diagram:

Loss	**Value**
Data loss	9.90e-5
Physics loss	1.095e-4
Boundary loss	< 1.640e-2
Max difference in E	< 1.574e11 Pa
R^2 value	> 0.937

Table 13: Results for the pathological case.

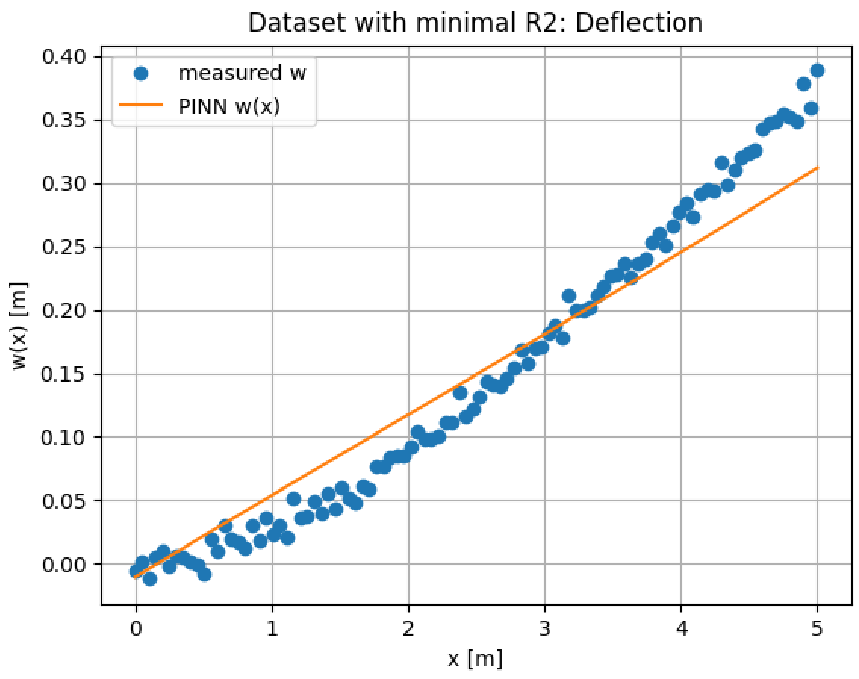

Figure 4: Estimated w and simulated measurements.

Beyond the wrong shape the boundary conditions are obviously not fulfilled. The following chart which shows the exact solution w versus the measured values seems to say that the measured values are not the reason for the poor approximation.

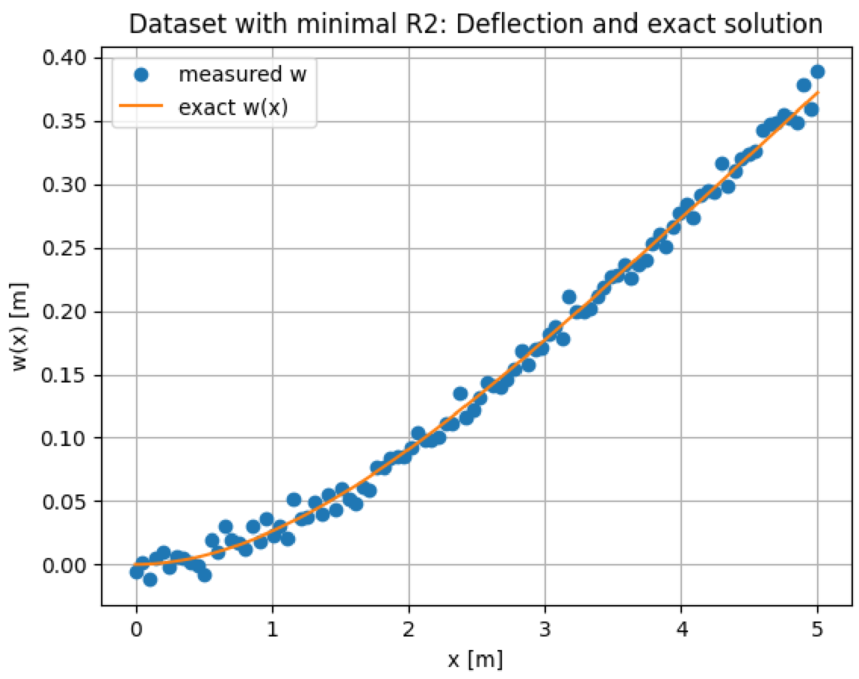

Figure 5: Eaxct w and simulated measurements.

Two of the 99 good samples have the same random seed as the pathological sample and thus the same generated measured values. It follows that these simulated measurements are not the reason for the poor approximation quality. Furthermore, choosing another random initial weights for the PINN training will solve the problem. Choosing another solver than ADAM or skipping ADAM to train only with LBFGS could be another solution.

5 Conclusion and Outlook

This short case study using PINNs for calculating solutions and parameters of the 1D Euler-Bernoulli equation leads to the following conclusions:

1. Calculating a solution of the Euler-Bernoulli equation using classical finite elements is much faster and more precise than using a PINN. Of course finite element methods have been just developed to solve linear mechanical problems and thus they are optimized for that purpose. The nonlinear and nested structure of a neural network may be not adequate or optimal for that purpose.

2. Estimating the elastic modulus from simulated deflection values is a rather data-driven problem and the PINN approach fits far better there. With a careful adjustment of the network architecture, the solver adjustments and the loss contributions it was possible to calculate results with very satisfactory precision in case of 99 samples.

3. For one single sample the estimation of elastic modulus failed producing results with poor accuracy. The reason is an appropriate choice of initial weights of the PINN and can be solved using another initial set up.

4. The very fine and careful setup of the network architecture, the solver settings and the loss balancing may be improved by varying these parameters in a systematic way. It is desirable to reduce the number of epochs without loosing precision. This probably can be achieved by another balancing of the loss contributions, a modification of the PINN architecture and better settings for the solver.

5. The calculations have been done in Python using Pytorch. To reduce the calculation time an implementation in C++ would be more adequate. One of the most performant C++ frameworks is OpenNN which is far faster than Tensorflow or Pytorch. Unfortunately, OpenNN does not really support the use of PINNs, especially there is no automatic differentiation and no possibility to access the loss function of the hidden layers via the OpenNN API. The other C++ Machine Learning framework, MLPack, has similar difficulties with PINN and does not support PINNS very well.

6. A way to implement a PINN study in C++ could be the use of the Pytorch C+ +API. That may be as not as fast as OpenNN or MLPack but faster than any Python implementation.

References

[1] K.-H. Hoffmann and G. Witterstein, *Mathematische Modellierung: Grundprinzipien in Natur- und Ingenieurwissenschaften*, Mathematik Kompakt, Birkhäuser Basel, 1st ed. (corrected reprint), 2014.

[2] R. Rojas, *Theorie der neuronalen Netze: Eine systematische Einführung*, 1st ed. (corrected reprint 1996), Springer-Lehrbuch, Springer-Verlag Berlin, 1993.

[3] H. Stöcker, *Taschenbuch mathematischer Formeln und moderner Verfahren*, 3rd rev. and ext. ed., Harri Deutsch Verlag, Frankfurt am Main, 1995.

[4] C. Teloli, A. Titarelli, A. Guerrieri, and S. Vidoli, *Solving linear beam dynamics with neural operators: A comprehensive study*, Mechanics of Materials, vol. 190, 104226, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888327024010884

[5] Code samples used in this work: https://github.com/Haasrobertgmxnet/BeamEquation