

ScimBa Feel++ Wrapper

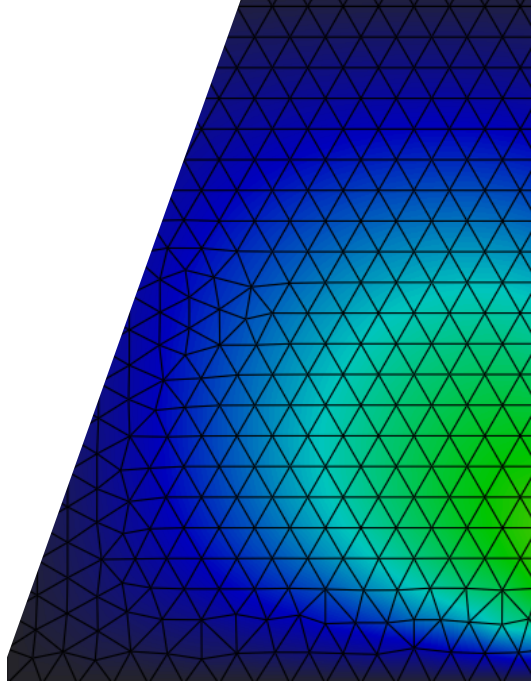
Rayen Tlili

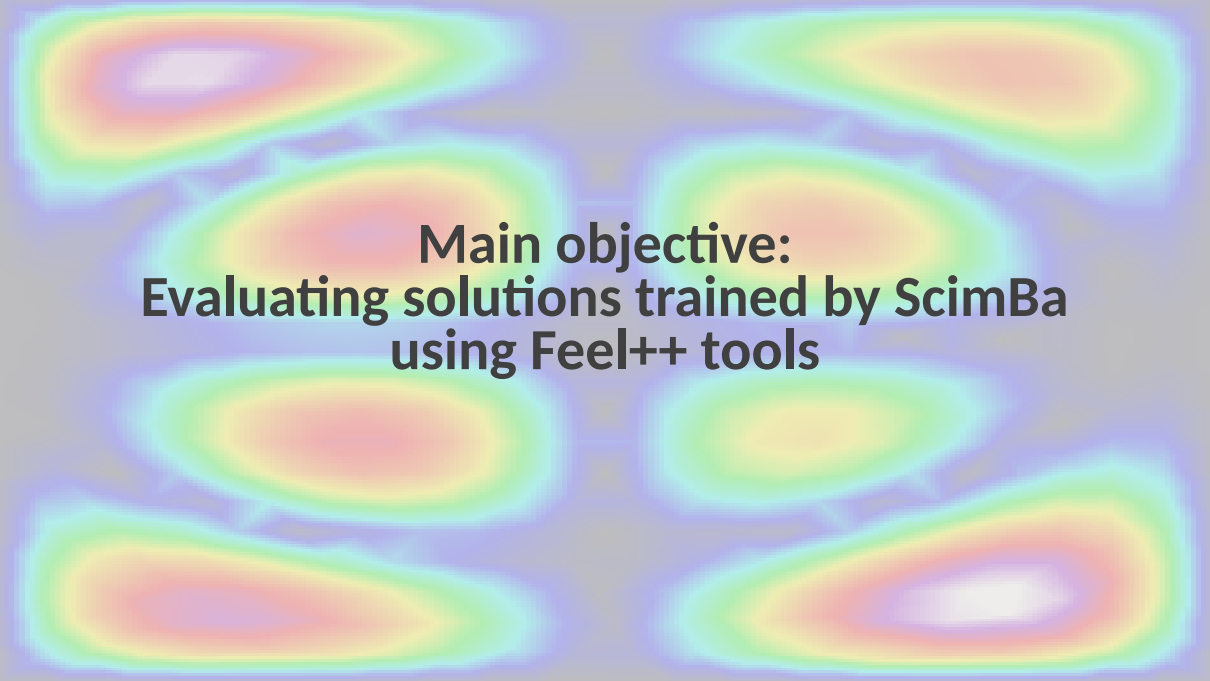
Supervisors:

Joubine Aghili

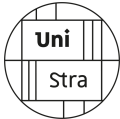
Christophe Prud'homme

August 23, 2024





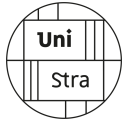
**Main objective:
Evaluating solutions trained by ScimBa
using Feel++ tools**



The main objective

Introduction

- Evaluating solutions trained by **Scimba** using **Feel++** tools .
 1. **Feel++** : Feel++ is a C++ and python library designed for solving partial differential equations (PDEs) using finite element methods.
 2. **Scimba** : ScimBa is a Python library designed to solve complex PDEs using neural networks.



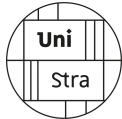
Roadmap

Introduction

1. Create a Poisson class that uses both solvers to solve Poisson PDEs.
2. Visualize and compare the results of both solvers with exact solutions.
3. Expand the use for variable anisotropy
4. Compute L2 and H1 errors and trace their convergence for both solvers..



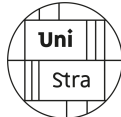
The work environment



The coefficient forms in PDE (Partial Differential Equation) toolboxes encapsulate essential properties such as diffusion, convection, and reaction coefficients.

$$-\nabla \cdot (c \nabla u) + au = f$$

1. c represents the diffusion coefficient
2. u is the unknown function
3. a is the reaction coefficient
4. f denotes the source term.



Using PINNs

pinns

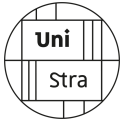
Physics-Informed Neural Networks (PINNs)¹ are used for solving partial differential equations (PDEs) of the form:

$$\mathcal{N}[u(\mathbf{x}, t)] = f(\mathbf{x}, t),$$

where $\mathcal{N}$ is a differential operator and f is a source term. The total loss function, which combines data loss and physics loss, is given by:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{data}}(\theta) + L_{\text{physics}}(\theta),$$

¹M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations," *Journal of Computational Physics*, vol. 378, pp. 686-707, 2019.

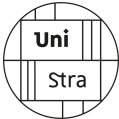


Creating a Docker container and image

Docker

Creating the Docker container

```
1 # Start with the Feel++ base image
2 FROM ghcr.io/feelpp/feelpp:jammy
3
4 # Set labels for metadata
5 LABEL maintainer="Rayen Tlili <rayen.tlili@etu.unistra.fr>"
6 LABEL description="Docker image with Feel++ & ScimBa"
7
8 USER root
9 # install system dependencies
10 RUN apt-get update && apt-get install -y \
11     git \
12     xvfb
13 # Install Python libraries
14 RUN pip3 install torch xvfbwrapper pyvista plotly panel ipykernel
15 matplotlib tabulate nbformat gmsh
```

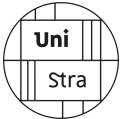


Initializing the environment

Docker

```
1 # Clone the Scimba repository
2 RUN git clone https://gitlab.inria.fr/scimba/scimba.git
3   /workspaces/2024-stage-feelpp-scimba
4
5 # Install Scimba and its dependencies
6 WORKDIR /workspaces/2024-stage-feelpp-scimba/scimba
7 RUN pip3 install scimba
8
9 # Copy the xvfb script into the container for visualization
10 COPY tools/load_xvfb.sh /usr/local/bin/load_xvfb.sh
11 RUN chmod +x /usr/local/bin/load_xvfb.sh
12
13 # Set the script to initialize the environment
14 CMD [ "/usr/local/bin/load_xvfb.sh" ]
```

Listing: Dockerfile for Feel++, Scimba, and Python libraries.

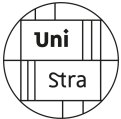


Docker container advantages

Docker

key advantages:

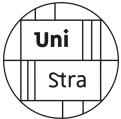
1. **Portability**
2. **Isolation**
3. **Reproducibility**
4. **Dependency Management**



Container limitations

Docker

- Needs access to root user
- Slow to build
- Often have to install scimba by hand inside the container



Setting the environment

Github

Provided in the documentation are the steps necessary to set up the work environment.

Launch

Follow these steps to get the project up and running on your local machine:

Open the project in Visual Studio Code:

Clone the repository

```
git clone https://github.com/master-csmi/2024-m1-scimba-feelpp.git
```

To build a Docker image:

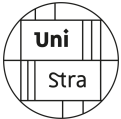
```
docker buildx build -t feelpp_scimba:latest .
```

Run the Docker container

```
docker run -it feelpp_scimba:latest
```

#VS Code will detect the .devcontainer configuration and prompt you to reopen the folder in a container

Figure: ReadMe file in the repository



Setting the environment

Feel++

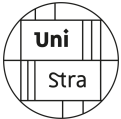
Create the right environment for using the CFPDE toolbox:

```
1 import sys
2 import feelpp
3 import feelpp.toolboxes.core as tb
4
5 from tools.solvers import Poisson
6 sys.argv = [ "feelpp_app" ]
7 e = feelpp.Environment( sys.argv ,
8                         opts=tb.toolboxes_options( "coefficient-form-pdes" ,
9                                                    "cfpdes" ) ,
10                        config=feelpp.globalRepository( 'feelpp_cfpde' ) )
```

Listing: CFPDE initialization and configuration.



Methodology and results



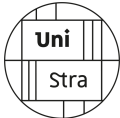
The Poisson class

Feel++

To solve the Poisson equation, we create the environment with Feel++ and configure the settings accordingly.

```
1 P = Poisson(dim = 2)
2 P(h=0.05,           # mesh size
3   order=1,          # polynomial order
4   name='u',          # name of the variable u
5   rhs='8*pi*pi*sin(2*pi*x)*sin(2*pi*y)', # right hand side
6   diff='{1,0,0,1}', # diffusion matrix
7   g='o',             # Dirichlet boundary conditions
8   gN='o',            # Neumann boundary conditions
9   shape='Rectangle', # domain shape (Rectangle, Disk)
10  geofile=None,      # geometry file
11  plot=1,            # plot the solution
12  solver='feelp',    # solver
13  u_exact='sin(2 * pi * x) * sin(2 * pi * y)',
14  grad_u_exact='{2*pi*cos(2*pi*x)*sin(2*pi*y),2*pi*sin(2*pi*x)*cos(2*pi*y)}',
15 }
```

15/34



Generating visuals on the same mesh

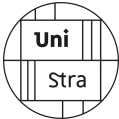
Feel++scimba

By visualizing both solutions on the same graph, we can directly compare the accuracy and differences between the two methods.

```
0 cfpdes.expr.grad_u_exact
1 cfpdes.expr.rhs
2 cfpdes.expr.u_exact
3 cfpdes.poisson.u
Number of features in coordinates: 3
Number of points: 517

Nodes from export.case: [[0.82477343 0.04606718]
[0.8300841 0.10191753]
[0.7806651 0.09123866]
...
[0.8390992 0.3016979 ]
[0.8367227 0.1427201 ]
[0.7476512 0.5844005 ]]
```

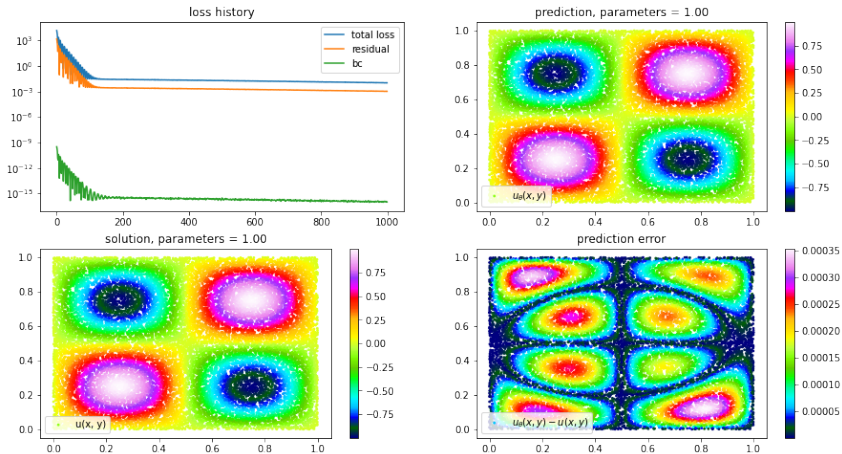
Figure: mesh information visualized on the mesh coordinates.

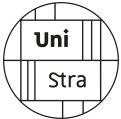


Generating visuals using ScimBa

ScimBa

ScimBa visual representation:





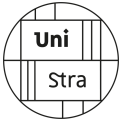
Visualizing the solution for a Laplacian problem

+++++

This segment focuses on visualizing the solutions to the Laplacian problem on a square domain. We compare the numerical accuracy and visual fidelity of the solutions using both Feel++ and Scimba solvers.

```
1 P = Poisson(dim = 2)
2
3 # for square domain
4 u_exact = 'sin(2*pi*x) * sin(2*pi*y)'
5 rhs = '8*pi*pi*sin(2*pi*x) * sin(2*pi*y)'
6
7 P(rhs=rhs, g='o', order=1, solver='feelpp', u_exact = u_exact)
8 P(rhs=rhs, g='o', order=1, solver='scimba', u_exact = u_exact)
```

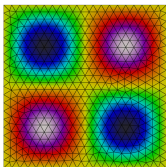
Listing: Using the class for a Laplacian problem.



Visualizing the solution for a Laplacian problem

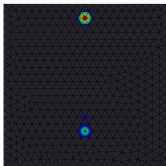
+++++

Feel++ Solution
clim = (-0.998092, 0.99793965)



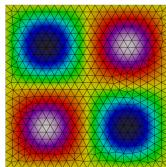
-0.998 -0.499 -7.52e-05 0.499 0.998

$|u_{\text{exact}} - u_{\text{feel}}| / |u_{\text{exact}}|$
clim = (5.370097e-07, 644418.4)



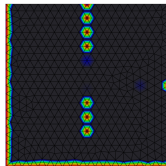
5.37e-07 1.61e+05 3.22e+05 4.83e+05 6.44e+05

Scimba Solution
clim = (-0.9982630545398806, 0.9980093489073726)



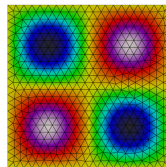
-0.998 -0.499 -0.000137 0.499 0.998

$|u_{\text{exact}} - u_{\text{scimba}}| / |u_{\text{exact}}|$
clim = (1.4726030781812534e-06, 1.1599423670972537)



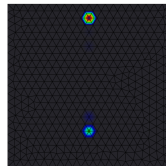
1.47e-06 0.290 0.580 0.870 1.16

Exact Solution
clim = (-0.9981019, 0.9981018)



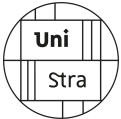
-0.998 -0.499 -5.96e-08 0.499 0.998

$|u_{\text{scimba}} - u_{\text{feel}}| / |u_{\text{exact}}|$
clim = (0.0, 644417.3571176253)



0.00 1.61e+05 3.22e+05 4.83e+05 6.44e+05

Figure: Laplacian on square domain

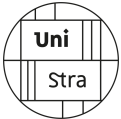


Error convergence rate

+++++

```
1 def runLaplacianPk(df, model, verbose=False):
2     """generate the Pk case"""
3     meas = dict()
4     dim, order, json = model
5     for h in df['h']:
6         m = laplacian(hsize=h, json=json, dim=dim, verbose=verbose)
7         for norm in ['L2', 'H1']:
8             meas.setdefault(f'P{order}-Norm_laplace_{norm}-error', [])
9             meas[f'P{order}-Norm_laplace_{norm}-error'].append(
10                 m.pop(f'Norm_laplace_{norm}-error'))
11     df = df.assign(**meas)
12     return df
```

Listing: Calculating L2 and H1 errors for different mesh sizes.



Tracing the convergence rate

+++++

Convergence rate for the 2D Poisson problem

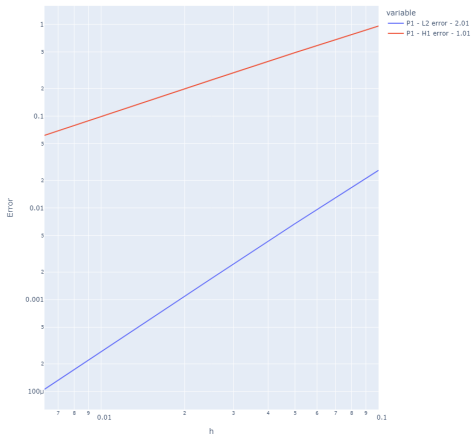
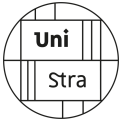


Figure: Error rate for 2D Poisson problem



Absolute Error - ScimBa Solver

+++++

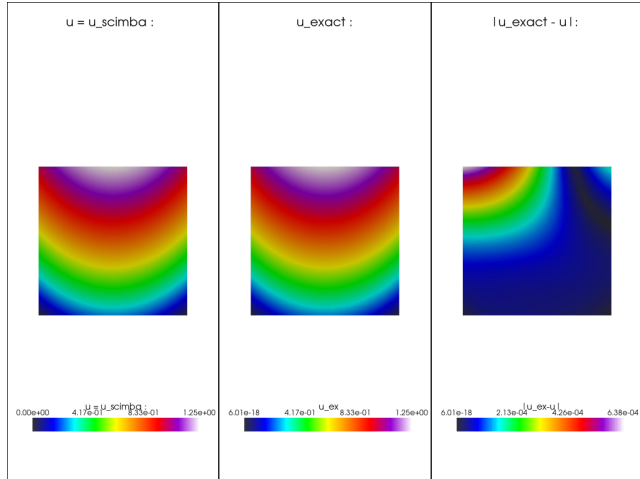
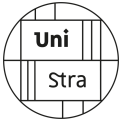


Figure: Absolute Error - ScimBa Solver



Absolute Error - Feel++ Solver

+++++

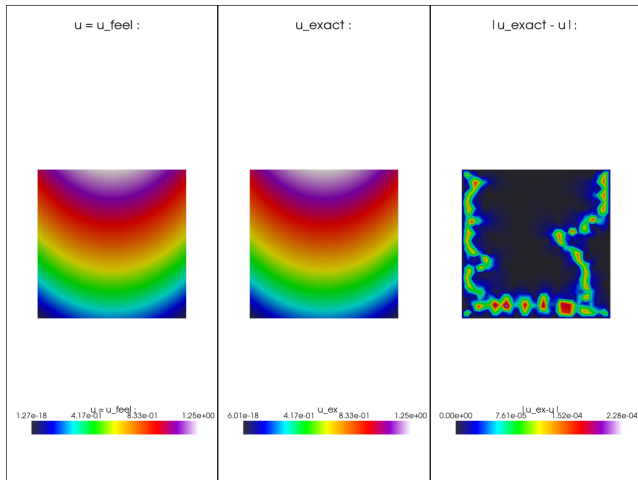
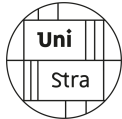


Figure: Absolute Error - Feel++ Solver



Example: Poisson Equation on a Disk Domain

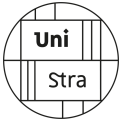
+++++

Consider the exact solution:

$$u_{\text{exact}} = 1 - x^2 - y^2$$

The Poisson equation with a source term $f = 4$ and Dirichlet boundary condition $g = 0$ is solved using ScimBa on a disk domain:

$$P(\text{rhs} = '4', g = '0', \text{shape} = 'Disk', \text{solver} = 'scimba', u_{\text{exact}} = u_{\text{exact}})$$



Example: Poisson Equation on a Disk Domain

+++++

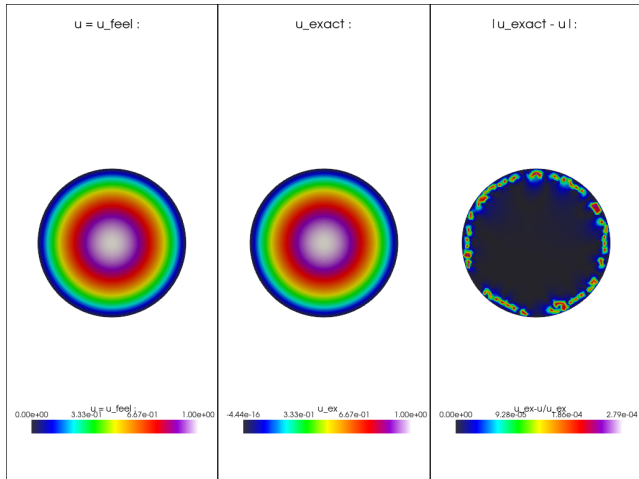
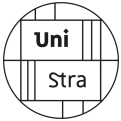


Figure: Absolute Error - Feel++ Solver



Example: Poisson Equation on a Disk Domain

+++++

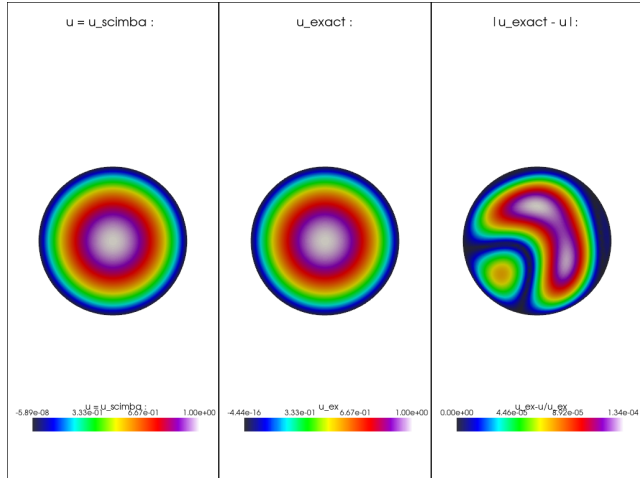
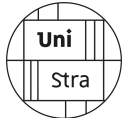


Figure: Absolute Error - ScimBa Solver



Example with Varying Anisotropy

+++++

We consider the following mathematical system: The exact solution is given by:

$$u_{\text{exact}} = \frac{x^2}{1+x} + \frac{y^2}{1+y}$$

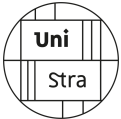
The corresponding Poisson equation with a source term f and diffusion matrix D is given by:

$$-\nabla \cdot (D \nabla u) = f \quad \text{in } \Omega$$

where the source term f and the diffusion matrix D are defined as:

$$f = -\frac{4 + 2x + 2y}{(1+x)(1+y)}$$

$$D = \begin{pmatrix} 1+x & 0 \\ 0 & 1+y \end{pmatrix}$$



Example with Varying Anisotropy

+++++

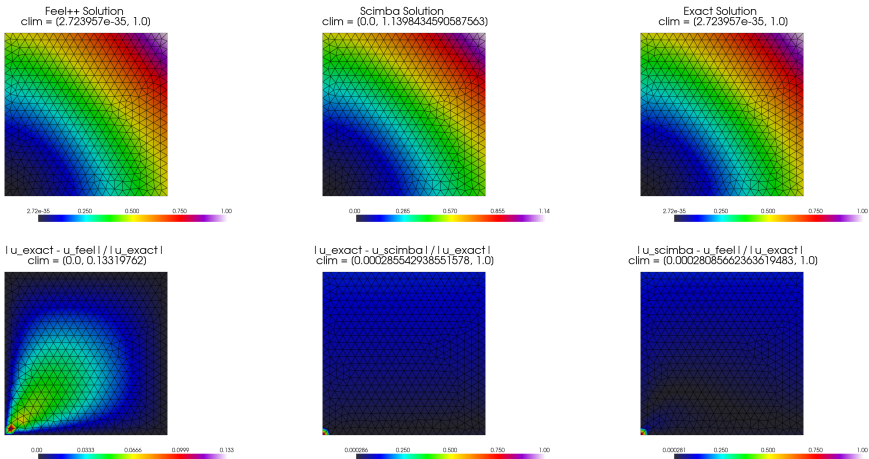
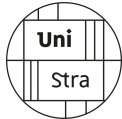


Figure: Example with varying anisotropy



Example with Varying Anisotropy

+++++

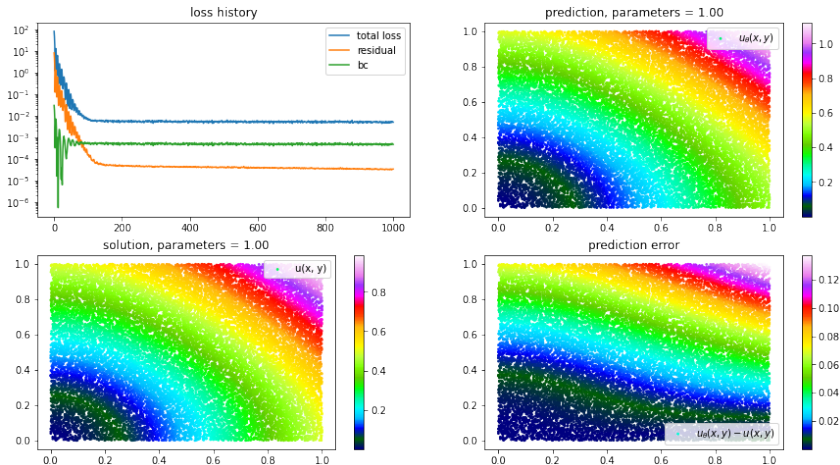
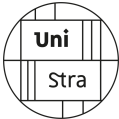
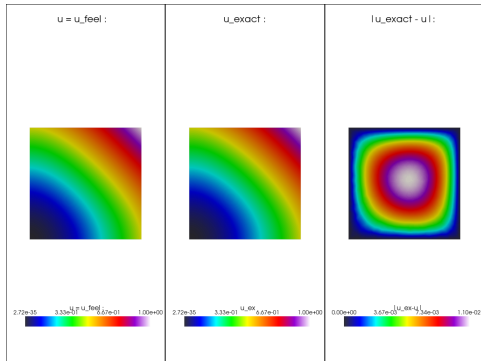


Figure: Varying anisotropy loss function

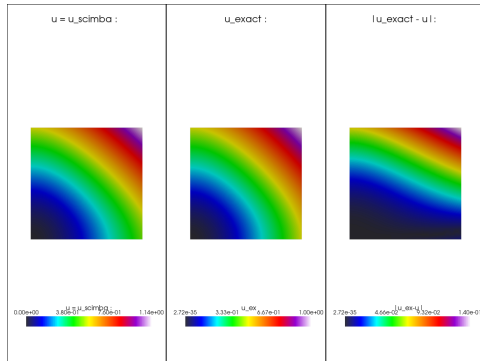


Example with Varying Anisotropy

+++++

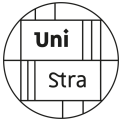


(a) Anisotropy - Feel



(b) Anisotropy - ScimBa

Figure: Comparison of varying anisotropy loss function

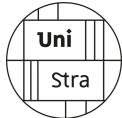


Conclusion

+++++

This internship builds on previous work and succeeds in fixing some of the previous issues and adding new capabilities but fails in certain aspects.

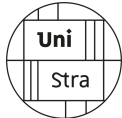
The combined tools allowed for efficient analysis, insightful visualizations, and a deeper understanding of machine learning and finite element methods.



Bibliography (Part 1)

o

- [1] Wikipedia. (n.d.). *Coupling (computer programming)*. Retrieved from [https://en.wikipedia.org/wiki/Coupling_\(computer_programming\)](https://en.wikipedia.org/wiki/Coupling_(computer_programming))
- [2] Feel++. (n.d.). *Finite method course*. Retrieved from <https://feelpp.github.io/cours-edp/#/>
- [3] Feel++. (n.d.). *Feel++ Documentation*. Retrieved from <https://docs.feelpp.org/user/latest/index.html>
- [4] Feel++. (n.d.). *Feel++ GitHub Repository*. Retrieved from <https://github.com/feelpp/feelpp>
- [5] Feel++. (n.d.). *Python Feel++ Toolboxes*. Retrieved from <https://docs.feelpp.org/user/latest/python/pyfeelpptoolboxes/index.html>



Bibliography (Part 2)

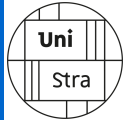
o

- [6] ScimBa. (n.d.). *ScimBa Repository*. Retrieved from <https://gitlab.inria.fr/scimba/scimba>

- [7] SciML. (n.d.). *ScimBa*. Retrieved from <https://sciml.gitlabpages.inria.fr/scimba/>

- [8] SciML. (n.d.). *Laplacian 2D Disk*. Retrieved from <https://sciml.gitlabpages.inria.fr/scimba/examples/laplacian2DDisk.html>

- [9] Feel++. (n.d.). *Quick Start with Docker*. Retrieved from <https://docs.feelpp.org/user/latest/using/docker.html>



Thank you for listening!
Any questions?