Technische Universität Berlin

Institut für Mechanik

FG Strukturmechanik und Strukturberechnung

Machine Learning in Computational Mechanics

Task Sheet 3

Contents

1 Documentation

The purpose of the given script is to approximate a displacement field U of an observed 2D domain problem based on prescribed boundary conditions and applied forces with the help of a Deep Energy Method (DEM).

The script can be broken down into following key steps:

1. Import required libraries such as *Matplotlib* and *PyTorch*.
2. Initialize desired settings for the torch model such as the default floating point, number of threads, and the device used for later major calculations.
3. Define the problem parameters for the observed 2D domain such as the grid dimensions and the applied boundary conditions.
4. Define the parameters for the material modelling.
5. Generating the meshgrid for the observed 2D domain with the help of *torch.meshgrid()*.
6. (Optional) Create a function that generates a scatter plot of the discretized domain of the solid material.
7. Create a function that computes the strain at each point in the observed 2D domain with respect to the displacement field U and the reference configuration X as given in Equation (1)(1).

$$\varepsilon = \frac{1}{2}(\nabla U + \nabla U^T) \tag{1}$$

8. Create a function that computes the stress tensor σ and the strain energy density ψ at each point of the observed 2D domain as given in Equation (2) and (3).

$$\sigma_{ij} = \mathbb{C}_{ijkl}\varepsilon_{kl} \tag{2}$$

$$\psi = \varepsilon : \sigma \tag{3}$$

9. Create a function that integrates the strain energy density ψ for each point on the observed 2D domain as given in Equation (4). For the numerical integration purpose, a trapezoidal method is implemented in the script.

$$\Psi = \iint \psi \, dydx \tag{4}$$

10. Create a function that calculates the divergence loss of the stress tensor σ to evaluate the static equilibrium condition as given in Equation (5).

$$\nabla\sigma = 0 \tag{5}$$

11. Create a function to calculate the energy of the traction energy T on the boundaries by integrating the multiplication of the applied force and the corresponding displacement, as shown below. Again, the trapezoidal method is used in the script for the numerical integration.

$$T = \int F_{right} \cdot U_x \, \mathrm{dy} \tag{6}$$

12. Create a function that computes the boundary losses based on the prescribed boundaries of the 2D domain. This should ensure that the prescribed boundary conditions are satisfied.
13. Create a function to calculate and plot the strain ε, the strain energy density ψ, and the displacement field U on each point of the observed 2D domain. This function should also

visualize the approximated solution of the displacement field $U_{predicted}$ and compute the losses of the model either using Physics-Informed Neural Network (PINN) or using the so-called Data-Driven technique. In the Data-Driven technique, the loss between the predicted displacement U and the analytical solution of the displacement $U_{analytic} = U_{target}$ is calculated using MSE loss function. With data-driven training, it is qualitatively capable of approximating the solution and could help to find bugs if the Neural Network was trained data-driven first and then physics-informed afterwards.

14. Define the architecture of the Neural Network.
15. Create a function that initializes the Neural Network parameters randomly.
16. Create a flexible, custom activation function for the Neural Network.
17. Initialize a class for the Neural Network and define its forward pass function.
18. Define a variable which decides whether a data-driven technique should be used.
19. Initialize the model with the predefined class for Neural Network. Select a suitable optimization algorithm and its learning rate. Lastly train the model using the generated (training) dataset.

The 2D domain problem and its boundary conditions given in the script can be seen in Figure 1. This observed domain will then be represented as a so-called meshgrid, as seen in Figure 2, so that one can calculate the mechanical characteristics at each point in the domain. This approach is pretty similar to Finite Element Method (FEM). The number of grids in x- and y-axis is defined as follows.

$$X = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \Delta x \cdot L_x + 1 \\ \Delta y \cdot L_y + 1 \end{bmatrix} \tag{7}$$

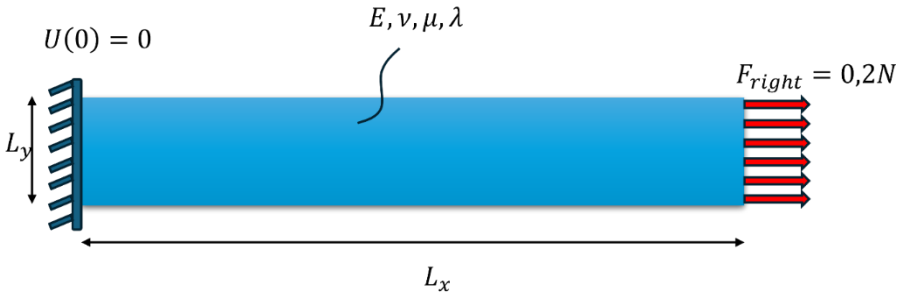


Figure 1: Observed 2D domain problem and its boundary conditions.

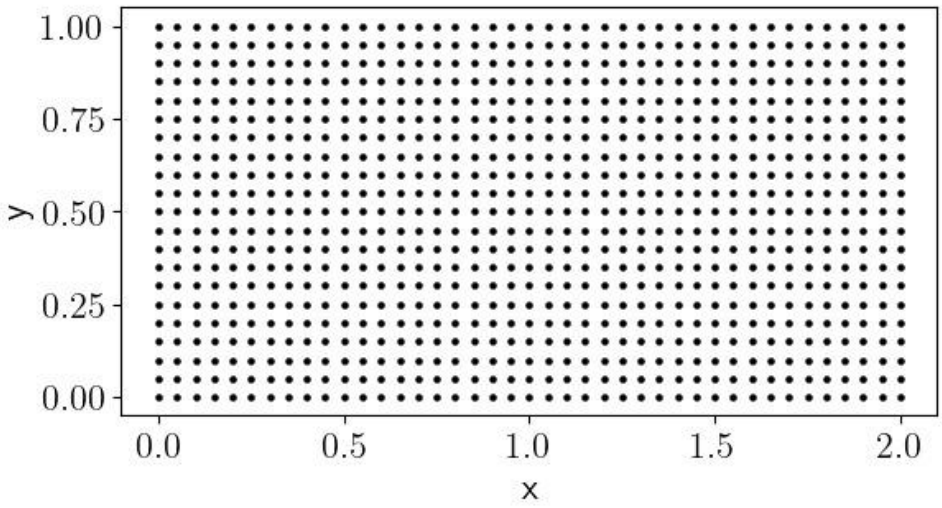


Figure 2: Generated sample points (Type1) of the observed 2D domain based on the parameters in The traction energy T is computed in this task as stated in Equation xx,

Before the training process, one must define the Machine Learning (ML) model using the corresponding class by also defining the architecture of the model. To achieve a good result for this specific task, a suitable optimizer must be selected based on its performance in the training and test process. In this case, the optimizer can be considered as a hyperparameter. In addition, it is also possible that the training in the beginning is data-driven and after some training iterations, the model is trained with physics-informed method. The model parameters are updated after each training iterations based on the computed gradients. By default, the random initialization of the NN parameters is turned off and will be considered in the next chapter. Figure 3 shows an overview of the selected Neural Network architecture, which consists of an input layer with 2 neurons, 1 hidden layer with 16 neurons, and an output layer with 2 neurons. The generated sample points are fed into the input of the Neural Network and will then be processed internally. As an output, the predicted displacement tensor in both x- and y-axis is generated. Essential (hyper-)parameters for generating the 2D domain problem, defining the training function, and constructing the NN architecture are shown in Table 1.

Table 1.

Furthermore, alongside the predefined Young's modulus and Poisson's ratio, the Lamé constants, assuming a plane strain, for the material modelling are calculated as stated in [1].

$$\mu_{\text{strain}} = \frac{E}{2 \cdot (1+\nu)} \tag{8}$$

$$\lambda_{strain} = \frac{E \cdot \nu}{(1+\nu) \cdot (1-2\nu)} \tag{9}$$

Based on the observed 2D domain and the prescribed boundary conditions, the analytical solution for the displacement tensor $U_{target} = U_{analytical}$ is calculated as stated in Equation (18). This analytical solution will then be compared to the predicted displacement tensor U in the Data-Driven technique.

$$U_{target} = \begin{bmatrix} U_x \\ U_y \end{bmatrix} = \begin{bmatrix} \frac{x}{L_x} \cdot \Delta L \\ -\frac{x}{L_x} \cdot \nu \cdot \left(\frac{y}{L_y} \cdot 0{,}5\right) \cdot \Delta L \end{bmatrix} \tag{10}$$

with

$$\Delta L = \frac{F_{right} \cdot L_x}{E}$$

The traction energy T is computed in this task as stated in Equation xx,

Before the training process, one must define the Machine Learning (ML) model using the corresponding class by also defining the architecture of the model. To achieve a good result for this specific task, a suitable optimizer must be selected based on its performance in the training and test process. In this case, the optimizer can be considered as a hyperparameter. In addition, it is also possible that the training in the beginning is data-driven and after some training iterations, the model is trained with physics-informed method. The model parameters are updated after each training iterations based on the computed gradients. By default, the random initialization of the NN parameters is turned off and will be considered in the next chapter. Figure 3 shows an overview of the selected Neural Network architecture, which consists of an input layer with 2 neurons, 1 hidden layer with 16 neurons, and an output layer with 2 neurons. The generated sample points are fed into the input of the Neural Network and will then be processed internally. As an output, the predicted displacement tensor in both x- and y-axis is generated. Essential (hyper-)parameters for generating the 2D domain problem, defining the training function, and constructing the NN architecture are shown in Table 1.

Table 1: (Hyper-)parameter used in the script for generating 2D domain problem, defining the training function, and constructing the Neural Network architecture.

Material modelling			
Nr.	Variable name	Selection	Note
1	device	CPU	Device used for major tensor calculations with maximum 8 number of threads.
2	L_x	2	Length of the problem domain in x-axis.
3	L_y	1	Length of the problem domain in y-axis.
4	Δx	20	Step size in x-axis.
5	Δy	20	Step size in y-axis.
6	$U(0)$	0	Boundary condition on the left side of the domain.
7	F_{right}	0.2	Applied force on the right side of the domain.
8	F_{upper}	0	Applied force on the upper side of the domain.
9	F_{lower}	0	Applied force on the lower side of the domain.
10	E	1	Young's modulus.
11	ν	0.3	Poission's ratio.
12	μ	0.3846153846153846	Lamé constant.
13	λ	0.5769230769230769	Lamé constant.
Training function and NN parameters			
14	criterion	nn.MSELoss(reduction="sum")	The selected loss function. In the given script, the MSE loss function is used.
15	hidden_dim	16	Number of neurons in the hidden layer.

16	input_dim	2	Number of neurons in the input layer.
17	output_dim	2	Number of neurons in the output layer
18	Activation function	Tanh or myPow	myPow is a custom activation function, which is given as follows, $f(x) = x^2 + n \cdot x$ (11) with n as the learnable slope.
19	data_driven	False	Initial value of the data_driven.
20	epochs	10000	Training iterations.
21	optimizer	Adam optimizer or LBFGS	Optimization algorithm.

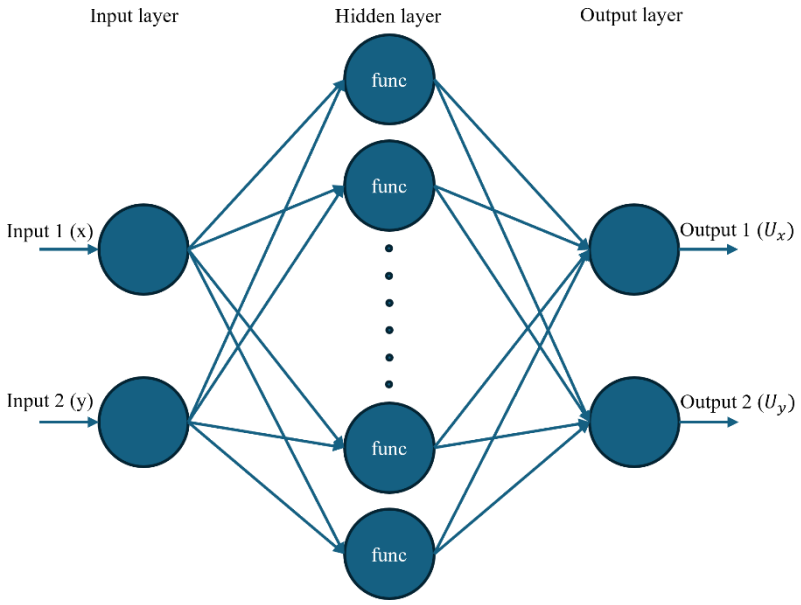


Figure 3: Neural Network architecture used in DEM algorithm.

2 Analysis

The purpose of this task is to analyse the influences of the parts in the loss function, when the model is not in the data-driven training. The provided script is then validated by comparing it with the analytical solution. This could trigger the detection of errors and mistakes in the script.

The loss function is mainly a combination of three different losses, namely[1]

- Strain energy and traction energy loss, which ensures the energy principle of mechanics.
- Boundary conditions, which ensures that the displacements or applied stresses are consistent with prescribed boundary conditions.
- Divergence loss, in which the internal equilibrium in the observed 2D domain is ensured.

As a reference, the analytical solution shown in Figure 4 will be compared with each mentioned loss or a combination of those losses as shown in Table 2. Furthermore, the implementation of a random initialization of the NN parameters was also put in a consideration for the analysis. For this specific task,

[1] Source: ChatGPT.

1000 training iterations are executed instead of 10000, since the model already should reach its convergence point. The predicted solutions, absolute errors of the solutions, and the loss curves are shown in the next following Figures.

Table 2: Combination of the loss terms

Type	Strain energy and traction energy loss	Boundary conditions loss	Divergence loss
100	Yes	No	No
010	No	Yes	No
001	No	No	Yes
110	Yes	Yes	No
101	Yes	No	Yes
011	No	Yes	Yes
111	Yes	Yes	Yes

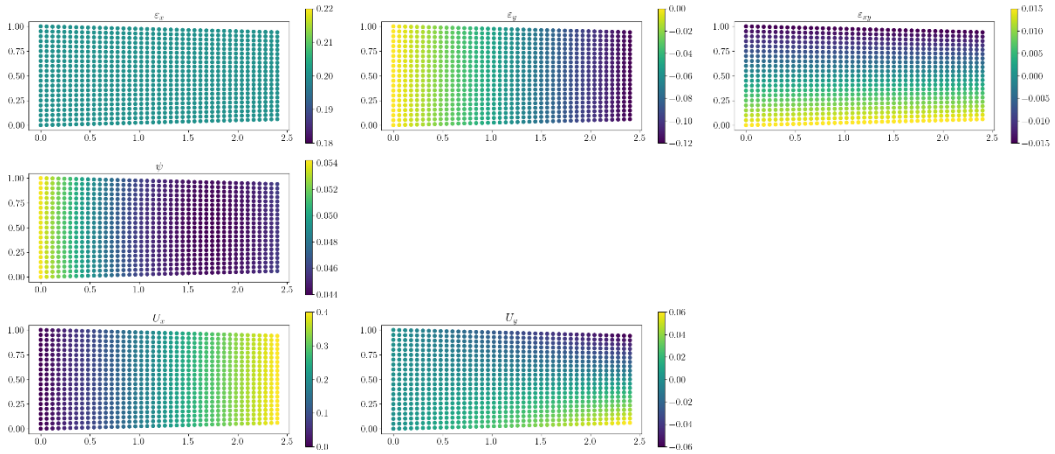


Figure 4: Analytical solutions for the observed 2D domain problem with a plane strain distribution.

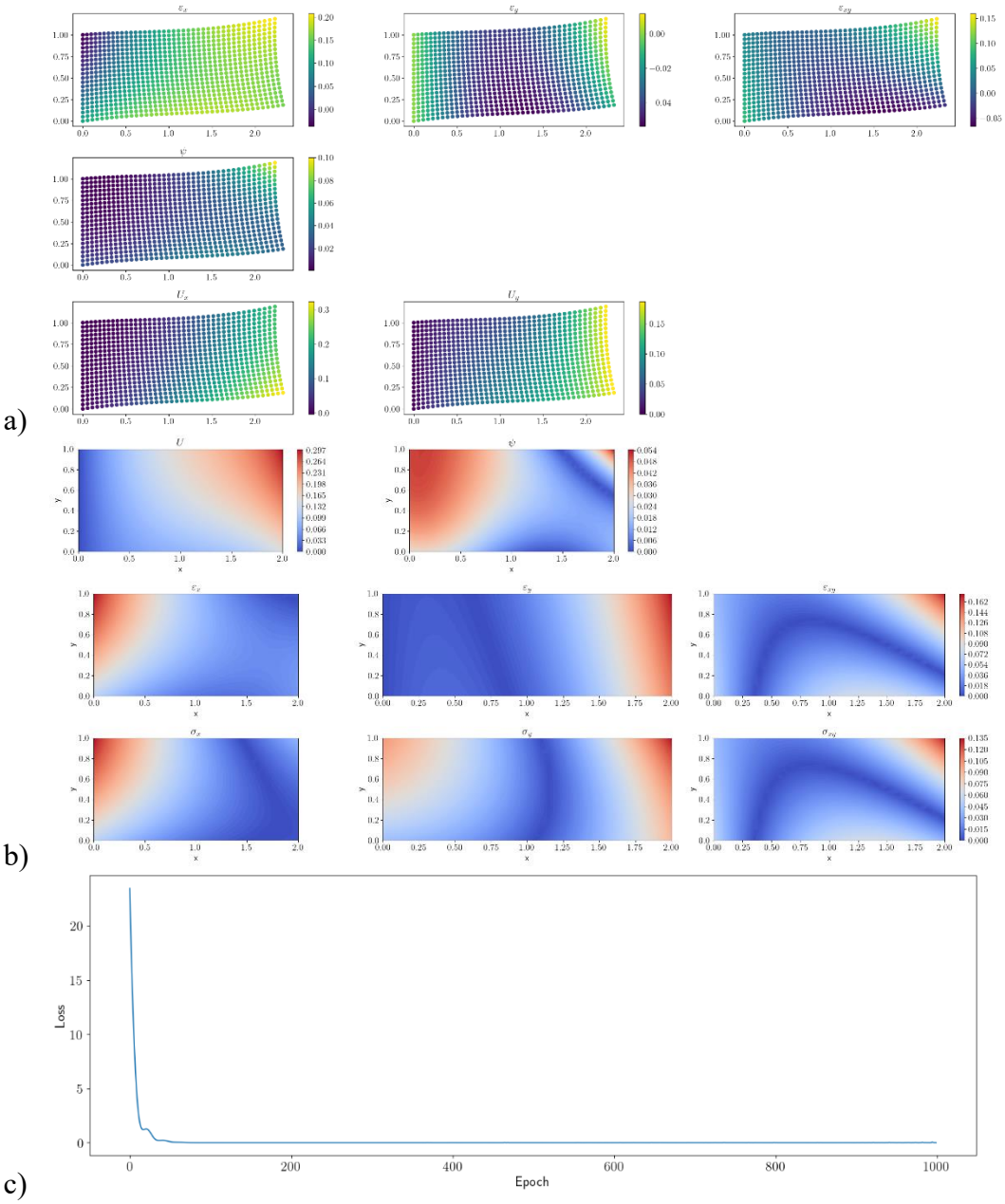


Figure 5: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution; c) the loss curve for the observed 2D domain problem with a plane strain distribution and loss function Type100.

As seen above, the strain energy and traction energy loss, compared to other terms of the loss function, attempted on constraining the model to have a "realistic" deformation pattern as possible. In this case, the left boundary condition is still satisfied, i.e. the beam is still fixed on the left side. However, there are still relatively a lot of inaccuracies on the predicted displacement and strain energy density. The model also tends to have a small initial loss compared to other loss type.

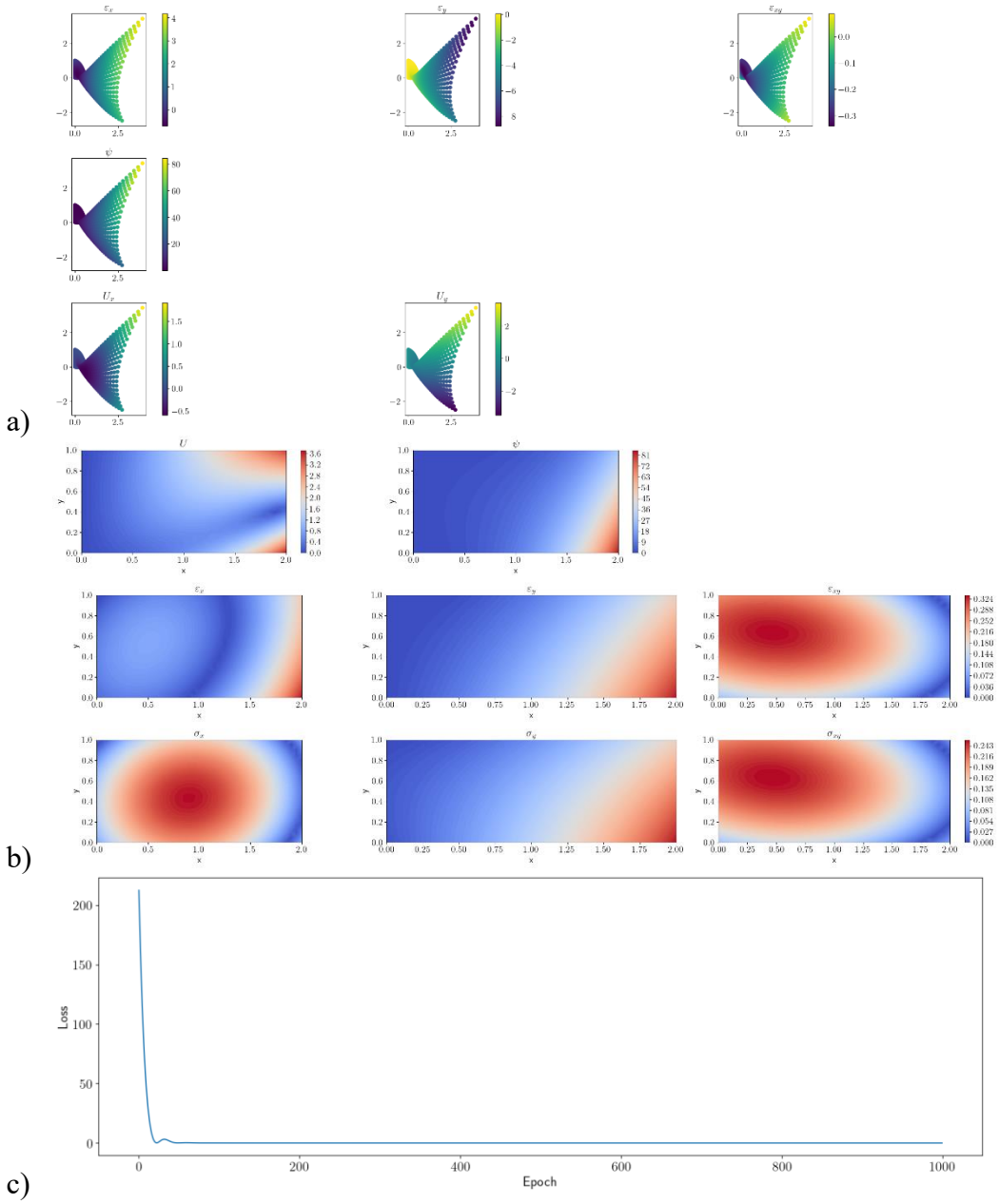


Figure 6: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution; c) the loss curve for the observed 2D domain problem with a plane strain distribution and loss function Type010.

As seen above, the boundary condition loss seems to have an “unrealistic” deformation. However, the left boundary condition is still satisfied, i.e. the beam is still fixed on the left side. On the other hand, the right boundary condition is not satisfied. Moreover, there are still a lot of inaccuracies on the predicted strain and the stress components on each sample points.

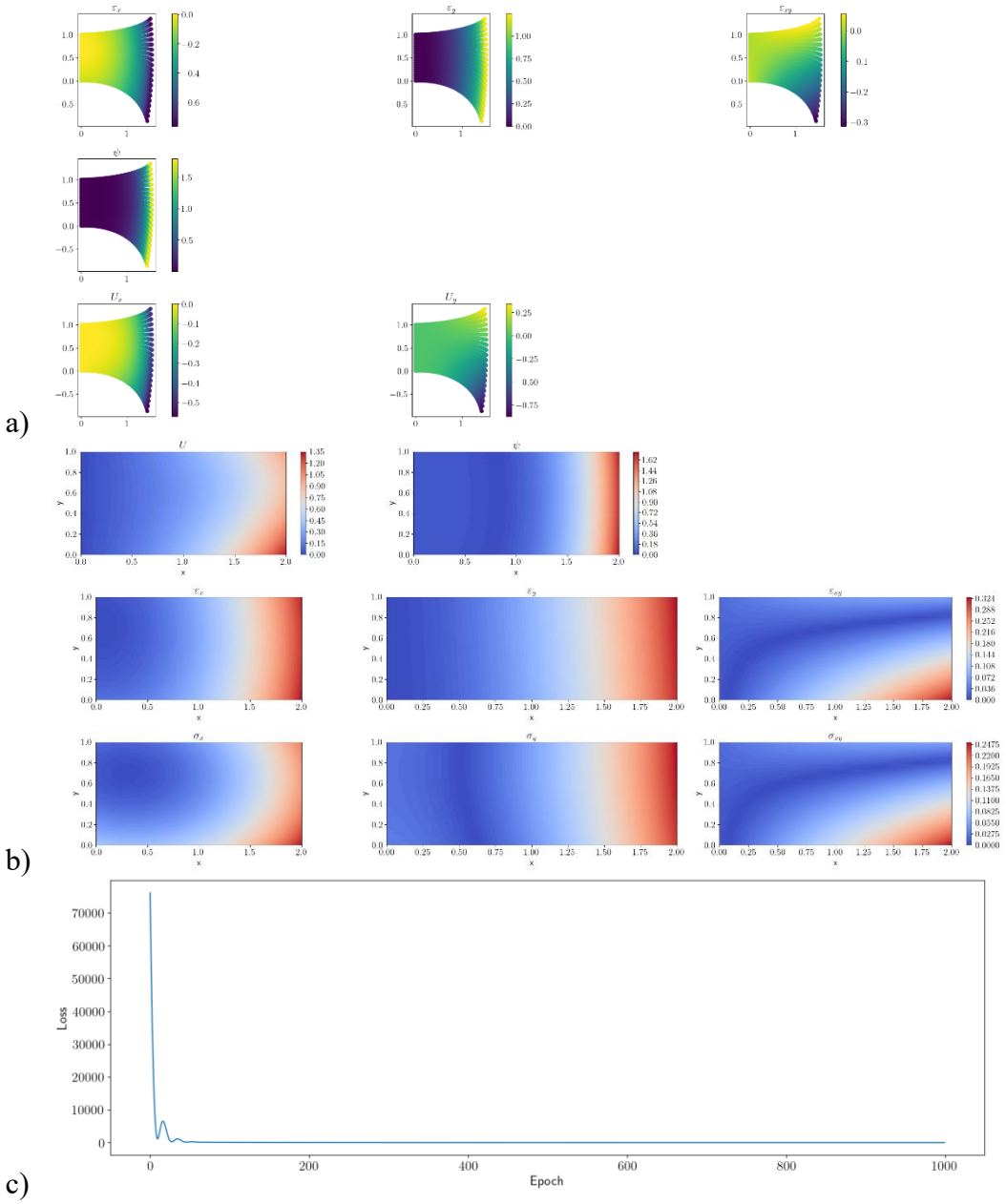


Figure 7: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution; c) the loss curve for the observed 2D domain problem with a plane strain distribution and loss function Type001.

As seen above, the divergence loss seems to have an “unrealistic” deformation, mainly because the loss function considered only the internal static equilibrium. The combination of the loss terms can be seen in the figures below.

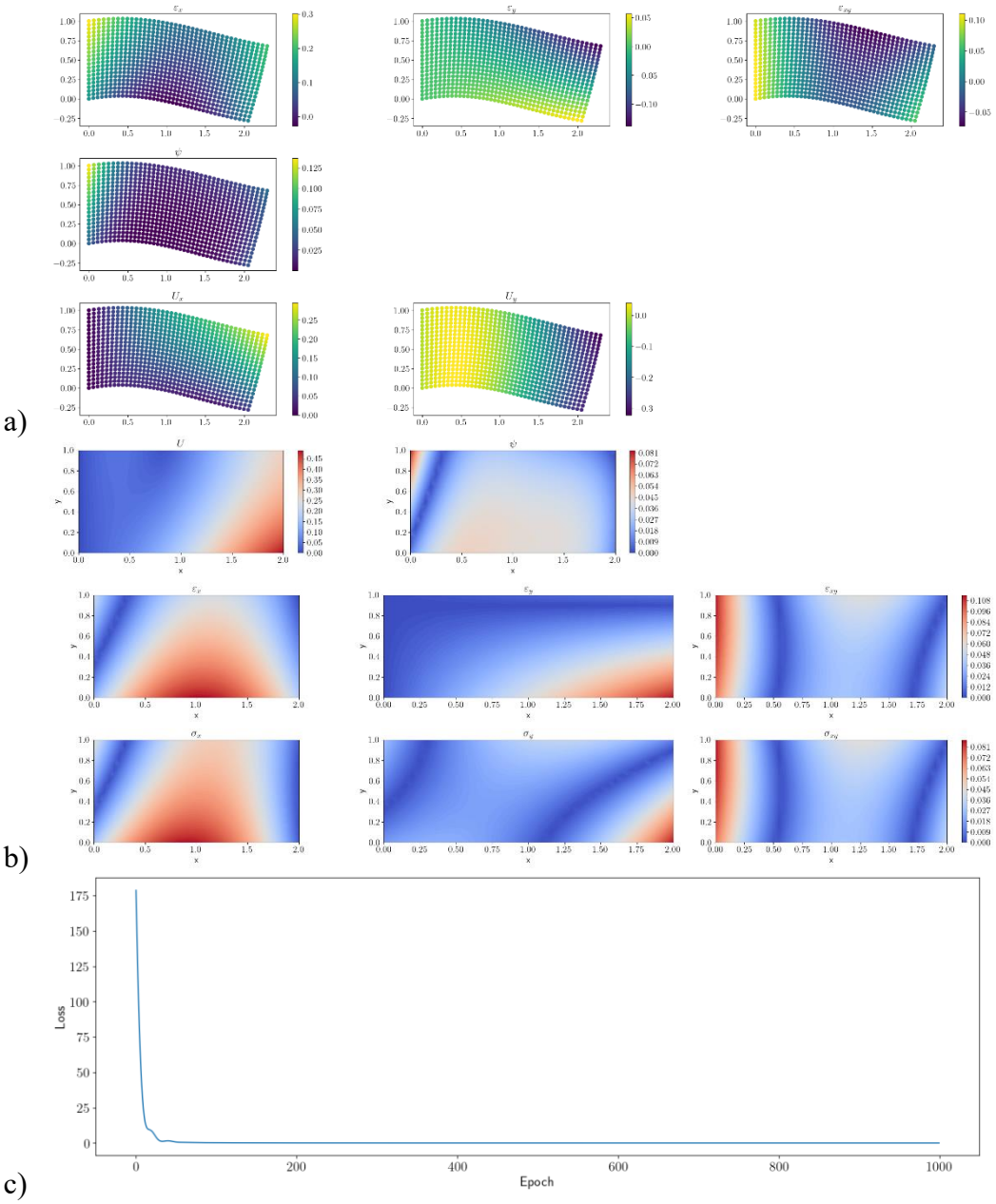


Figure 8: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution; c) the loss curve for the observed 2D domain problem with a plane strain distribution and loss function Type110.

As seen above, the combination of the strain energy and traction energy loss and the boundary condition loss attempted on constraining the model to have a “realistic” deformation pattern. In this case, the left boundary condition is still satisfied, i.e. the beam is still fixed on the left side. Moreover, the combination of this losses offers a relatively accurate prediction on the strain energy density. The model also tends to have a small initial loss compared to other loss type.

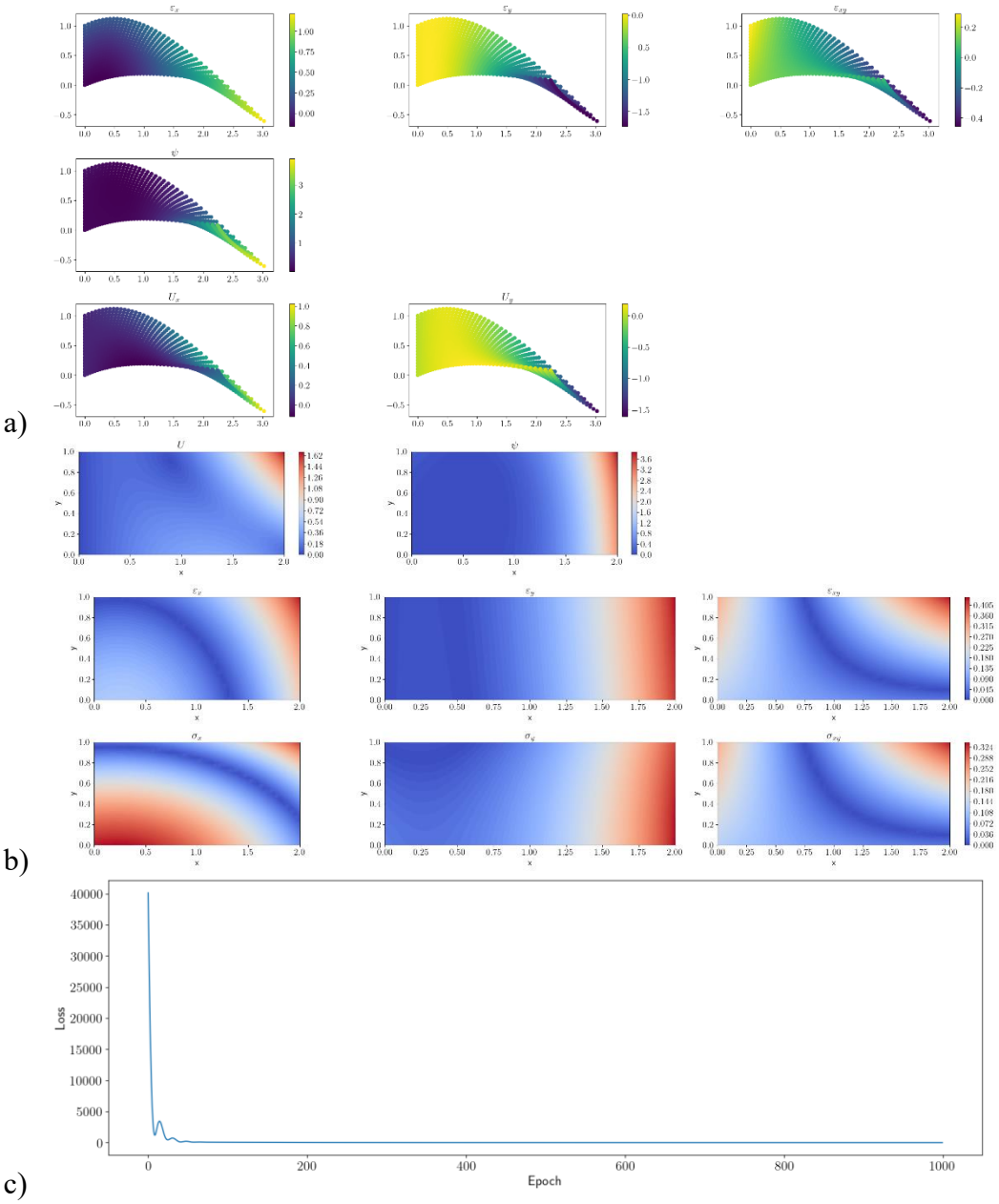


Figure 9: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution; c) the loss curve for the observed 2D domain problem with a plane strain distribution and loss function Type101.

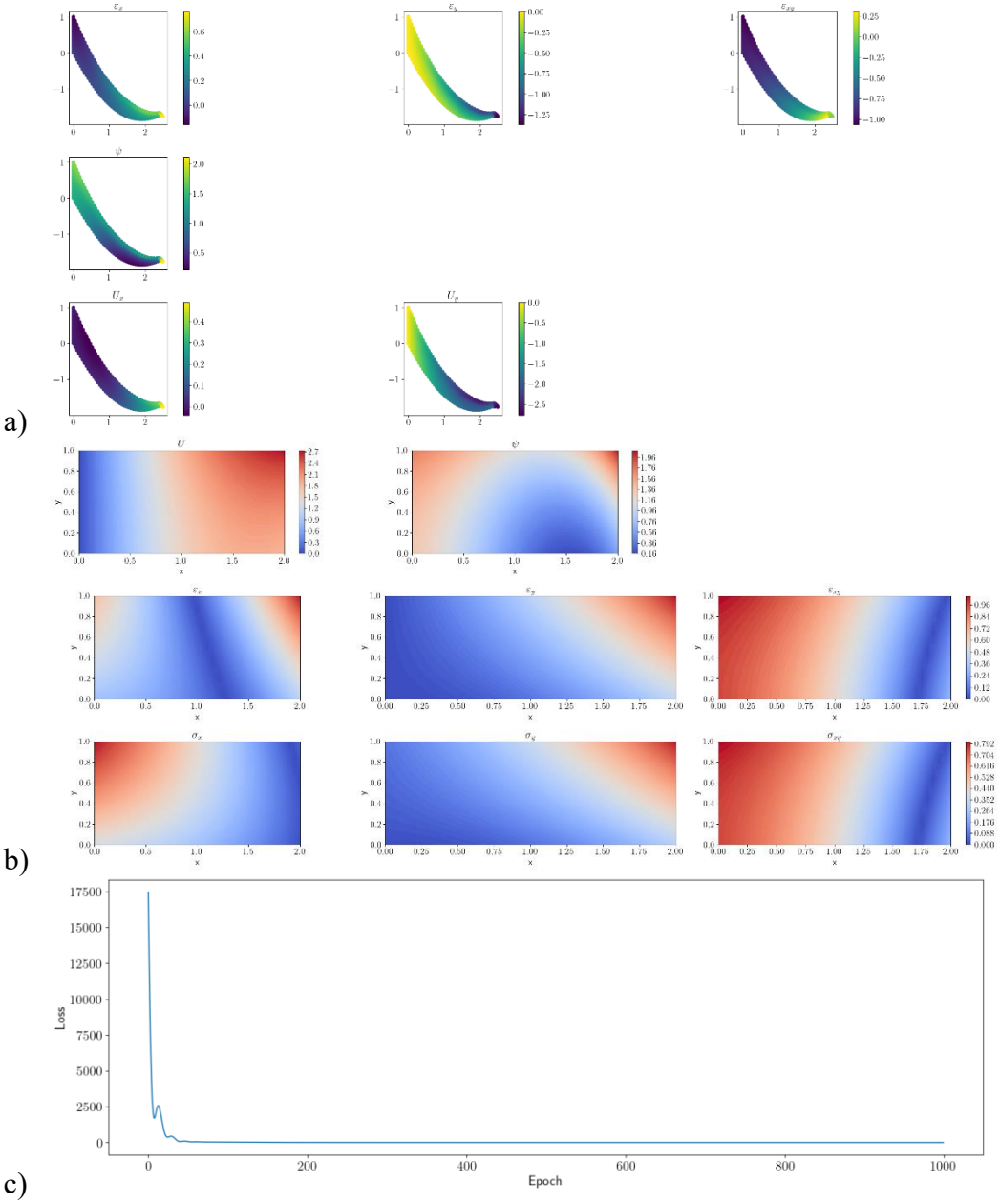


Figure 10: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution; c) the loss curve for the observed 2D domain problem with a plane strain distribution and loss function Type011.

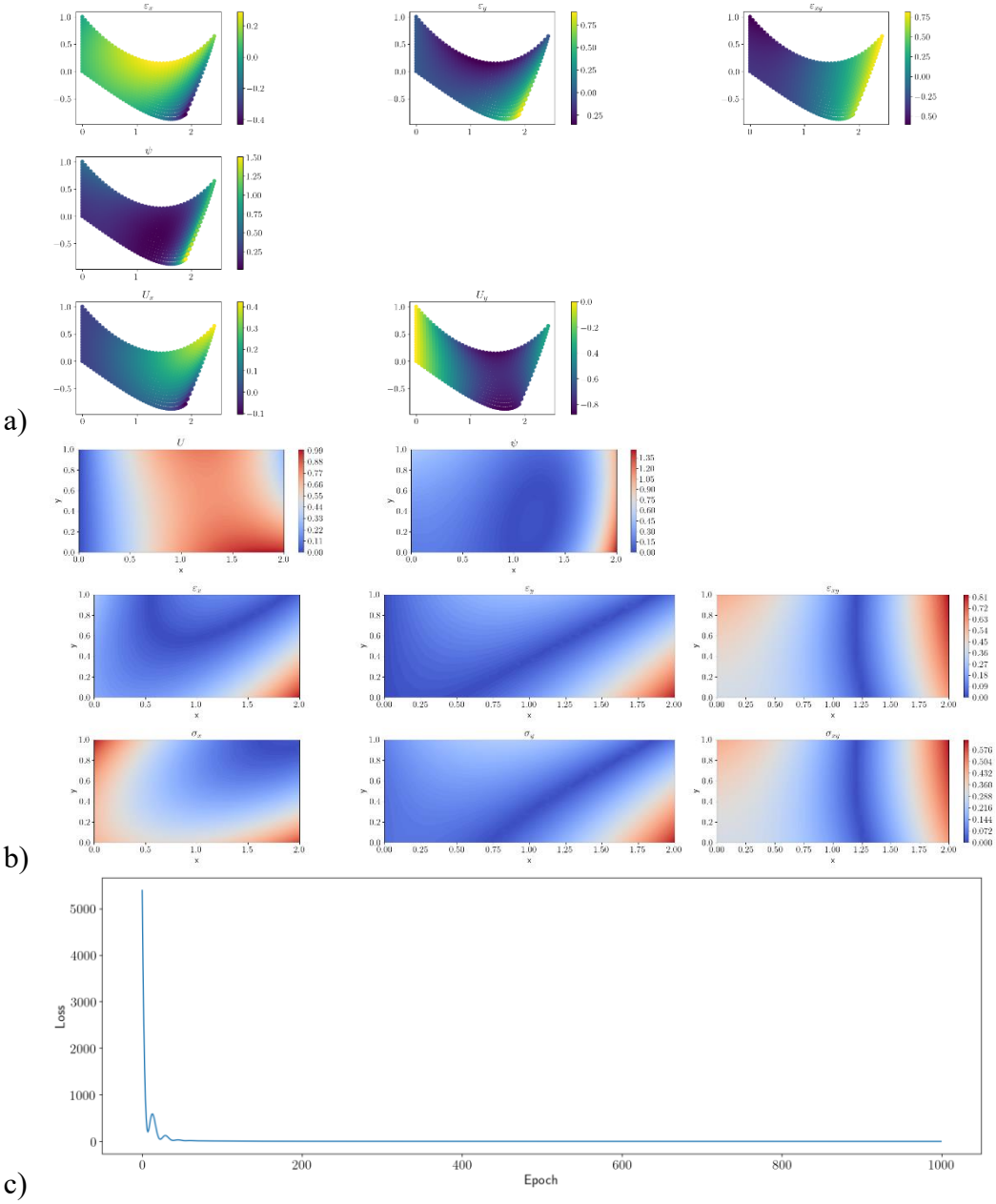


Figure 11: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution; c) the loss curve for the observed 2D domain problem with a plane strain distribution and loss function Type111.

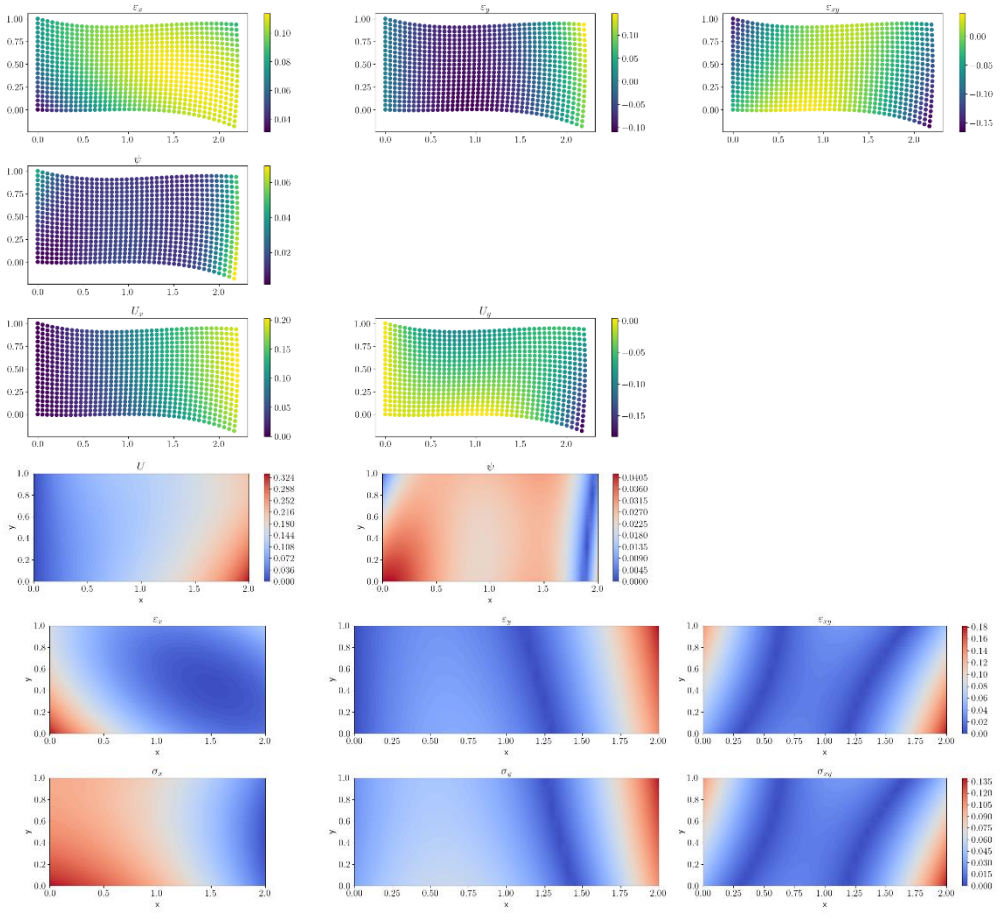
Based on the given results above, the model still could not give qualitatively nor quantitatively sufficient predictions. This relatively large inaccuracy may be due to the inappropriate selection of the 2D model simplifications, i.e. assuming a plane strain. The observed domain problem is a 2D beam, thus a very thin object loaded in-plane is assumed. This makes a plane stress distribution is more suitable for this task, as stated in [2] and [3]. In this case, the Lamé constants implemented in the code become as shown below.

$$\mu_{\text{stress}} = \frac{E}{2 \cdot (1 + \nu)} \tag{12}$$

$$\lambda_{stress} = \frac{E \cdot \nu}{(1 + \nu) \cdot (1 - 2\nu)} \tag{13}$$

The following figures illustrate the training results of the modified model distribution after 1000 training iteration. The selection of a plane stress distribution slightly improves qualitatively the predictions, although quantitatively speaking, it is still far from the analytical solution. Moreover, hyperparameter optimization, implementation of higher-order formula and enhancement of the amount and distribution of the sample point could also lead to a more precise predictions of the model, which are going to be the focus of the next chapter. Note that in this chapter, a test function is not yet implemented.

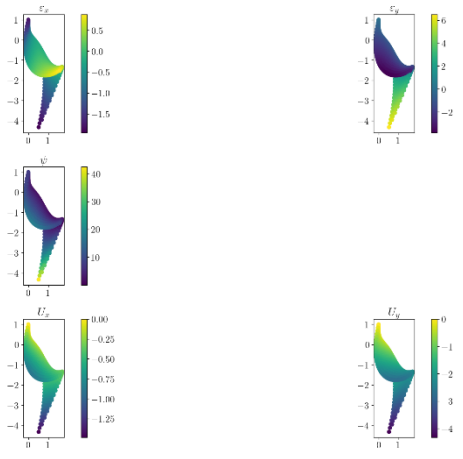
a)

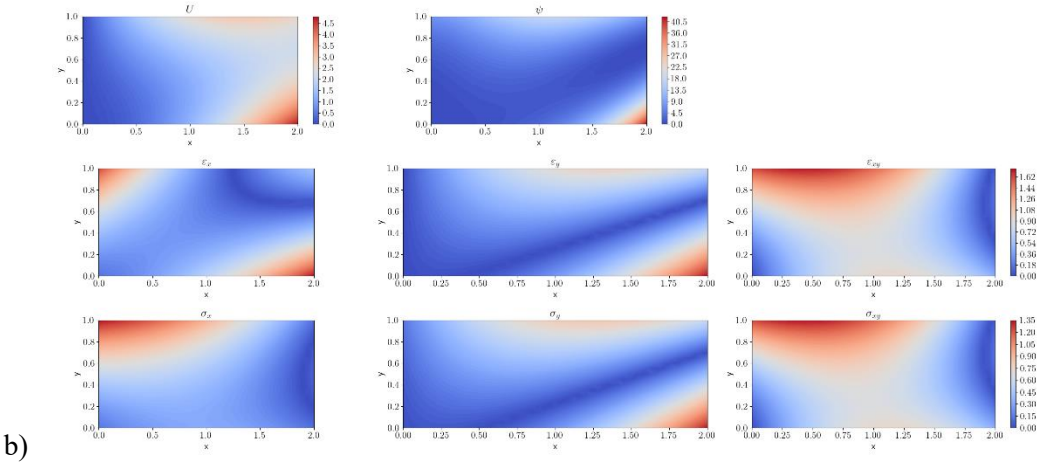


b)

Figure 12: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution for the observed 2D domain problem with a **plane stress distribution** and loss function Type100.

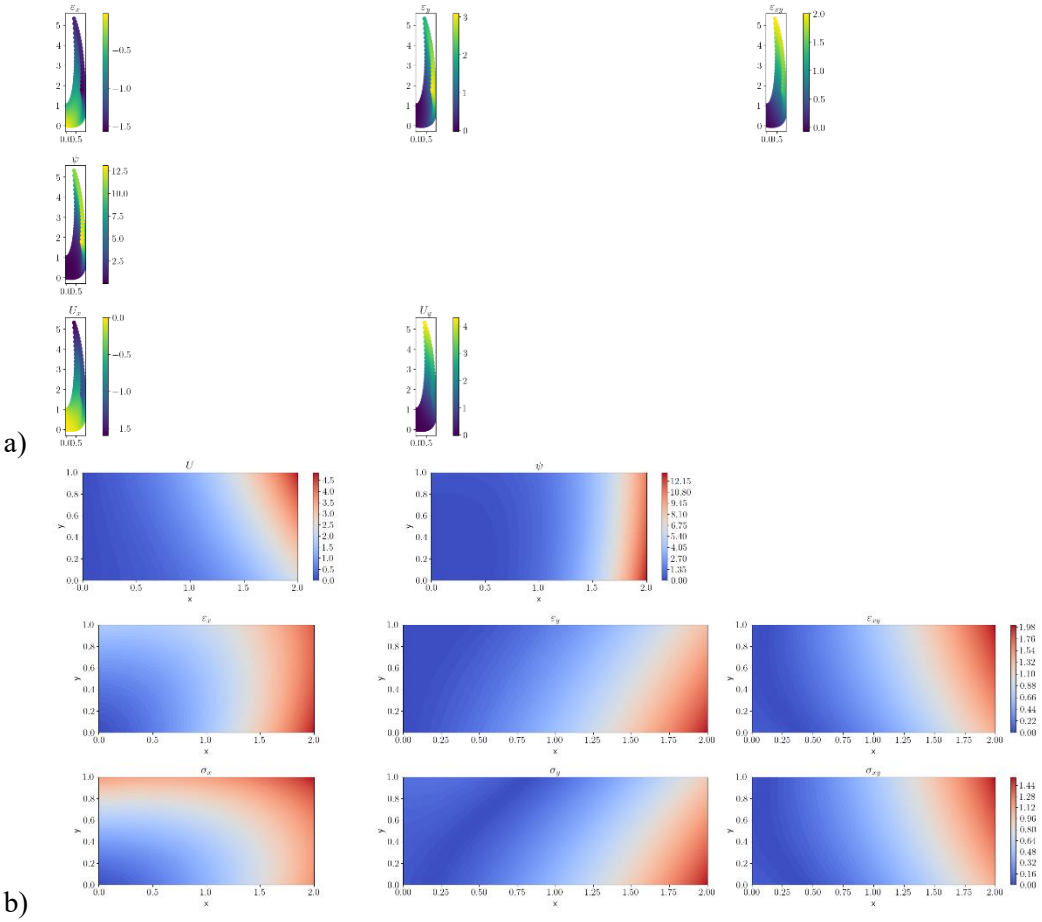
a)





b)

Figure 13: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution for the observed 2D domain problem with a **plane stress distribution** and loss function Type010.



a)

b)

Figure 14: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution for the observed 2D domain problem with a **plane stress distribution** and loss function Type001.

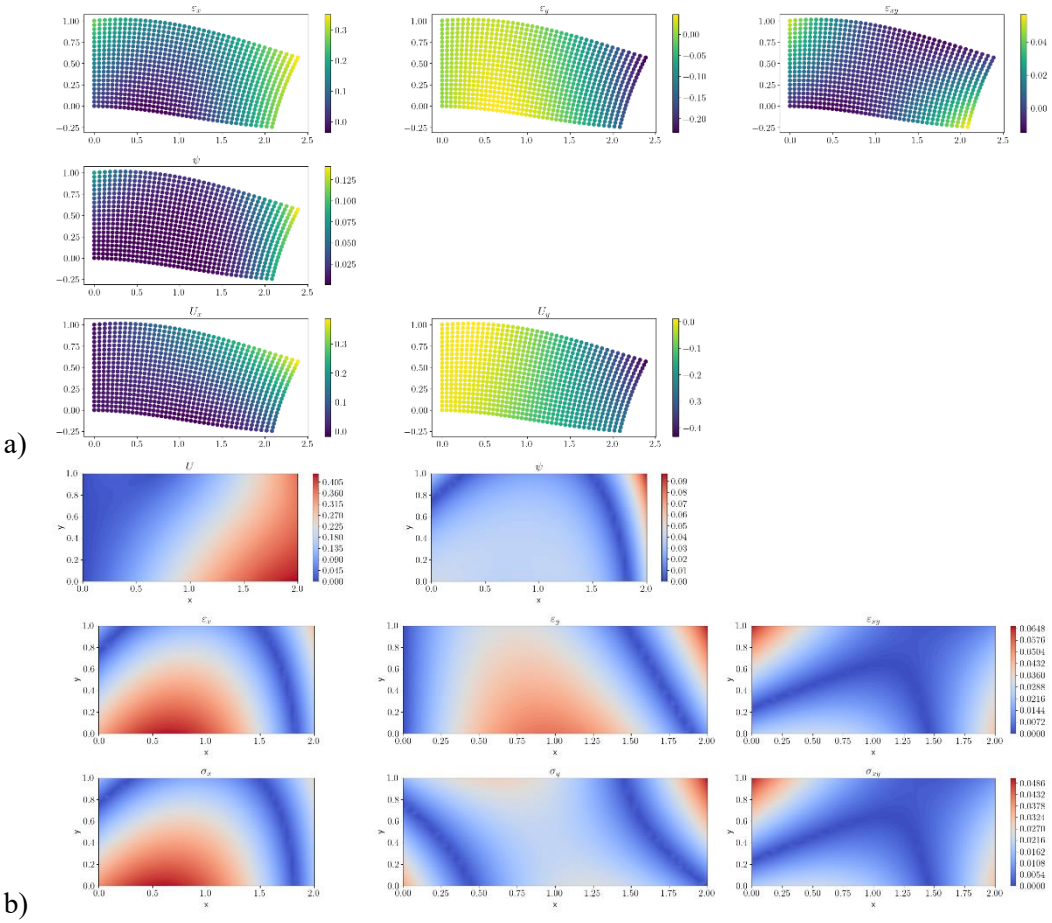


Figure 15: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution for the observed 2D domain problem with a **plane stress distribution** and loss function Type110.

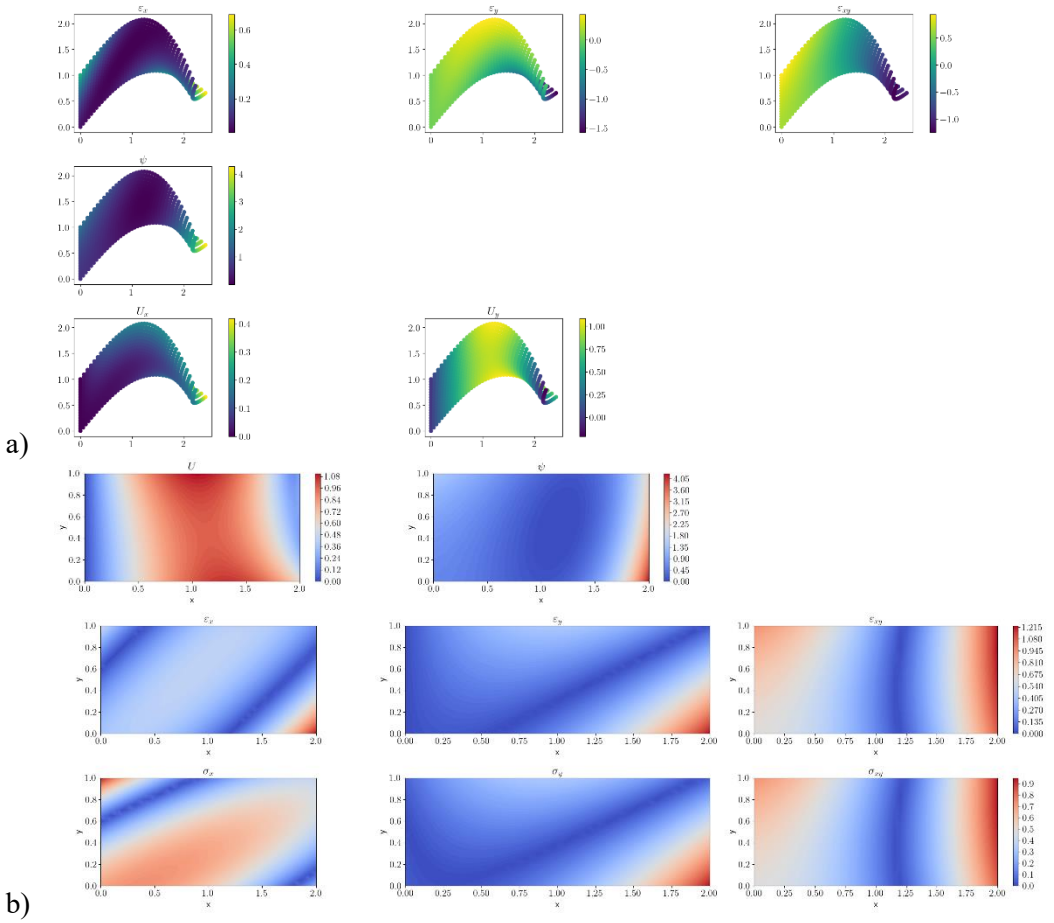
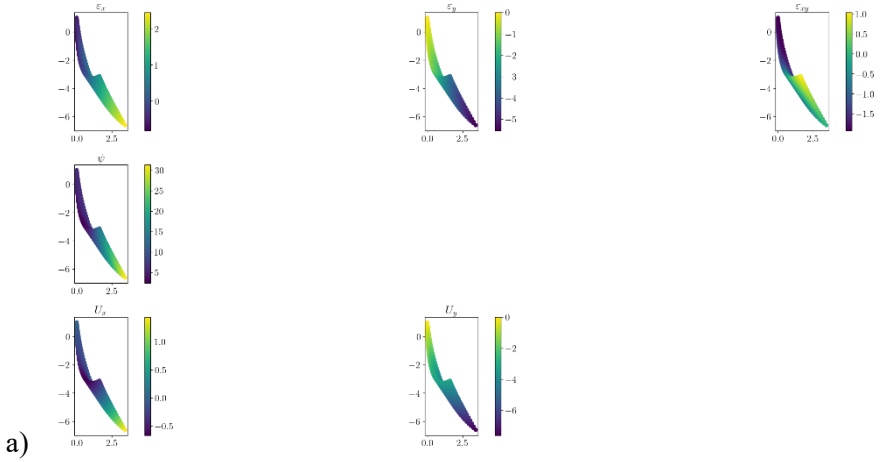
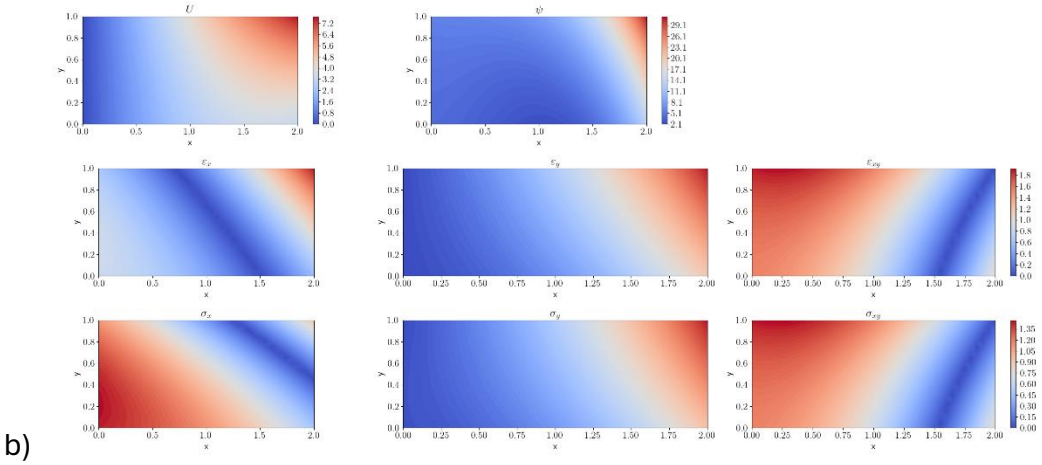


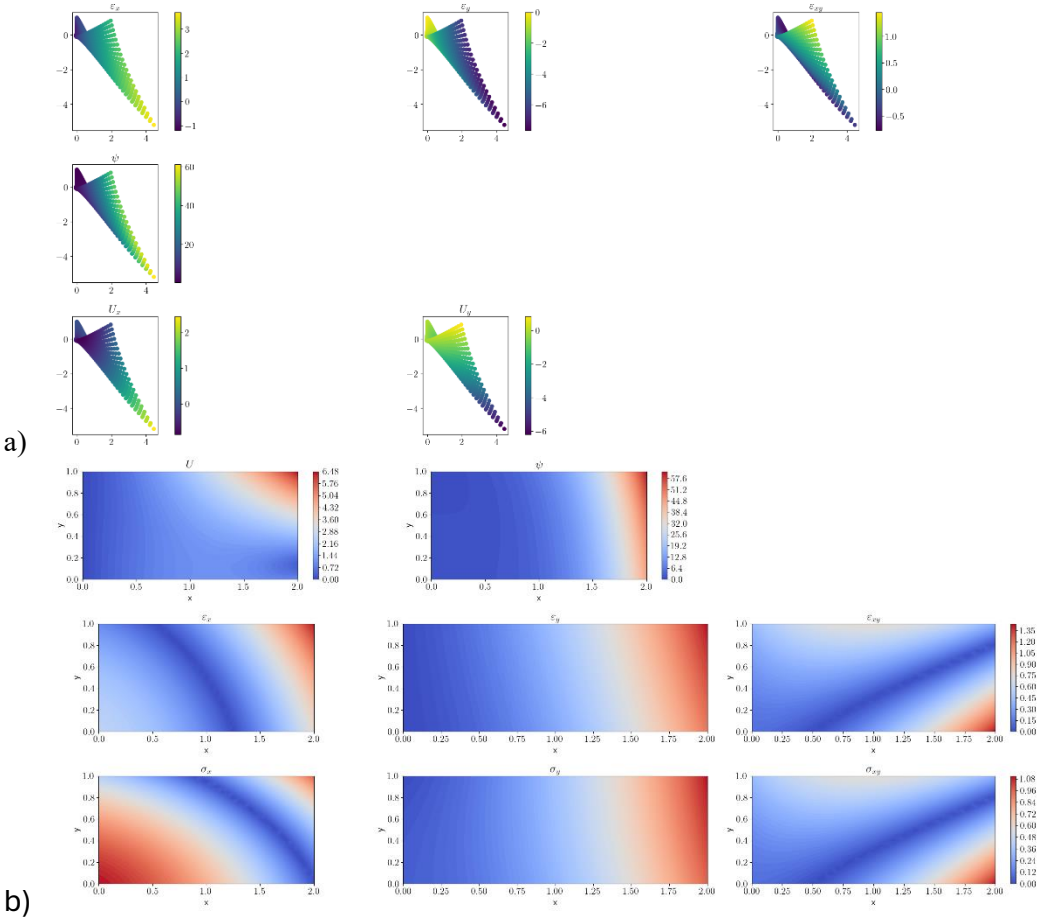
Figure 16: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution for the observed 2D domain problem with a **plane stress distribution** and loss function Type101.





b)

Figure 17: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution for the observed 2D domain problem with a **plane stress distribution** and loss function Type011.



a)

b)

Figure 18: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution for the observed 2D domain problem with a **plane stress distribution** and loss function Type111.

3 Optimization

The purpose of this task is to optimize the hyper parameters, implement a higher-order formula, in this case Simpson's rule, improve the amount and distribution of sample points, increment scheme for external loads, and improve the zero-load boundary conditions in order to achieve a better accuracy of the training results.

Refinement of the sample points

As alternatives, the modified code script offers other sampling methods than uniform sampling, such as:[2]

1. Non-uniform grid sampling (Type2), in which the density of the sample points is increased near the boundaries.
2. Modified non-uniform grid sampling (Type3), where more points near boundaries, applied force regions are generated. This offers finer sampling near boundaries and coarser in the centre.

Furthermore, one should also consider increasing the sampling density by increasing the step size in both axes, also Δx and Δy. Figure 19 illustrates the sampling distributions of the non-uniform grid sampling (Type2) and modified non-uniform grid sampling (Type3).

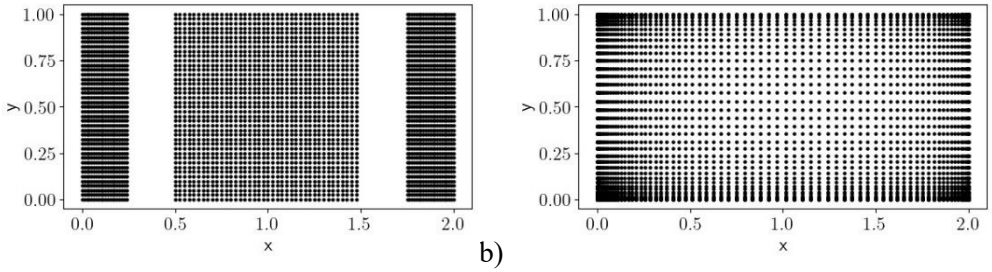


Figure 19: Sampling distributions of a) Type2 and b) Type3 with $L_x = 2,\ L_y = 1,\ \Delta x = \Delta y = 40$

Implementation of Simpson's rule for numerical integrations

Simpson's rule, as shown in Equation (18), is known as an approximation for definite integrals, which offers a higher-order convergence order. It was set to replace the numerical integration with trapezoidal rule found in the code script, i.e. for calculating the integration of the strain energy density Ψ and calculating the traction energy T. This was done directly by using a function called integrate.*simpson()* from SciPy library [4].

$$\int_a^b f(x)dx \approx \frac{b-a}{6} \cdot \left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right] \tag{14}$$

Implementation of increment scheme for external load

It is also considered to implement the increment scheme for external load, since it should stabilize the training process. In this case, instead of defining the applied force(s) as a constant, it linearly increased during the training, starting from the half of its value as the initial value. For every 10 training iterations, the applied force(s) increases until it reaches the desired value. Theoretically, this method offers a more stable training and a better generalization, since the model learns for varied applied force(s). Furthermore, if this should be put in a consideration, it could mimic a real-world problem, since realistically the loads are gradually increased and not make a sudden impact.

[2] Generated with the help of ChatGPT.

Extension of missing stress components in the loss function of boundary condition

The provided code implements some zero-load boundary conditions in the loss function, but not all the stress components are analysed consistently on all boundaries and edges. The implemented and its extended stress components are shown in Table 3. The left boundary is constrained by a Dirichlet boundary condition, i.e. the displacement must be zero. This indirectly implies that the stress components σ_{xx} and σ_{yy} must be zero. Additionally, no shear traction should appear on the left boundary to prevent any slipping. On the right boundary, it must be ensured that the normal stress σ_{xx} fits the applied force there. Moreover, no shear traction σ_{xy} and normal stress σ_{xx} should also exist on the right boundary. Since there are no applied forces on the lower and upper boundary, the normal stress σ_{yy} and σ_{xy} must be zero. In this case, there is no need to constrain the normal stress σ_{xx} on the x-axis. Now, the losses of each boundary should be considered in the boundary condition loss function to improve the prediction accuracy.

Table 3: The stress components and its constraint based on the boundary conditions.

Left boundary	Right boundary	Lower boundary	Upper boundary
Dirichlet B.C.	Neumann B.C.		
Implemented stress components			
$\sigma_{xx} = 0$	$\sigma_{xx} = F_{right}$	$\sigma_{yy} = 0$	$\sigma_{yy} = 0$
$\sigma_{yy} = 0$	$\sigma_{xy} = 0$	$\sigma_{xy} = 0$	$\sigma_{xy} = 0$
Stress components, that are still missing and should be implemented			
$\sigma_{xy} = 0$	$\sigma_{yy} = 0$	-	-

Modification in the integral computation of the traction energy

So far, the traction energy is computed based on Equation (18), i.e. by integrating the multiplication of the applied force and the corresponding displacement. One possible way to improve the prediction accuracy is by modifying how the traction energy is computed. The modified computation is stated in Equation (18), since theoretically the traction is calculated from the stress and not from the force boundary condition. Again, the integration is approximated using the Simpson's rule, as stated in the beginning of this chapter.

$$\mathrm{T_y} = \int \sigma_{xx} \cdot U_x \, \mathrm{dy} \tag{15}$$

Hyperparameter optimization

To conduct a hyperparameter optimization, a so-called method called *Grid Search* is implemented, in which a systemic search is conducted through a prescribed set of hyperparameters in order to find the best combination of hyperparameters that optimizes the model's performance. The selected grid search is shown in Table 4. In total, there are 144 possible combinations of hyperparameter selections. After some running, it is only considered to use Adam optimizer as the optimization algorithm and use only 1 hidden layer to spare some time. The other optimizations, which already mentioned in this chapter, are also considered in the Grid Search.

Table 4: Hyperparameter selections

Learning rate (lr)	0.001	0.005	0.01	0.02
Optimization algorithm	Adam	Adamax	SGD	LBFGS
Neurons in hidden layer	8	16	32	-
Number of hidden layers	1	2	3	-
Loss factor	[1,1,0]	[1, 0, 1]	[0, 1, 1]	[1, 1, 1]

Here are the results for the hyperparameter selections:

1) Learning rate of 0.02, 32 neurons in hidden layer, and the loss combination Type011 best fit for the model with Adam optimizer, Simpson's rule integral approximator, linear elasticity model, and a non-uniform grid sampling Type1.
2) Learning rate of 0.02, 16 neurons in hidden layer, and the loss combination Type011 best fit for the model with Adam optimizer, Simpson's rule integral approximator, linear elasticity model, and a non-uniform grid sampling Type2.
3) Learning rate of 0.01, 32 neurons in hidden layer, and the loss combination Type011 best fit for the model with Adam optimizer, Simpson's rule integral approximator, linear elasticity model, and a non-uniform grid sampling Type3.

a)

b)

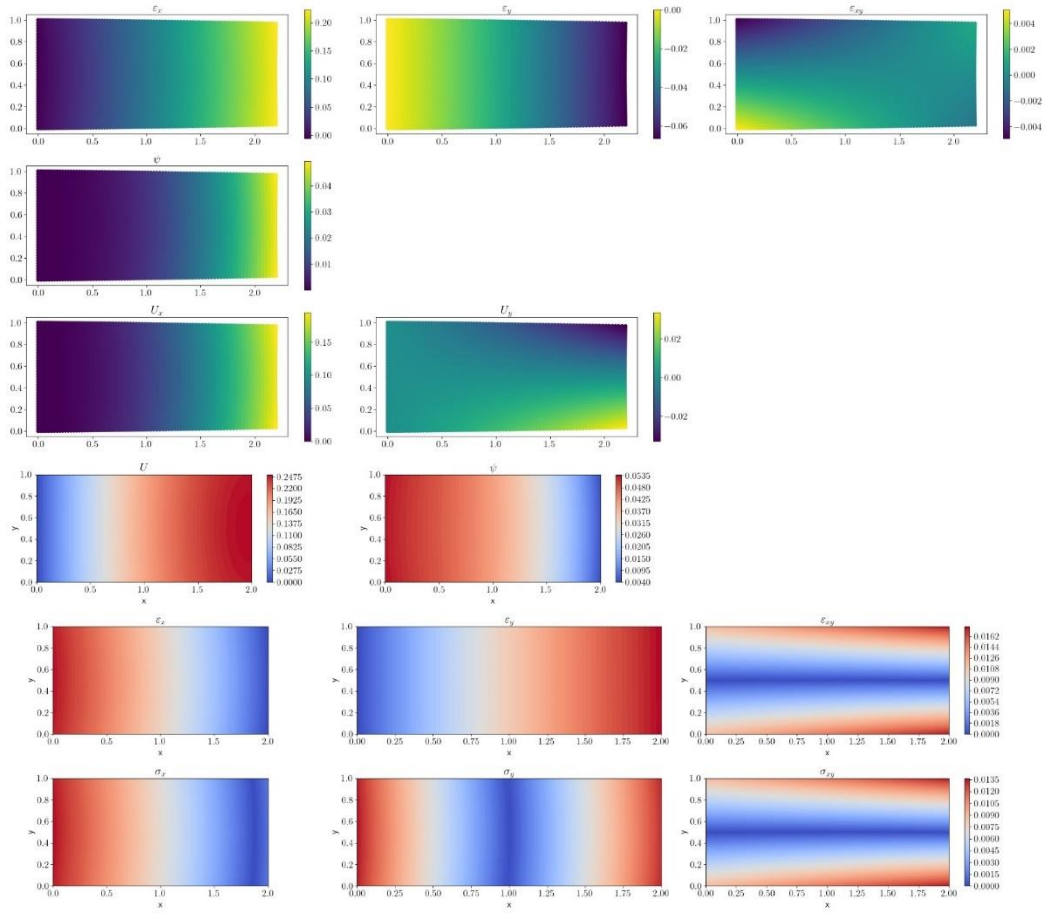


Figure 20: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution for the observed 2D domain problem from 2)

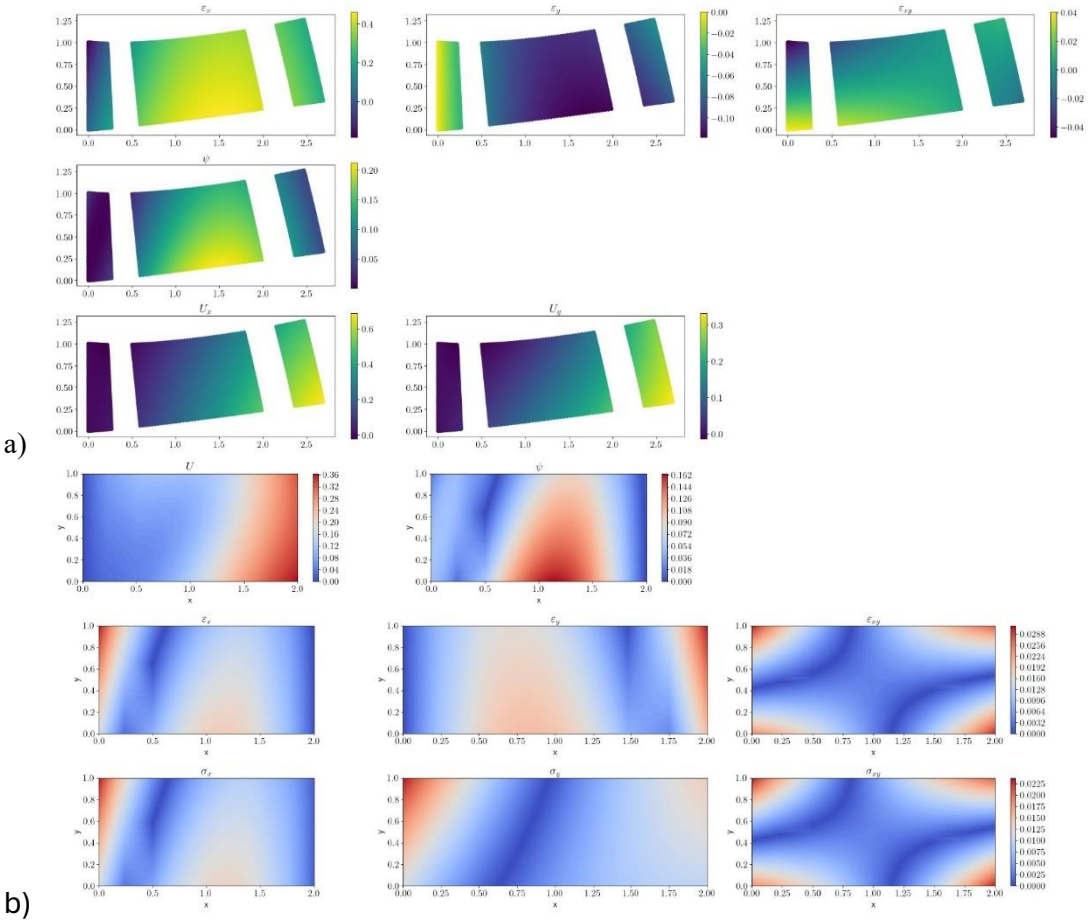


Figure 21: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution for the observed 2D domain problem from 2)

c)

d)

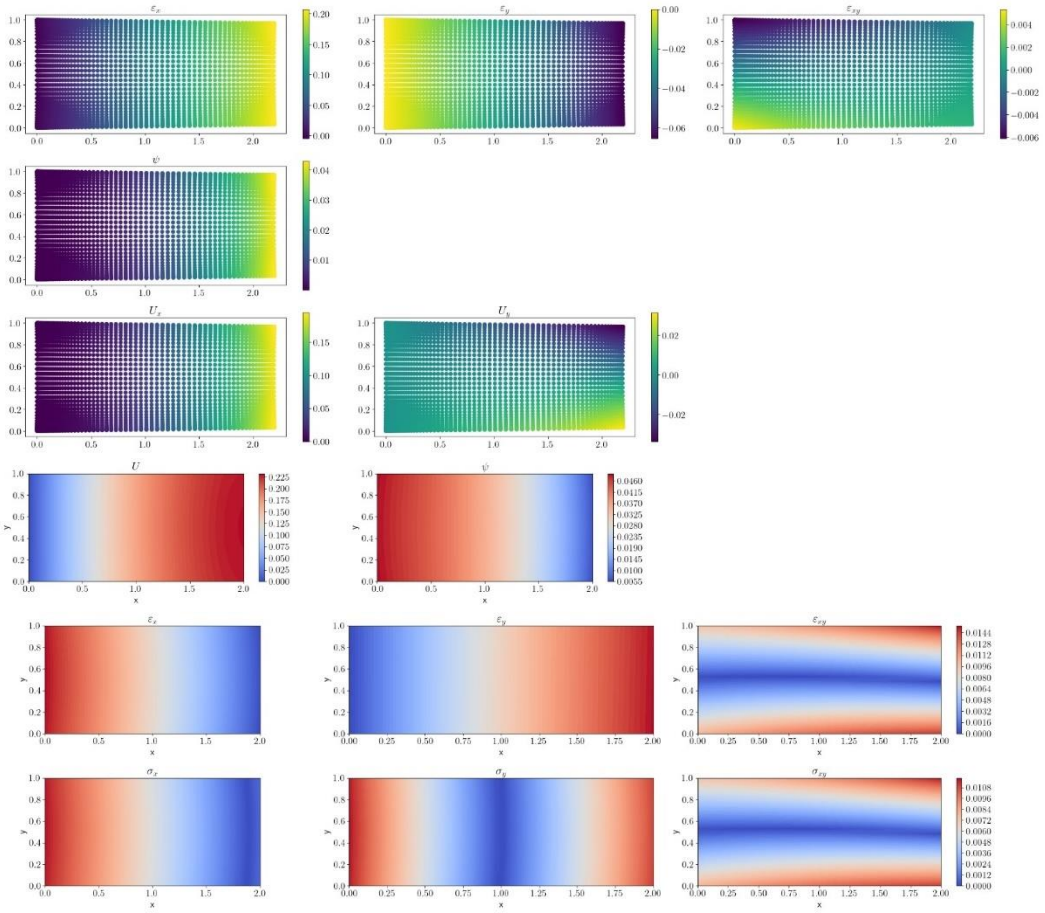


Figure 22: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution for the observed 2D domain problem from 3)

In conclusion, the hyperparameter selection 3) will be used in the next chapter for future implementation, with the results based on this model, since it offers qualitatively and quantitatively better predictions of the material.

4 Material modelling

In this chapter, the material of the analysed body is changed from linear elasticity (Hooke's law) to a compressible Neo-Hookean material in 2D similar to the second homework. Furthermore, the CANN is applied to replace the analytical Neo-Hookean material model to predict the stress components and the strain energy density.

The strain energy density ψ of a compressible Neo-Hookean material in 2D is analytically determined according to Equation (16). In the next step, the stress tensor σ, in this case the Cauchy stress tensor, is calculated analytically as shown in Equation (17). In the case of computing the Cauchy stress tensor σ from the predicted strain energy density ψ, it can be derived directly from the gradient of the strain energy density with respect to the strain ε, as stated in Equation (18).

$$\psi = \frac{\lambda}{2} \cdot (\ln J)^2 - \mu \cdot (\ln J) + \frac{\mu}{2} \cdot (I_1 - 2) \tag{16}$$

$$\sigma = \mu(FF^T - I) + \lambda \cdot \ln(JI) \text{ with I the identity tensor} \tag{17}$$[3]

$$\sigma = \frac{\delta \psi}{\delta \varepsilon} \tag{18}$$[4]

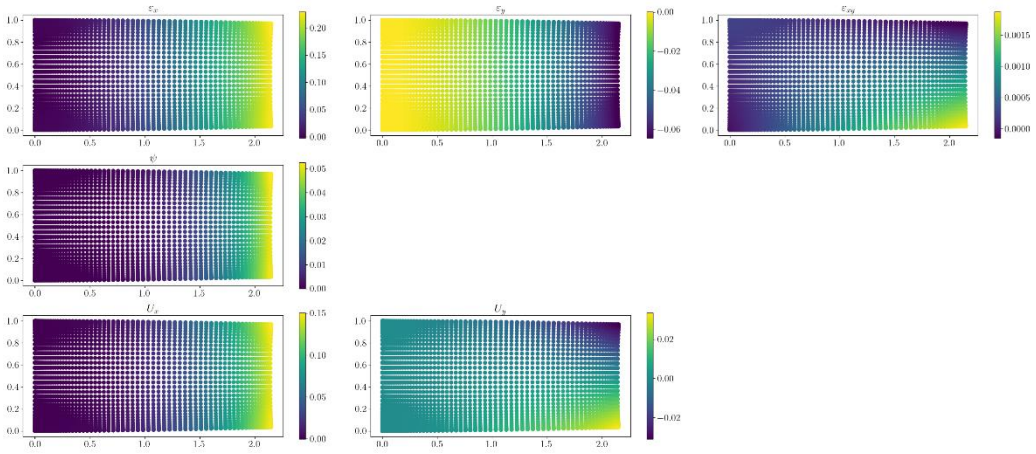
Unfortunately, the implementation of the Neo-Hookean material will not be pursued further due to multiple errors, likely stemming from the computation of the Helmholtz free energy density. However, for transparency purposes, the code will still be uploaded as well. It is also likely that this will be reimplemented before the presentation takes place.

5 Kolmogorov-Arnold Networks (KAN)

The main focus of this chapter is to exchange the implemented Conventional Fully Connected Neural Network (FCNN) in the Deep Energy Method (DEM) by a Kolmogorov-Arnold-Network (KAN). It is also considered to conduct a hyperparameter optimization and compare the computational effort and accuracy with the conventional implementation of the DEM.

The implementation of KAN in our code didn't use any library, since we cannot install *pykan* [6] via github and PyPi. Overall, the implementation of KAN offers a much faster computation than other models. The results are shown below, although the hyperparameter optimization is not implemented yet.

a)



[3] Source: ChatGPT
[4] Source: ChatGPT

b)

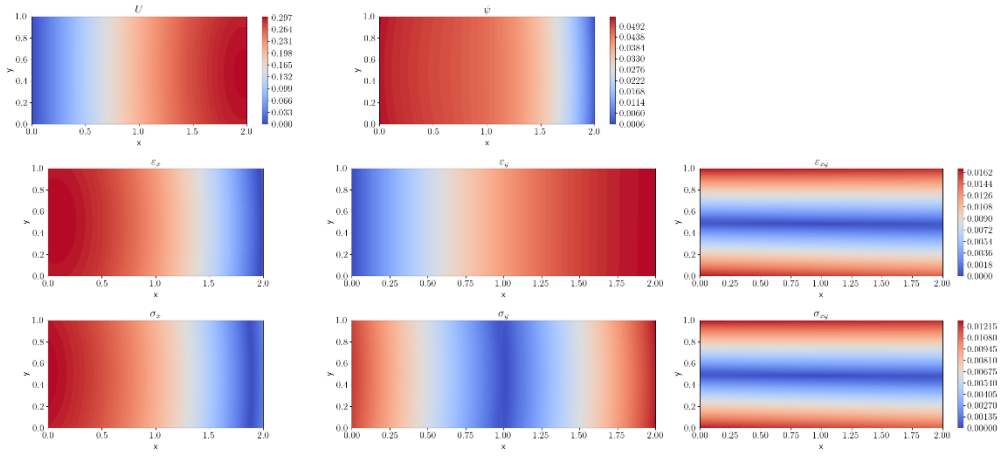


Figure 23: a) Predicted solutions; b) absolute errors w.r.t. the analytical solution for the observed 2D domain problem from KAN

6 Performance optimization

In this task, the final version of the source code should be optimized for maximal execution speed. Furthermore, the performance of the code, e.g. runtime per epoch, will be measured and the effect of each optimization step is documented.

Likely presented in the presentation.

7 List of Figures

8 List of Tables

9 References

[1] Vien Minh Nguyen-Thanh, Xiaoying Zhuang, and Timon Rabczuk. “A deep energy method for finite deformation hyperelasticity”. In: European Journal of Mechanics A/Solids 80 (2020), p. 103874. issn: 0997-7538. doi: https://doi.org/10.1016/j.euromechsol.2019.103874. url: https://www.sciencedirect.com/science/article/pii/S0997753819305352

[2] https://e6.ijs.si/medusa/wiki/index.php/Solid_Mechanics

[3] https://jth-computation.hj.se/GeneralizedHookesLaw/

[4] https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.simpson.html

[5] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljaciˇ c, T. Y. Hou, and M. Tegmark. ´ Kan: Kolmogorov-arnold networks. https://arxiv.org/html/2404.19756v1