NN-based reduced order modeling of PDEs

Andrea Boselli
Carlo Ghiglione
Leonardo Perelli
Stefano Pagani
Francesco Regazzoni

Objective

Given a PDE depending on parameters:

➡ Case study 1: Comet Equation

$$\begin{cases} -\mu\Delta u + 10(\cos\theta, \sin\theta)\cdot\nabla u = 10e^{-100|\underline{x}-\underline{x}_0|} & \underline{x} \in \Omega = [0,1]^2 \\ u = 0 & \underline{x} \in \partial\Omega \end{cases}$$

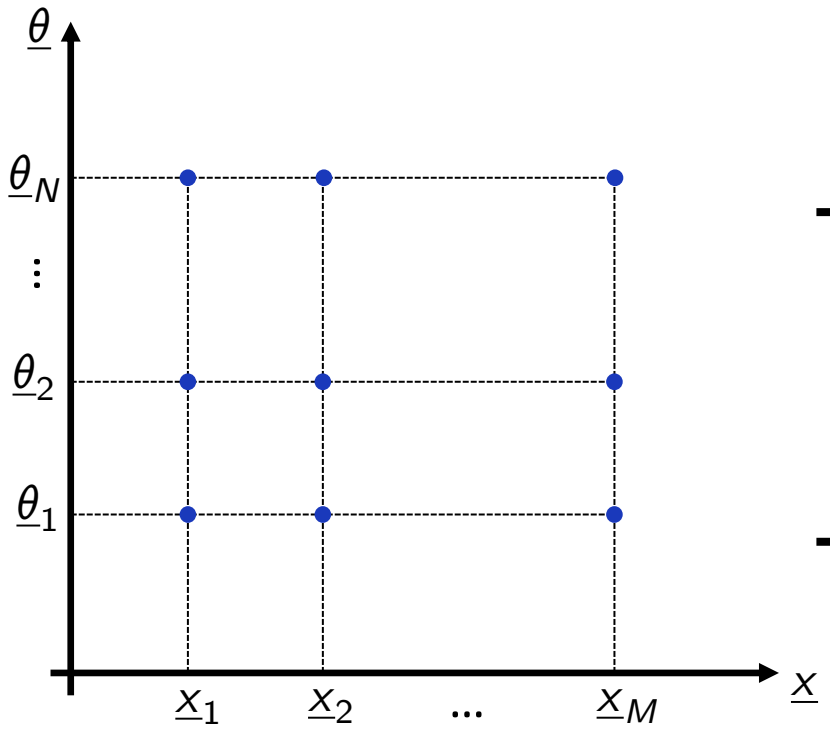
- Test the ability of Neural Networks to approximate the manifold of the solutions of the PDE for all the parameters
- Compute fast the solutions of the PDE for many values of the parameters
- 2 main ingredients
  - Data
  - Optimization

Ingredient 1: Data

Neural Networks need to learn from given high fidelity solution

➡

Compute numerical solutions of the PDE for many values of the parameters on a grid



Numerical method

Training set $\mathbb{X}$

$(\underline{x}_i, \underline{\theta}_1, u_{HF}(\underline{x}_i, \underline{\theta}_1)) \quad i = 1, ..., M$

...

$(\underline{x}_i, \underline{\theta}_N, u_{HF}(\underline{x}_i, \underline{\theta}_N)) \quad i = 1, ..., M$

Ingredient 2: Optimization

Loss functions measure the quality of the result:

Loss $L(w)$ ⬇ $\iff$ Quality of the approximation ⬆

$$L(\underline{w}) = \alpha_1 L_{Fit}(\underline{w}) + \alpha_2 L_{PDE}(\underline{w}) + \alpha_3 L_{BC}(\underline{w})$$

- $L_{Fit}(\underline{w})$: Approximation error
- $L_{PDE}(\underline{w})$: PDE residual of Neural Network solution
- $L_{BC}(\underline{w})$: BC residual of Neural Network solution

} **PINN** approach

Find the solution minimizing the loss through gradient descent:

$$\underline{w}_{n+1} = \underline{w}_n - \eta \nabla L(\underline{w}_n)$$

Direct Solver

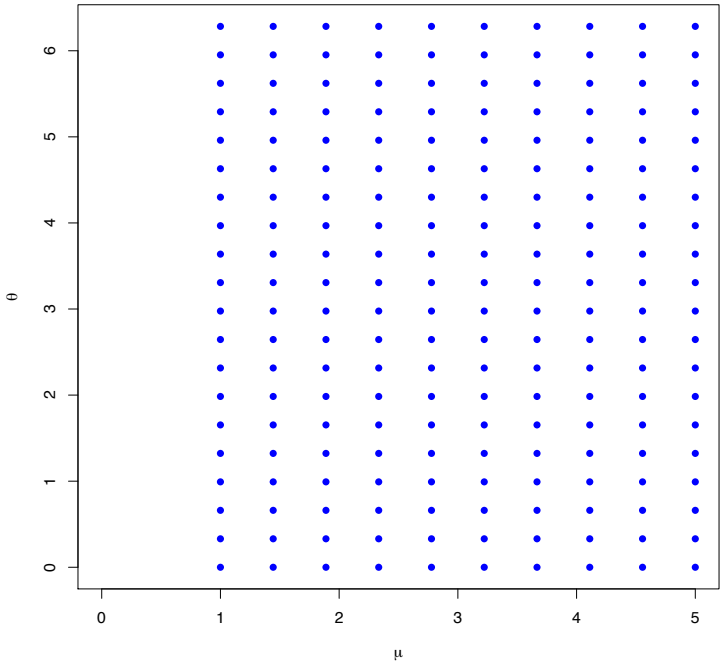
- Represent the manifold of the PDE solutions through a Neural Network:

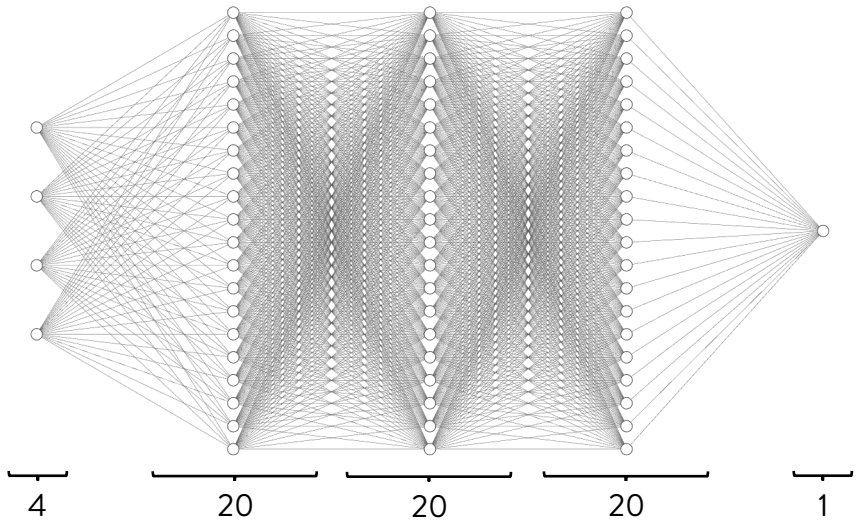
$$u_{NN} : \quad [0,1]^2 \times \mathbb{R}^+ \times \mathbb{R} \to \mathbb{R}$$
$$(x, y, \mu, \theta) \mapsto u_{NN}(x, y, \mu, \theta)$$

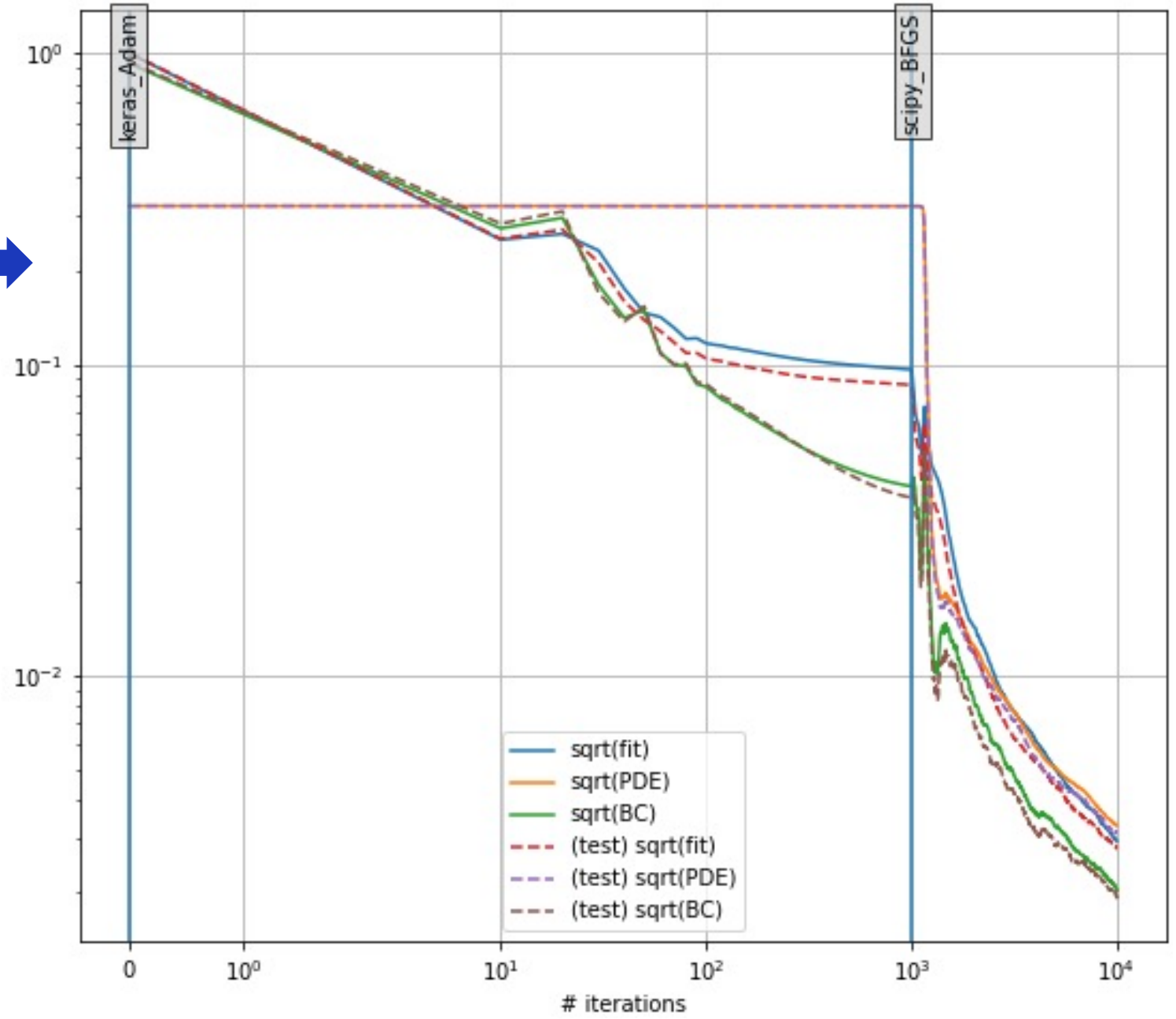
- Test different datasets: different ranges of values for μ
- Test different architectures
  - Number of inner layers
  - Number of neurons per layer
- Test different weights for the loss :

$$L(\underline{w}) = \alpha_1 L_{Fit}(\underline{w}) + \alpha_2 L_{PDE}(\underline{w}) + \alpha_3 L_{BC}(\underline{w})$$

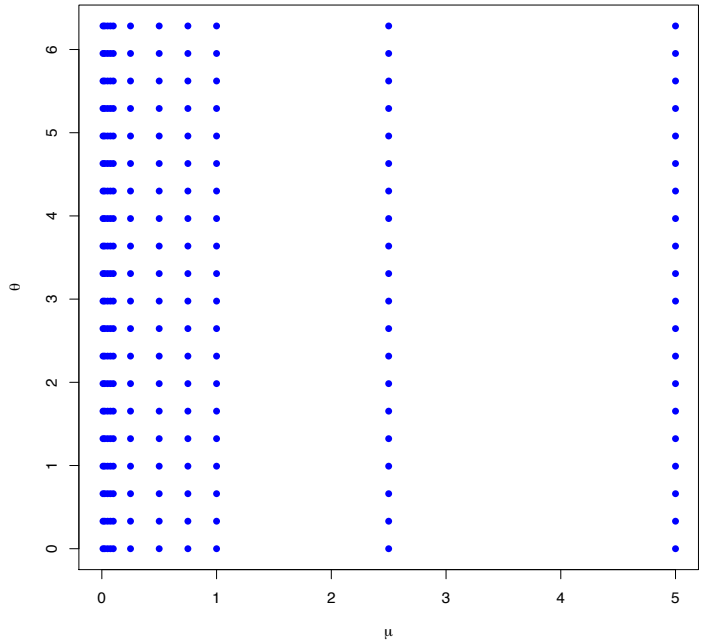
A

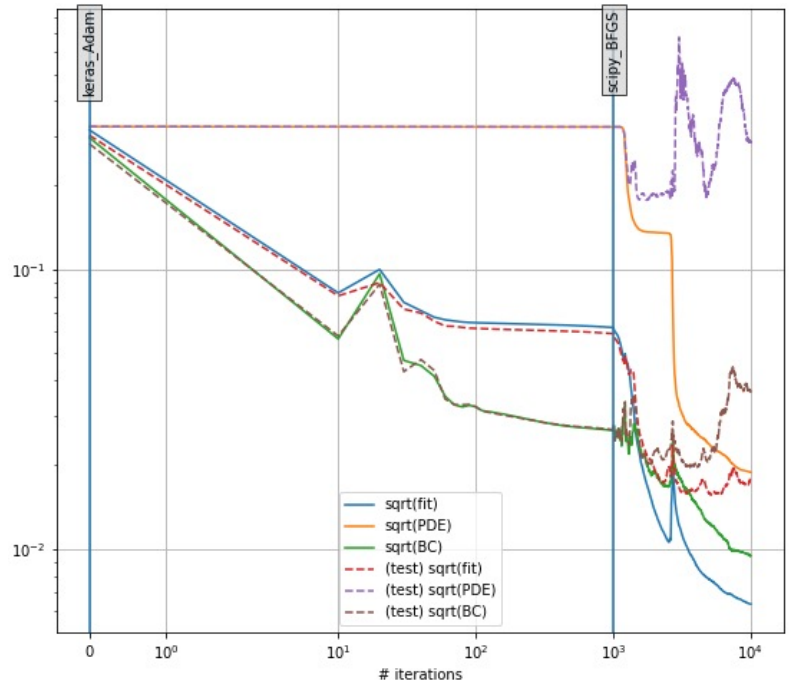

θ
μ
0
1
2
3
4
5
6


4
20
20
20
1


keras_Adam
scipy_BFGS
10^0
10^{-1}
10^{-2}
sqrt(fit)
sqrt(PDE)
sqrt(BC)
(test) sqrt(fit)
(test) sqrt(PDE)
(test) sqrt(BC)
0
10^0
10^1
10^2
10^3
10^4
iterations

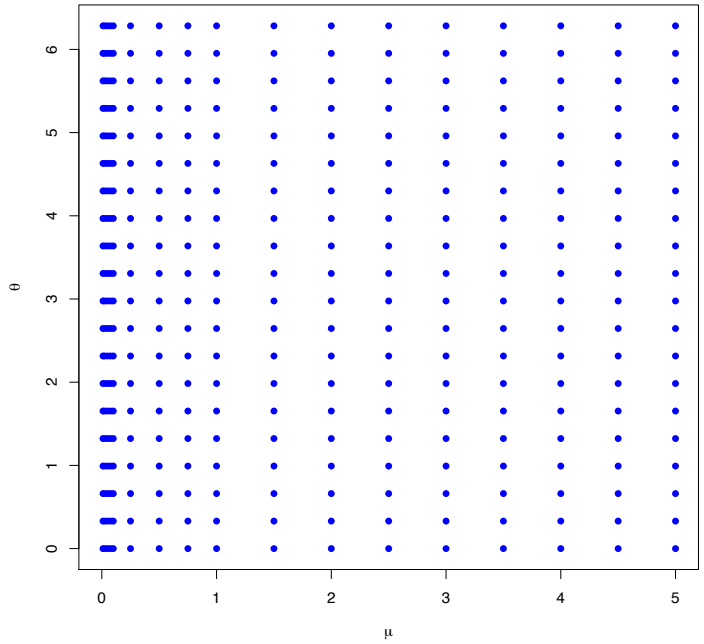
B

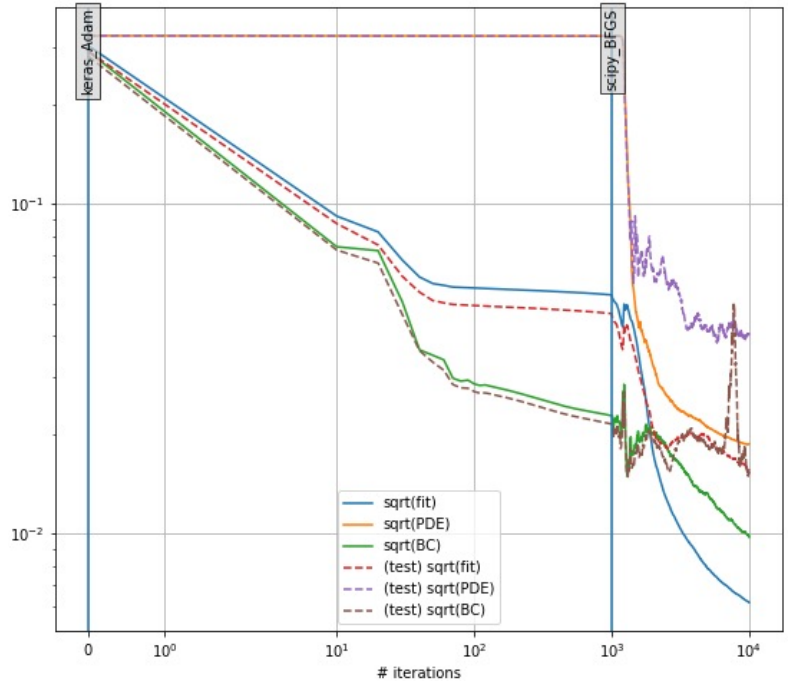

θ
μ
0
1
2
3
4
5
6


keras_Adam
scipy_BFGS
sqrt(fit)
sqrt(PDE)
sqrt(BC)
(test) sqrt(fit)
(test) sqrt(PDE)
(test) sqrt(BC)
iterations

X

C


θ
μ
0
1
2
3
4
5
6


keras_Adam
scipy_BFGS
sqrt(fit)
sqrt(PDE)
sqrt(BC)
(test) sqrt(fit)
(test) sqrt(PDE)
(test) sqrt(BC)
iterations

V

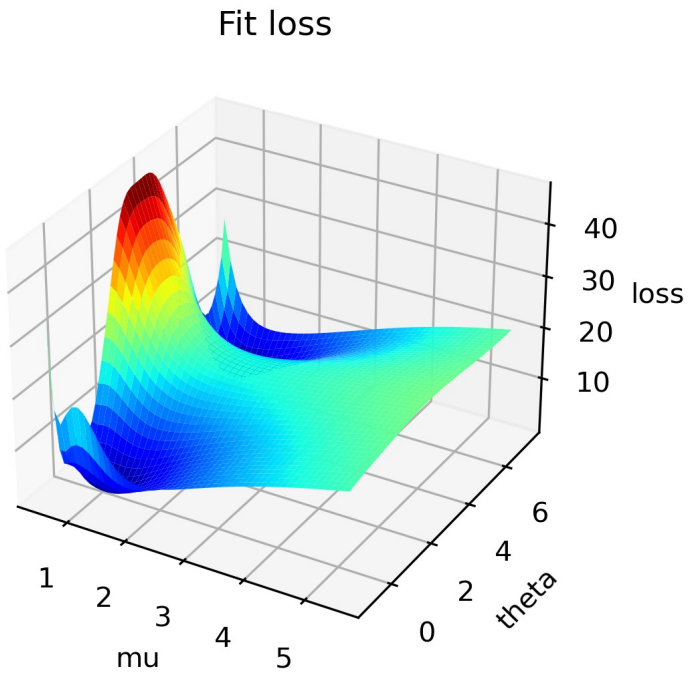
Identification

- Given u solution for some $(\tilde{\mu}, \tilde{\theta})$ ➡ Find such $(\tilde{\mu}, \tilde{\theta})$
- Exploit the estimate of the manifold of the solutions u_{NN} ➡ Identification tests the reliability of the trained u_{NN}

$$\mathcal{L}(\mu, \theta) = \mathcal{L}_{fit}(\mu, \theta) = MSE_{int}[u - u_{NN}(\cdot, \cdot, \mu, \theta)]$$

➡ $(\tilde{\mu}, \tilde{\theta}) = argmin\mathcal{L}(\mu, \theta)$ Through gradient descent

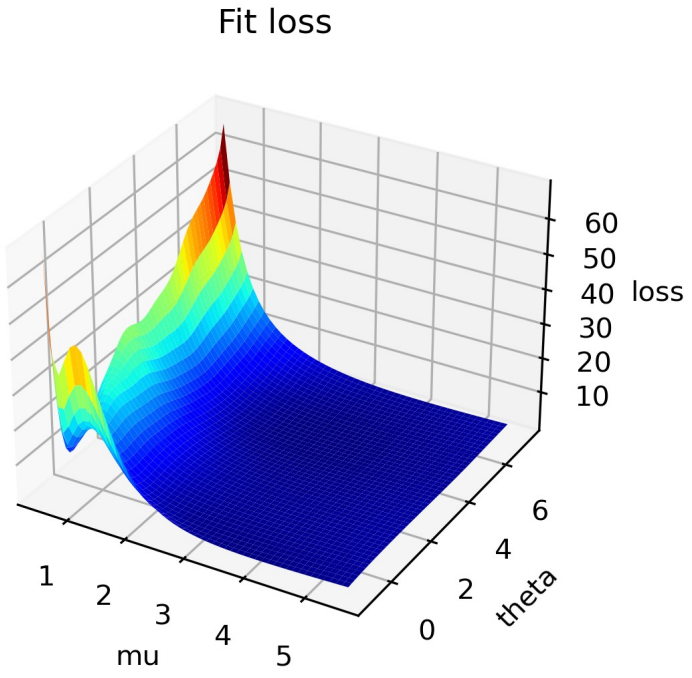
Experiment 1



	μ	θ
Actual Value	1.00	0.00
Initial Value	5.00	3.141592
Final Value	0.999286	6.297931

	μ	θ
Actual Value	3.00	4.45
Initial Value	1.00	3.00
Final Value	3.137339	4.481961

Experiment 2



	μ	θ
Actual Value	3.00	4.45
Initial Value	3.00	1.00
Final Value	4.092807	-1.043056

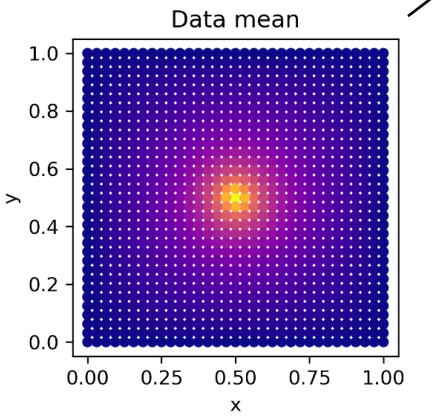
Proper Orthogonal Decomposition

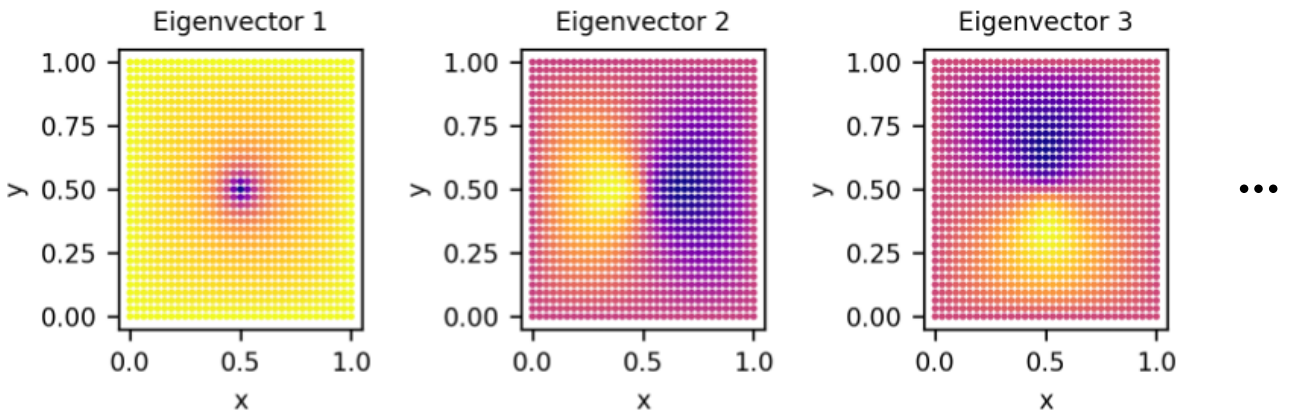
u as linear combination of basis functions:

$$u(x,y) \approx \sum_{k=1}^{K} u_k(\mu,\theta)\phi_k(x,y) + u_0(x,y) + b(\mu,\theta)$$

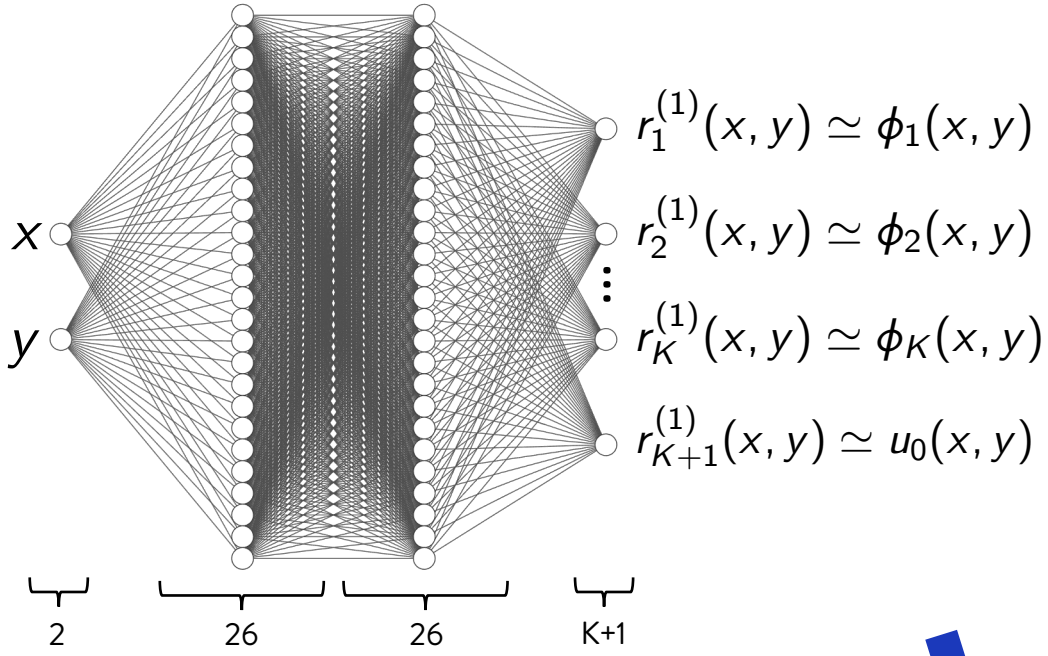
coefficients — basis functions — mean function — bias

Phase 1: extract through POD u_0 and $\{\phi_k\}_{k=1}^{K}$



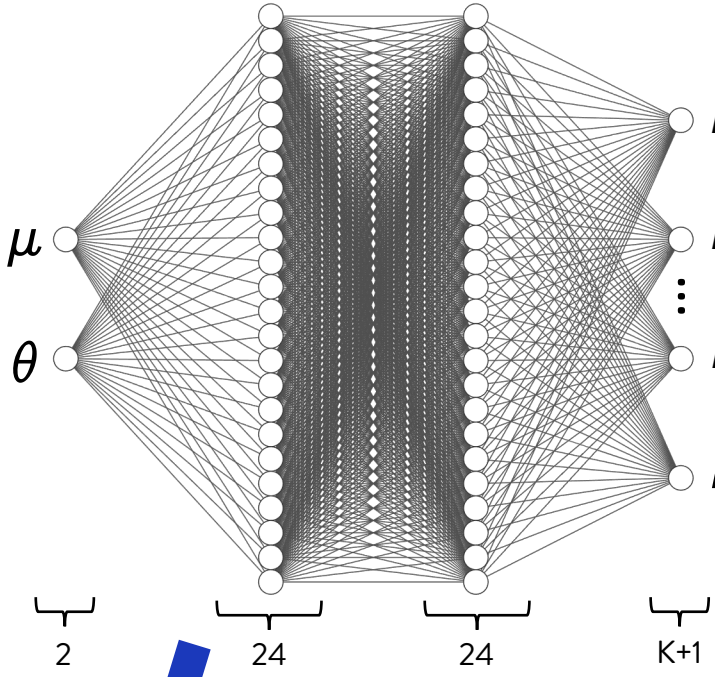


Phase 2: train $\underline{r}^{(1)}$ to learn basis functions



$$r_1^{(1)}(x,y) \simeq \phi_1(x,y)$$
$$r_2^{(1)}(x,y) \simeq \phi_2(x,y)$$
$$r_K^{(1)}(x,y) \simeq \phi_K(x,y)$$
$$r_{K+1}^{(1)}(x,y) \simeq u_0(x,y)$$

Phase 3: train $\underline{r}^{(2)}$ to learn coefficients

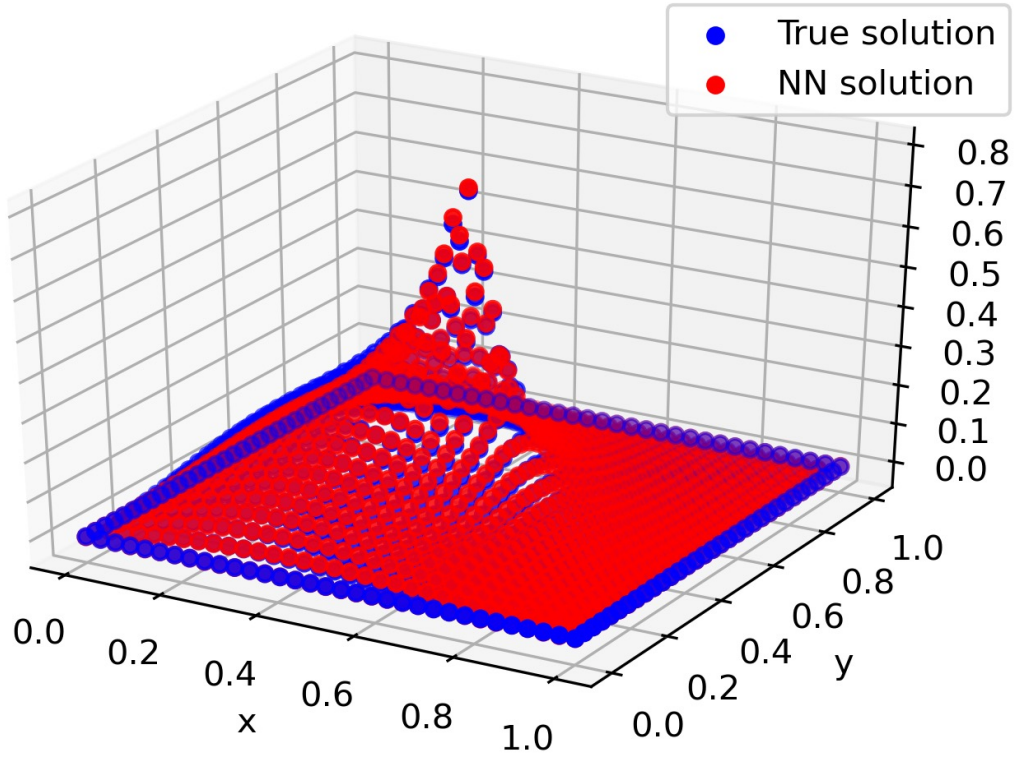


$$r_1^{(2)}(\mu,\theta) \simeq u_1(\mu,\theta)$$
$$r_2^{(2)}(\mu,\theta) \simeq u_2(\mu,\theta)$$
$$r_K^{(2)}(\mu,\theta) \simeq u_K(\mu,\theta)$$
$$r_{K+1}^{(2)}(\mu,\theta) \simeq b(\mu,\theta)$$

$$u_{POD}(x,y,\mu,\theta) = \sum_{k=1}^{K} r_k^{(2)}(\mu,\theta) r_k^{(1)}(x,y) + r_{K+1}^{(1)}(x,y) + r_{K+1}^{(2)}(\mu,\theta) \simeq u(x,y)$$

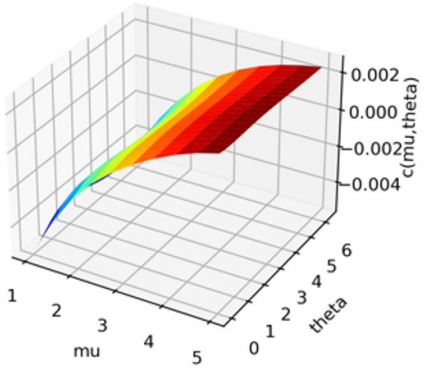
Results

❖ Reconstruction is correct

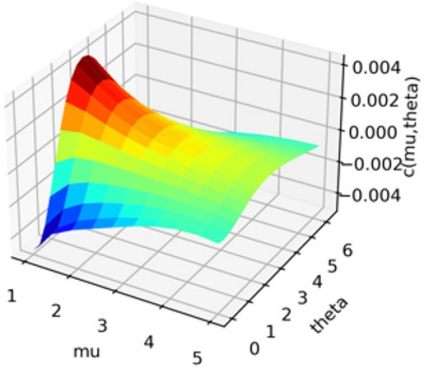


❖ Coefficients are interpretable

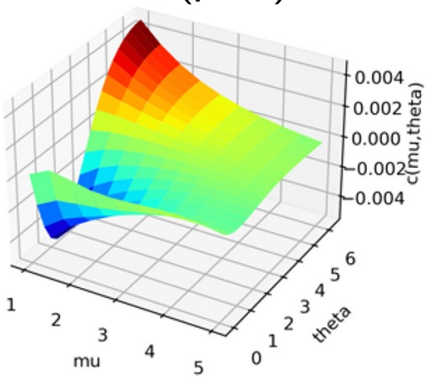
$u_1(\mu, \theta)$



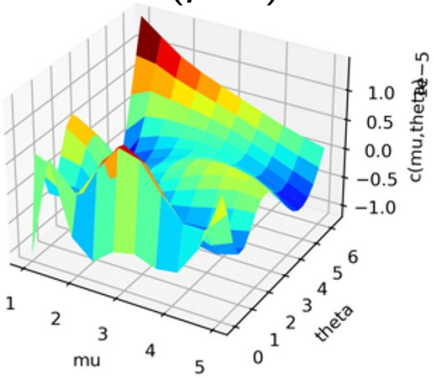
$u_2(\mu, \theta)$



$u_3(\mu, \theta)$



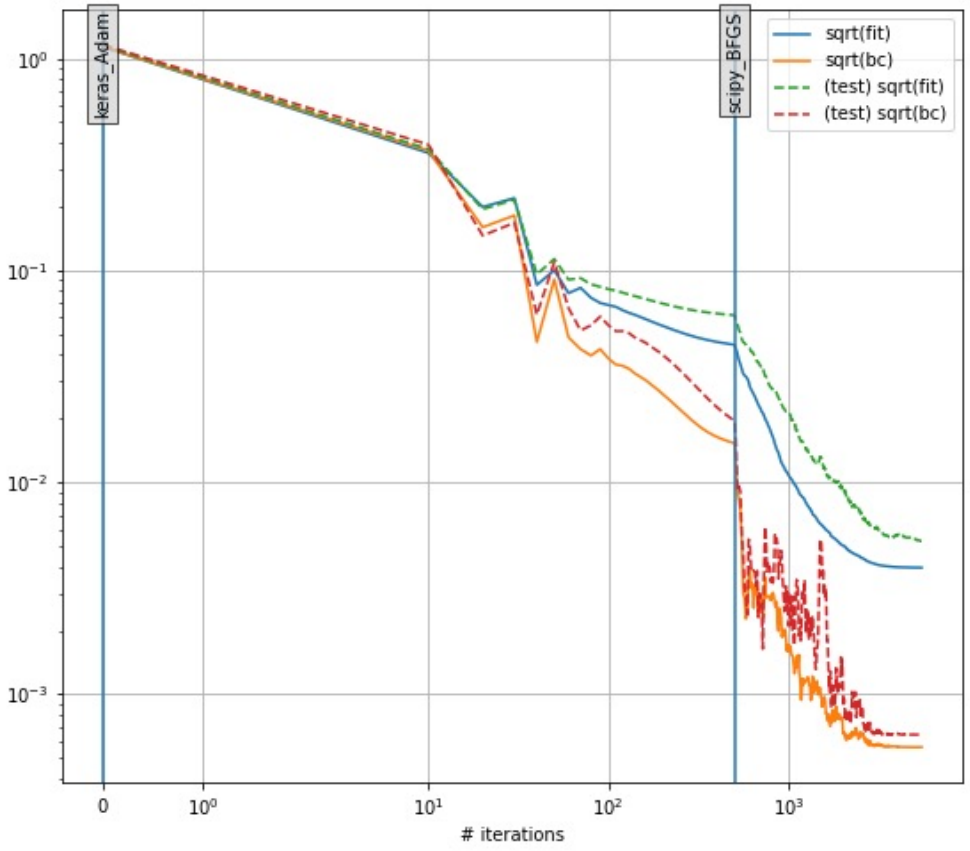
$b(\mu, \theta)$



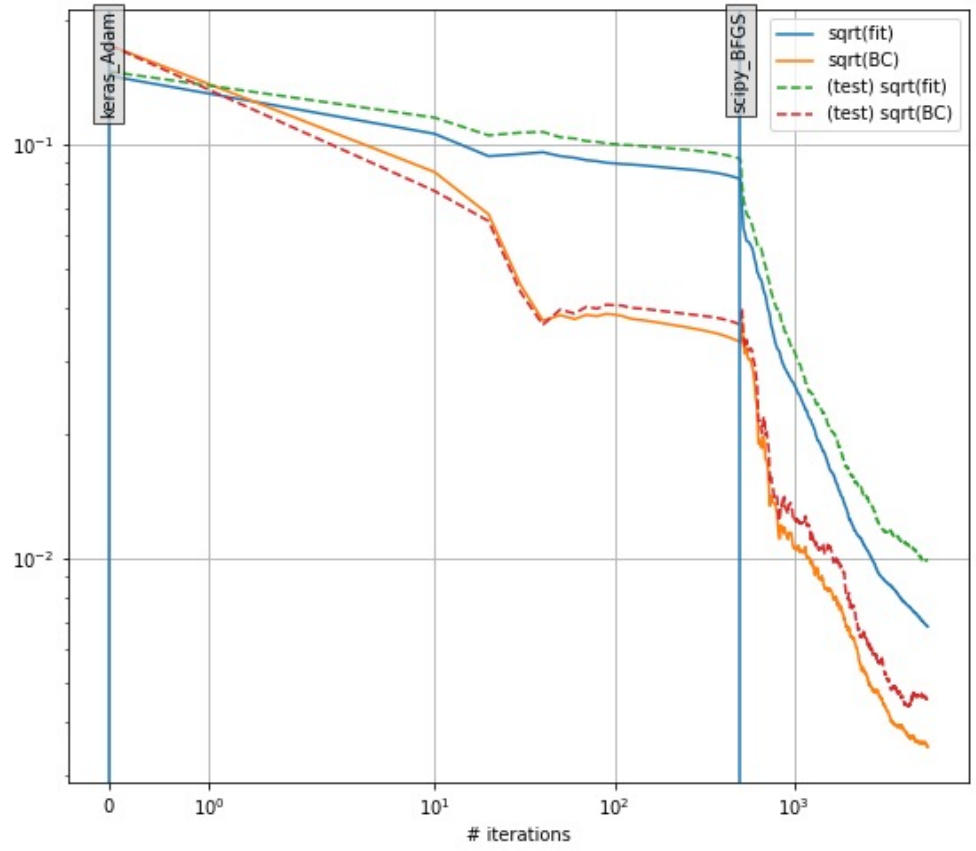
Comparison POD - Solver

POD reaches a significantly lower global loss in almost ¾ of the time

POD (K = 5)

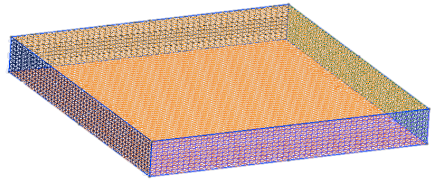


Solver

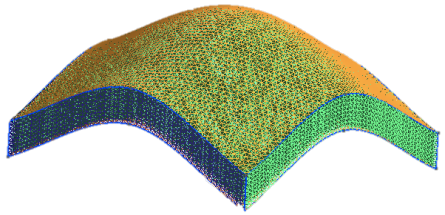


Variable Domain Framework

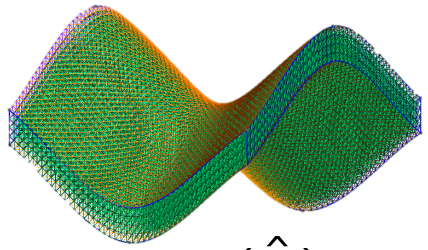
Consider a PDE defined on a variable domain:



$\hat{\Omega}$



$\pi_{\underline{p}_1}(\hat{\Omega})$



$\pi_{\underline{p}_2}(\hat{\Omega})$

1. Provide a parametrization π of the PDE domain

$$\pi : (\underline{\hat{x}}, \underline{p}) \mapsto \underline{x} = \pi_{\underline{p}}(\underline{\hat{x}})$$

2. Build a solver that outputs u_{HF} solutions from different geometries

$$u_{HF}(\underline{x}, \underline{p}) = u_{HF}(\pi_{\underline{p}}(\underline{\hat{x}}), \underline{p})$$

3. Represent the manifold of the PDE solutions u_{NN} for various geometries

$$u_{NN}(\underline{\hat{x}}, \underline{p}) \simeq u_{HF}(\pi_{\underline{p}}(\underline{\hat{x}}), \underline{p})$$

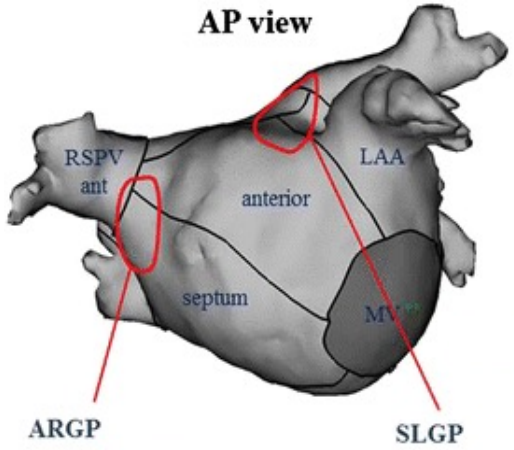
Our Case Study

Monodomain Equation:

$$\begin{cases} \chi_m C_m \frac{\partial u}{\partial t} - \nabla \cdot (\Sigma \nabla u) + \chi_m I_{ion} = I^{ext} & \underline{x} \in \Omega, \quad t \in (0, T] \\ \frac{\partial u}{\partial \nu} = 0 & \underline{x} \in \partial\Omega, t \in (0, T] \\ u(\underline{x}, 0) = u_0(\underline{x}) & \underline{x} \in \Omega \end{cases}$$

$u(\underline{x}, t)$: transmembrane potential

➡ Model deformations of the membranes of the human heart:

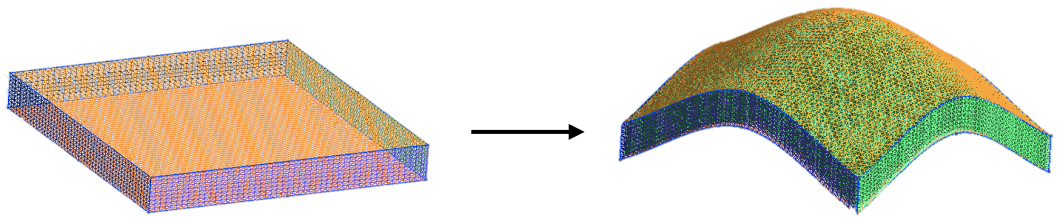


❖ Instead of learning $u(\underline{x}, t)$, learn the Activation Time: $AT(\underline{x}) := \underset{t}{argmax} \frac{\partial u}{\partial t}$ → PDE loss X / Fit loss V

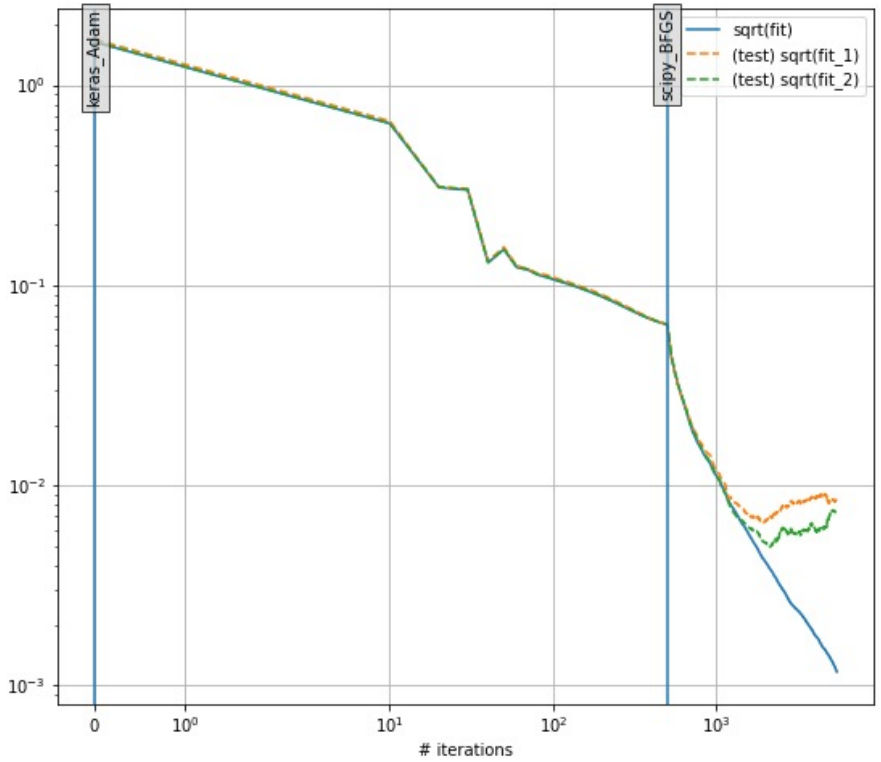
❖ Learn directly through a Neural Network ➡ **Direct Solver**

Symmetric Meshes

All middle points are raised to the same height

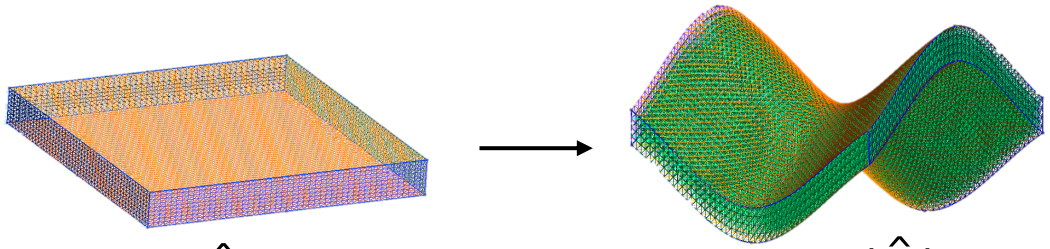


$\hat{\Omega}$ $\pi_{\underline{p}}(\hat{\Omega})$

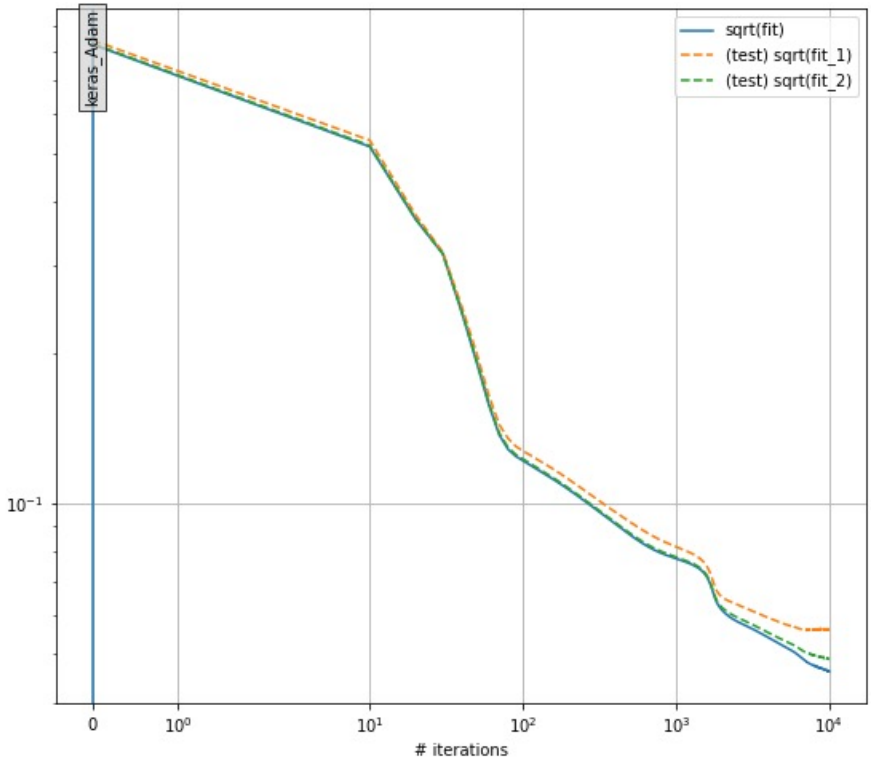


Asymmetric Meshes

Middle points of facing sides are raised to the same height



$\hat{\Omega}$ $\pi_{\underline{p}}(\hat{\Omega})$



Geometrical + Physical Parameters

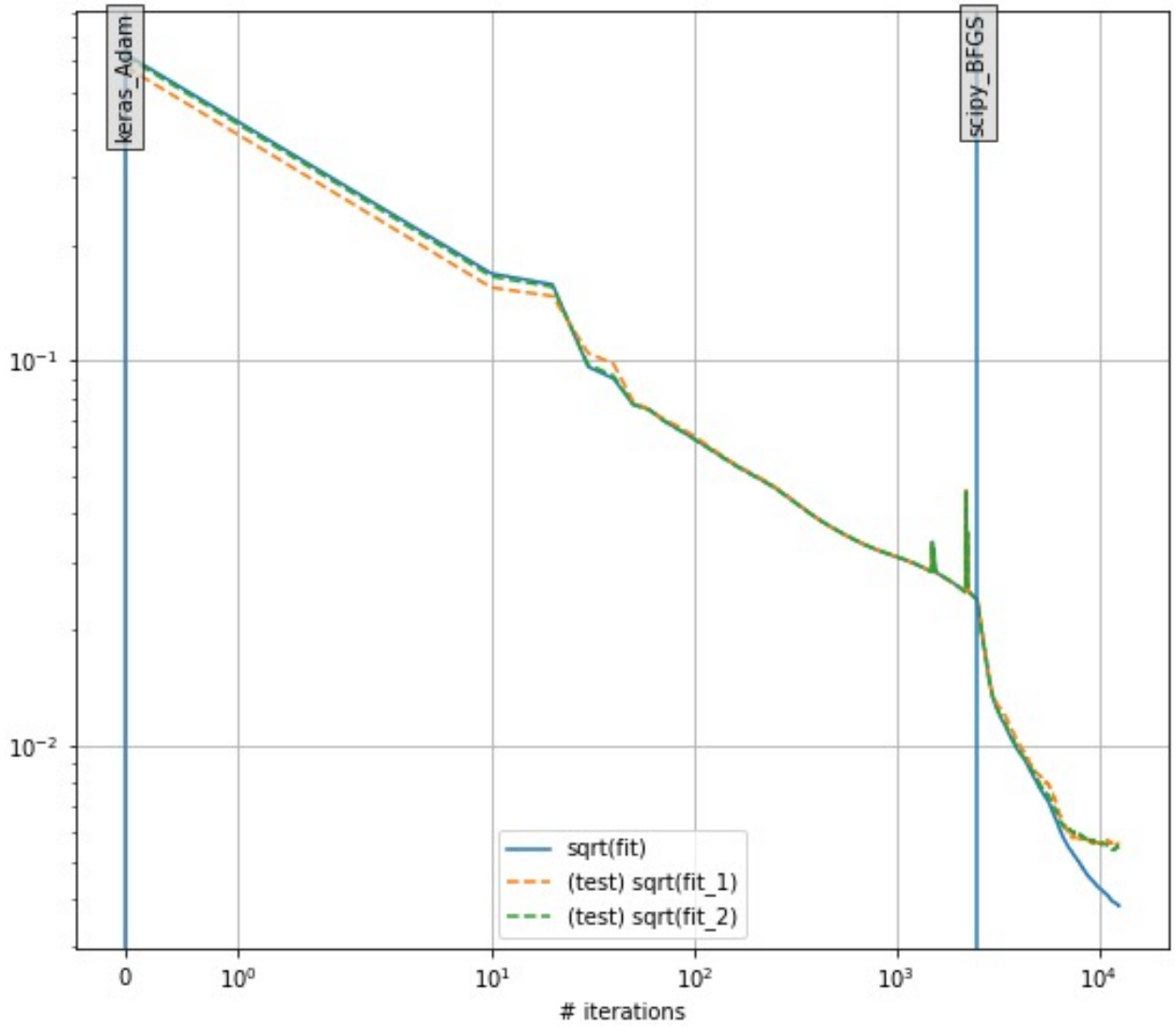
Geometrical parameters:

- Dealing with mesh deformation
- Symmetric meshes

Physical parameters:

- Conductivity of the material
- Directly influences AT:

  Conductivity ⬆ ⟺ AT ⬇



Conclusions

- Implemented and compared different architectures → Direct solver, POD
- Checked the reliability of the obtained solutions → Identification
- Applied to the framework of PDE domain dependent on parameters → Monodomain equation