Workshop

eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

Taller

Coding Deep Neural Networks for PDEs
Session III: Introducing Least Squares in (V)PINNs

Professors: Carlos Uriarte*, Ángel Javier Omella*, David Pardo*

Organizers: Ignacio Muga†, Paulina Sepúlveda†

*Research Group on Applied Mathematical Modelling, Statistics, and Optimization (MATHMODE) at UPV/EHU

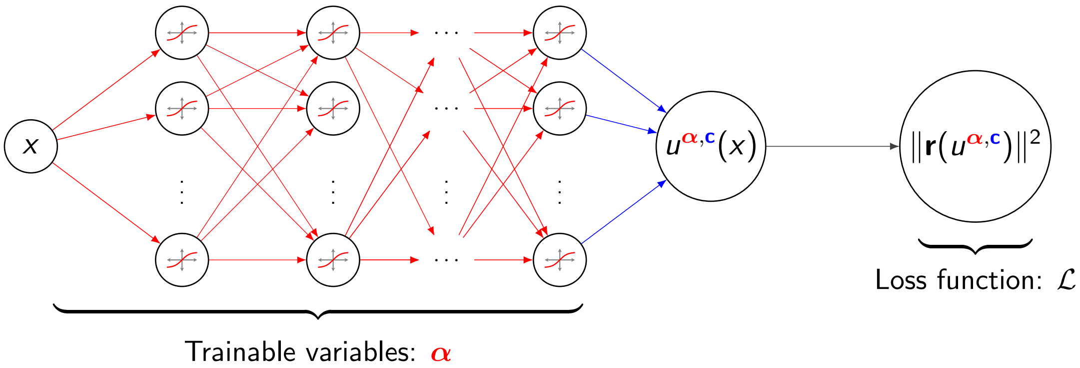
†IMA Numerics Research Group at PUCV

Follow us on: @mathmode.science, @ima.numerics.pucv†*

Biosfera Lodge, Olmué. January 19-24, 2025

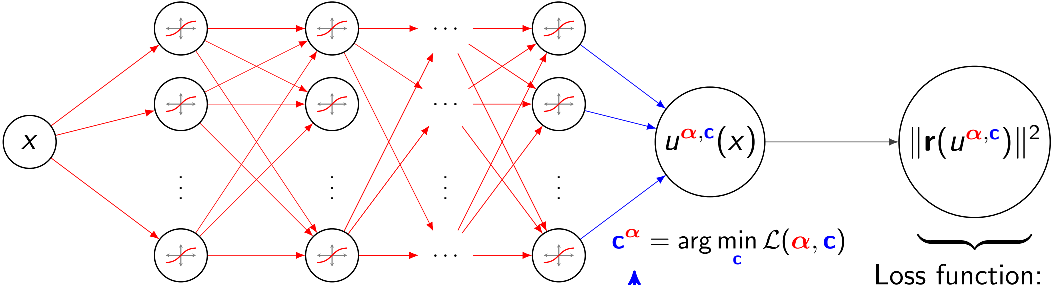
The LS/GD Optimizer

$$\underbrace{u^{\boldsymbol{\alpha},\mathbf{c}}(x)}_{\substack{\text{scalar-valued}\\ \text{neural network}}} = \mathbf{c}\cdot\mathbf{u}^{\boldsymbol{\alpha}}(x) = \sum_{n=1}^{N} c_n \cdot u_n^{\boldsymbol{\alpha}}(x), \qquad \underbrace{\mathbf{u}^{\boldsymbol{\alpha}}(x) := \{u_n^{\boldsymbol{\alpha}}(x)\}_{n=1}^{N}}_{\substack{\text{vector-valued}\\ \text{neural network}}} \text{ spanning set to discretize } \mathbb{U}$$



The LS/GD Optimizer

$$\underbrace{u^{\boldsymbol{\alpha},\mathbf{c}}(x)}_{\substack{\text{scalar-valued}\\ \text{neural network}}} = \mathbf{c}\cdot\mathbf{u}^{\boldsymbol{\alpha}}(x) = \sum_{n=1}^{N} c_n \cdot u_n^{\boldsymbol{\alpha}}(x), \qquad \underbrace{\mathbf{u}^{\boldsymbol{\alpha}}(x) := \{u_n^{\boldsymbol{\alpha}}(x)\}_{n=1}^{N}}_{\substack{\text{vector-valued}\\ \text{neural network}}} \text{ spanning set to discretize } \mathbb{U}$$



$$\mathbf{c}^{\boldsymbol{\alpha}} = \arg\min_{\mathbf{c}} \mathcal{L}(\boldsymbol{\alpha}, \mathbf{c})$$

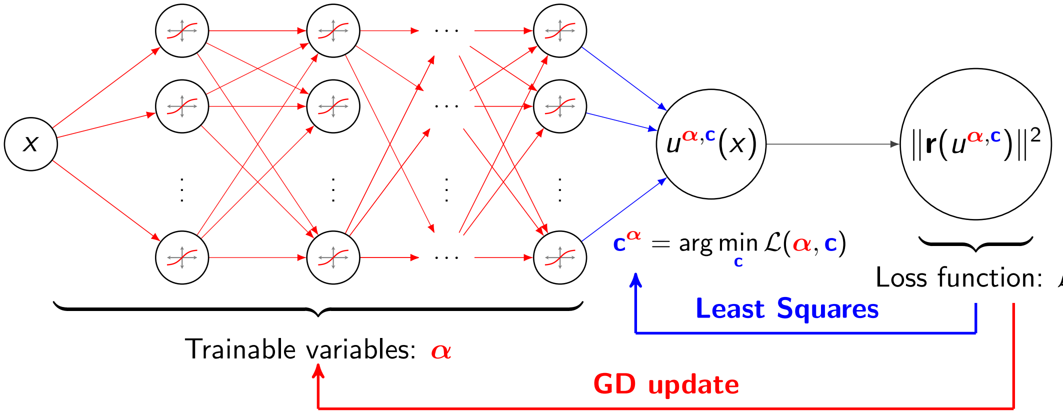
Loss function: $\mathcal{L}$

Least Squares

Trainable variables: $\boldsymbol{\alpha}$

The LS/GD Optimizer

$$\underbrace{u^{\boldsymbol{\alpha},\mathbf{c}}(x)}_{\substack{\text{scalar-valued}\\ \text{neural network}}} = \mathbf{c}\cdot\mathbf{u}^{\boldsymbol{\alpha}}(x) = \sum_{n=1}^{N} c_n \cdot u_n^{\boldsymbol{\alpha}}(x), \qquad \underbrace{\mathbf{u}^{\boldsymbol{\alpha}}(x) := \{u_n^{\boldsymbol{\alpha}}(x)\}_{n=1}^{N}}_{\substack{\text{vector-valued}\\ \text{neural network}}} \text{ spanning set to discretize } \mathbb{U}$$



Petrov-Galerkin-Based Gradient-Descent Optimization

$$\mathcal{L}(\boldsymbol{\alpha}, \mathbf{c}) = \|\mathbf{r}(u^{\boldsymbol{\alpha},\mathbf{c}})\|^2$$

$$\boxed{\min_{\boldsymbol{\alpha},\mathbf{c}} \mathcal{L}(\boldsymbol{\alpha}, \mathbf{c})}$$

Conventional GD optimization

For $\boldsymbol{\theta} := (\boldsymbol{\alpha}, \mathbf{c})$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \boldsymbol{\eta}_t \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_{t-1})$$

$\boldsymbol{\eta}$: learning rate

Petrov-Galerkin-Based Gradient-Descent Optimization

$$\mathcal{L}(\boldsymbol{\alpha}, \mathbf{c}) = \left\| \mathbf{r}(u^{\boldsymbol{\alpha},\mathbf{c}}) \right\|^2$$

$$\boxed{\min_{\boldsymbol{\alpha},\mathbf{c}} \mathcal{L}(\boldsymbol{\alpha}, \mathbf{c})}$$

$$\boxed{\min_{\boldsymbol{\alpha}} \min_{\mathbf{c}} \mathcal{L}(\boldsymbol{\alpha}, \mathbf{c})}$$

Conventional GD optimization

For $\boldsymbol{\theta} := (\boldsymbol{\alpha}, \mathbf{c})$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \boldsymbol{\eta}_t \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_{t-1})$$

$\boldsymbol{\eta}$: learning rate

Hybrid LS/GD optimization

1. **Petrov-Galerkin.** Given $\boldsymbol{\alpha}$, find $\mathbf{c}^{\boldsymbol{\alpha}}$.

2. Update $\boldsymbol{\alpha}$ using GD.

Implementation

Given $\boldsymbol{\alpha}$, we have:

$$\mathbf{r}(u^{\boldsymbol{\alpha},\mathbf{c}}) = \mathbf{B}^{\boldsymbol{\alpha}}\mathbf{c} - \mathbf{l}$$
$$\mathbf{c}^{\boldsymbol{\alpha}} = \arg\min_{\mathbf{c}\in\mathbb{R}^N} \|\mathbf{r}(u^{\boldsymbol{\alpha},\mathbf{c}})\|^2 = \arg\min_{\mathbf{c}\in\mathbb{R}^N} \|\mathbf{B}^{\boldsymbol{\alpha}}\mathbf{c} - \mathbf{l}\|^2$$

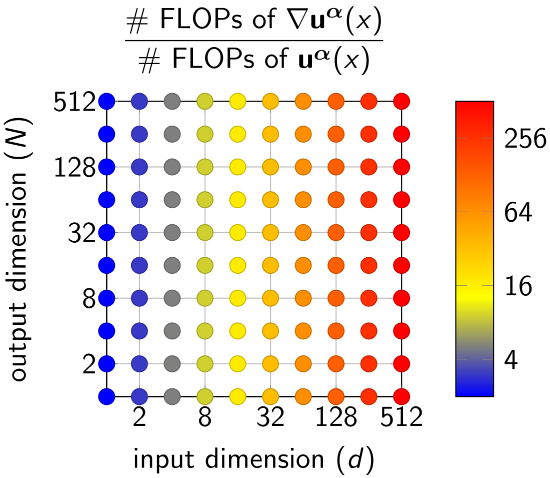
Matrix and load vector (Poisson's equation: $-\Delta u = f$)

$$\mathbf{B}^{\boldsymbol{\alpha}}_{m,n} = \int_\alpha \nabla u_n^{\boldsymbol{\alpha}} \cdot \nabla v_m \approx \underbrace{\sum_{k=1}^{K} \gamma_k \, \nabla u_n^{\boldsymbol{\alpha}}(x_k) \cdot \nabla v_m(x_k)}_{\text{Quadrature}}, \qquad \mathbf{l}_m = \int_\alpha f v_m \approx \underbrace{\sum_{k=1}^{K} \gamma_k \, f(x_k)\, v(x_k)}_{\text{Quadrature}}$$

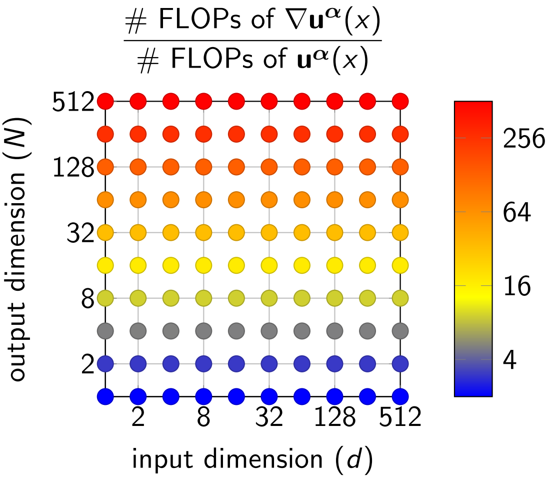
γ_k: quadrature weight, x_k: quadrature point, $1 \le m \le M$, $1 \le n \le N$.

- We compute $\mathbf{c}^{\boldsymbol{\alpha}}$ via a direct LS solver, e.g. `tf.linalg.lstsqr` $(\mathbf{B}^{\boldsymbol{\alpha}}, \mathbf{l})$. $\boxed{\textbf{Cost: } \mathcal{O}(MN^2)}$

Automatic Differentiation



(a) Forward AD (`tf.autodiff.ForwardAccumulator`).



(b) Backward AD (`tf.GradientTape`).

Computational Cost

Using LS has the following dominant cost:

- Evaluation of $\nabla \mathbf{u}^{\boldsymbol{\alpha}}$ over $\{x_k\}_{k=1}^{K}$ for $1 \leq n \leq N$. $\boxed{\textbf{Cost: } \begin{cases} \mathcal{O}(KNC_{net}), & \text{if backward AD} \\ \mathcal{O}(KC_{net}), & \text{if forward AD} \end{cases}}$

- Summing for $1 \leq k, m, n \leq K, M, N$. $\boxed{\textbf{Cost: } \mathcal{O}(KMN)}$

- Solver of $\min\limits_{\mathbf{c} \in \mathbb{R}^N} \|\mathbf{B}^{\boldsymbol{\alpha}}\mathbf{c} - \mathbf{l}\|^2$. $\boxed{\textbf{Cost: } \mathcal{O}(MN^2)}$

Considerations

- $K \gtrsim M \gtrsim N$ (# of integration points $\gtrsim$ highest test frequency $\gtrsim$ # of spanning functions).
- $C_{net} \gtrsim MN$ (consider a network with “significant” evaluation cost).
- When applying **forward AD** (**backward AD**):

$$\boxed{\text{Cost of LS: } \mathcal{O}(KNC_{net}) + \text{Cost of GD: } \mathcal{O}(KC_{net}) = \text{Cost of LS/GD: } \mathcal{O}(KNC_{net})}$$

Computational Cost – Illustrative Experiment

- We consider a neural network with evaluation cost $C_{net} \gtrsim 512M$ and input dimension $d = 1$.
- We compare the costs of a **conventional GD optimizer** versus the **hybrid LS/GD optimizer** for $M = 5N$, $K = 10N$, and $N \in \{2, 4, 8, \ldots, 512\}$.

$$\frac{\#\ \text{FLOPs when using the } \textbf{hybrid LS/GD optimizer}}{\#\ \text{FLOPs when using a } \textbf{conventional GD optimizer}}$$

N	2	4	8	16	32	64	128	256	512
Forward AD	1.33	1.33	1.33	1.33	1.34	1.34	1.33	1.35	1.37
Backward AD	1.49	1.83	2.49	3.83	6.46	11.82	22.35	43.77	86.05

Numerical Experiments

- Given f, find $u^* \in \mathbb{U} = H_0^1(\alpha)$ s.t. $\boxed{\underbrace{\int_\alpha \nabla u^* \cdot \nabla v \; dx}_{b(u^*,v)} = \underbrace{\int_\alpha f \;\, v \; dx}_{l(v)}, \; \forall v \in \mathbb{V} = H_0^1(\alpha)}$

- We equip $\mathbb{U} = H_0^1(\alpha) = \mathbb{V}$ with $b(\cdot, \cdot)$ as the inner product.

- (Continuum-level) residual $r(u)$ coincides with error function $e(u) = u - u^*$. Then,

$$\|e(u)\|_{\mathbb{U}} = \|r(u)\|_{\mathbb{V}}, \qquad \forall u \in \mathbb{U} \qquad (\mu = \alpha = 1).$$

- We consider a Fourier basis for the test space.

- $e(u) = r(u) = \sum_{m=1}^{\infty} \{b(u, v_m) - l(v_m)\} v_m \Rightarrow \|e(u)\|_{\mathbb{U}}^2 = \|r(u)\|_{\mathbb{V}}^2 = \sum_{m=1}^{\infty} \{b(u, v_m) - l(v_m)\}^2.$

- VPINN loss: $\mathcal{L}(u) = \|r_M\|_{\mathbb{V}}^2 = \sum_{m=1}^{M} \{b(u, v_m) - l(v_m)\}^2.$

Experiment #1: Simple smooth solution in 1D

$$u^* = \sin(4x)\sin\left(\frac{x}{2}\right)$$

Discretization setup

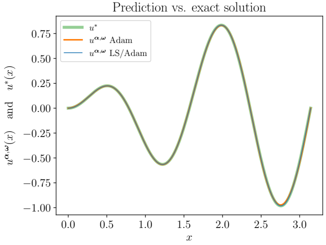
- Fully-connected NN with 3 hidden layers:

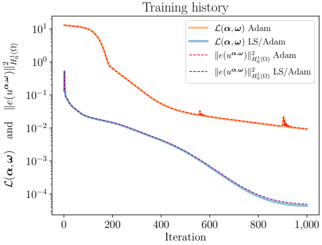
$$x \in \mathbb{R}^{d=1} \to \mathbb{R}^{16} \to \mathbb{R}^{16} \to \mathbb{R}^{N=16} \to \mathbb{R} \ni u^{\boldsymbol{\alpha},\mathbf{c}}(x)$$

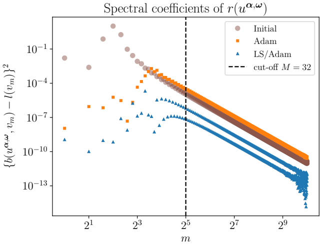
- Activation function: tanh
- Fourier modes (test space discretization): $M = 32$
- Iterations: 1,000
- Learning rate: 10^{-3}

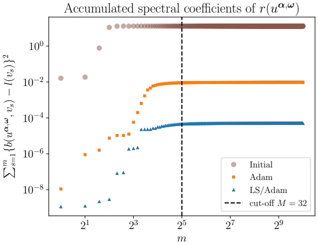
We train two independent and equally initialized neural networks employing either Adam or LS/Adam optimizers.

Experiment #1: Simple smooth solution in 1D









- **Relative errors:** 2.72% (Adam), 0.19% (LS/Adam).
- Huge difference between LS/Adam and Adam alone from the very first training iteration
- Significant decrease difference of first Fourier modes when using LS/Adam compared to Adam alone.

Experiment #2: Smooth solution with high frequency in 1D

$$u^* = \sin(40x)\sin\left(\frac{x}{2}\right)$$

Discretization setup

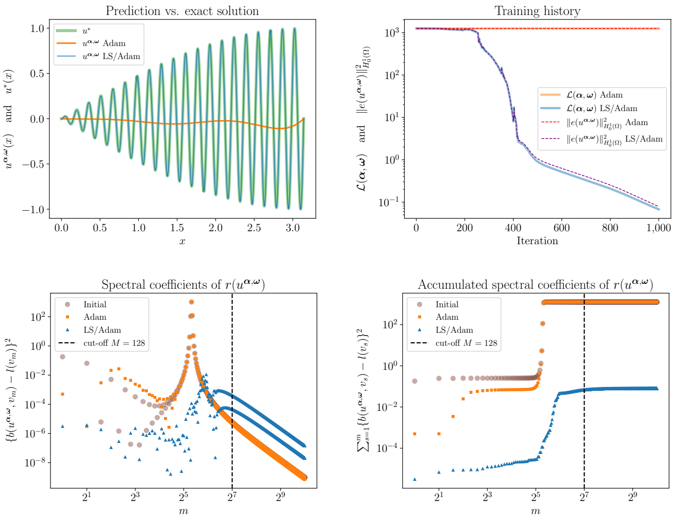
- Fully-connected NN with 3 hidden layers:

$$x \in \mathbb{R}^{d=1} \to \mathbb{R}^{64} \to \mathbb{R}^{64} \to \mathbb{R}^{N=64} \to \mathbb{R} \ni u^{\boldsymbol{\alpha},\mathbf{c}}(x)$$

- Activation function: tanh
- Fourier modes (test space discretization): $M = 128$
- Iterations: 1,000
- Learning rate: 10^{-3}

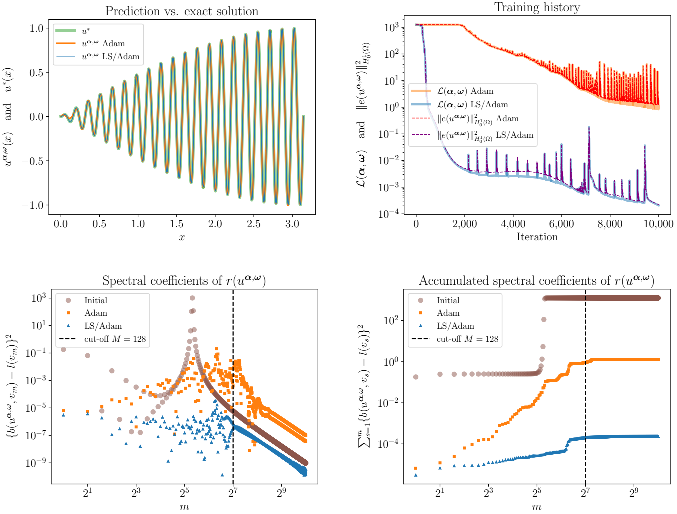
We train two independent and equally initialized neural networks employing either Adam or LS/Adam optimizers.

Experiment #2: Smooth solution with high freq. in 1D (V1)



- **Relative errors:** 99.99% (Adam), 0.79% (LS/Adam).
- LS/Adam is able to decrease the loss, while Adam alone is not. This occurs due to an "slow" spectral bias of Adam.
- LS/Adam accelerates the overcoming of Adam's spectral bias.

Experiment #2: Smooth solution with high freq. in 1D (V2)



- **Relative errors:** 3.28% (Adam), 0.04% (LS/Adam).
- In the long-term run, Adam alone is able to “discover” (and therefore diminish) the most significant nodes.
- LS/Adam accelerates the overcoming of this spectral bias notably.

Experiment #3: Smooth solution with high freq. in 2D

$$u^* = \sin(10x)\sin(2y)e^{\frac{x+y}{2}}$$

Discretization setup

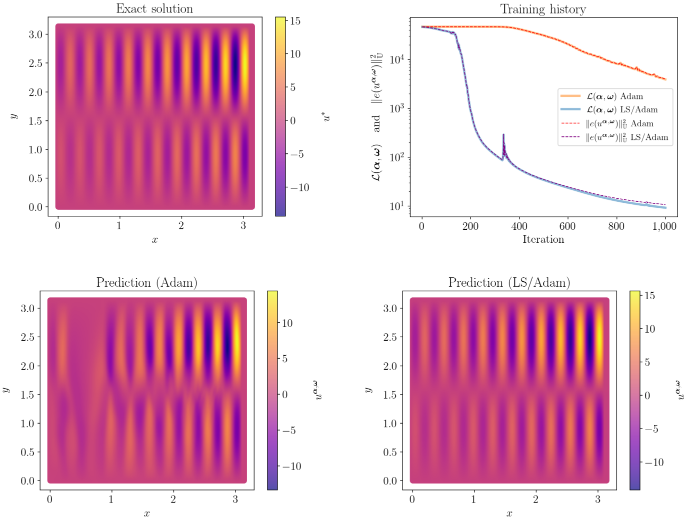
- Fully-connected NN with 3 hidden layers:

  $$(x, y) \in \mathbb{R}^{d=2} \to \mathbb{R}^{N=128}(3 \text{ times}) \to \mathbb{R} \ni u^{\boldsymbol{\alpha},\mathbf{c}}(x, y)$$

- Activation function: tanh
- Fourier modes: $M = 64 \times 16$
- Iterations: 1,000
- Learning rate: 10^{-3}

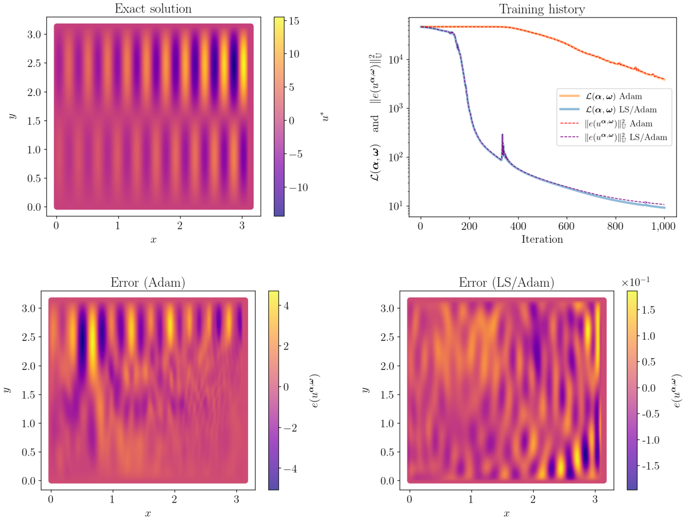
We train two independent and equally initialized neural networks employing either Adam or LS/Adam optimizers.

Experiment #3: Smooth solution with high freq. in 2D



- **Relative errors:** 29.24% (Adam), 1.51% (LS/Adam).
- We observe a similar behavior as in the 1D case.

Experiment #3: Smooth solution with high freq. in 2D



- **Relative errors:** 29.24% (Adam), 1.51% (LS/Adam).
- We observe a similar behavior as in the 1D case.

References

Cyr, E. C., Gulian, M. A., Patel, R. G., Perego, M., & Trask, N. A. (2020). Robust training and initialization of deep neural networks: An adaptive basis viewpoint. In Mathematical and Scientific Machine Learning (pp. 512-536). PMLR.

Uriarte, C., Bastidas, M., Pardo, D., Taylor, J. M., & Rojas, S. (2024). Optimizing variational physics-informed neural networks using least squares. arXiv preprint arXiv:2407.20417.

Baharlouei, S., Taylor, J. M., Uriarte, C., & Pardo, D. (2024). A Least-Squares-Based Neural Network (LS-Net) for Solving Linear Parametric PDEs. arXiv preprint arXiv:2410.15089.