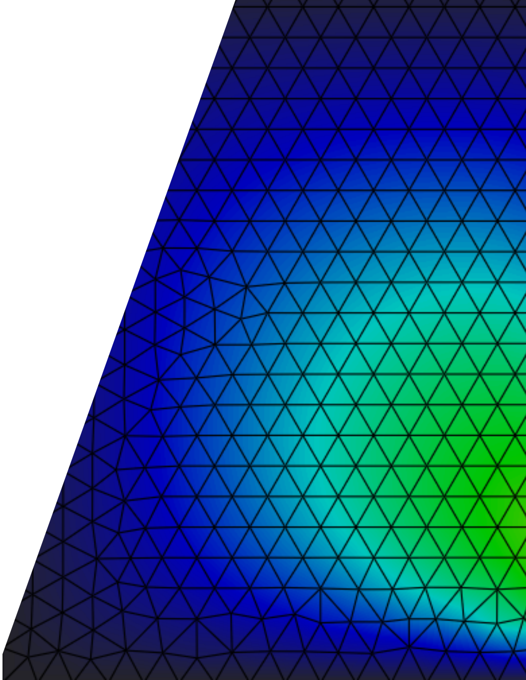
# ScimBa Feel++ Wrapper

**Rayen Tlili**

August 22, 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

# CFPDE toolbox

## cfpde

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


Physics-Informed Neural Networks (PINNs)[1] incorporate the physical laws described by PDEs into the neural network training process by including the residuals of the PDEs in the loss function. For a PDE 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:

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

This approach allows the neural network to respect the underlying physical laws during training.

# Creating a Docker container and image

## Docker

Creating the Docker container

```
# Start with the Feel++ base image
FROM ghcr.io/feelpp/feelpp:jammy

# Set labels for metadata
LABEL maintainer="Rayen Tlili <rayen.tlili@etu.unistra.fr>"
LABEL description="Docker image with Feel++ & ScimBa"

USER root
# install system dependencies
RUN apt-get update && apt-get install -y \
     git \
    xvfb
# Install Python libraries
RUN pip3 install torch xvfbwrapper pyvista plotly panel ipykernel
matplotlib tabulate nbformat gmsh
```

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

# Initializing the environment

## Docker

```
# Clone the Scimba repository
RUN git clone https://gitlab.inria.fr/scimba/scimba.git
    /workspaces/2024-stage-feelpp-scimba

# Install Scimba and its dependencies
WORKDIR /workspaces/2024-stage-feelpp-scimba/scimba
RUN pip3 install scimba

# Copy the xvfb script into the container for visualization
COPY tools/load_xvfb.sh /usr/local/bin/load_xvfb.sh
RUN chmod +x /usr/local/bin/load_xvfb.sh

# Set the script to initialize the environment
CMD ["/usr/local/bin/load_xvfb.sh"]
```

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

# Docker container advantages



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
```

# Setting the environment

## Feel++

Create the right environment for using the CFPDE toolbox:

```python
import sys
import feelpp
import feelpp.toolboxes.core as tb

from tools.solvers import Poisson
sys.argv = ["feelpp_app"]
e = feelpp.Environment(sys.argv,
                       opts=tb.toolboxes_options("coefficient-form-pdes",
                       "cfpdes"),
                       config=feelpp.globalRepository('feelpp_cfpde'))
```

# Methodology and results

# The Poisson class

## Feel++

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

```
P = Poisson(dim = 2)
P(h=0.05,               # mesh size
  order=1,              # polynomial order
  name='u',             # name of the variable u
  rhs='8*pi*pi*sin(2*pi*x)*sin(2*pi*y)',  # right hand side
  diff='{1,0,0,1}', # diffusion matrix
  g='0',                # Dirichlet boundary conditions
  gN='0',               # Neumann boundary conditions
  shape='Rectangle',# domain shape (Rectangle, Disk)
  geofile=None,         # geometry file
  plot=1,               # plot the solution
  solver='feelpp',      # solver
  u_exact='sin(2 * pi * x) * sin(2 * pi * y)',
  grad_u_exact='{2*pi*cos(2*pi*x)*sin(2*pi*y),2*pi*sin(2*pi*x)*cos(2*pi*y)}'
)
```

# 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.

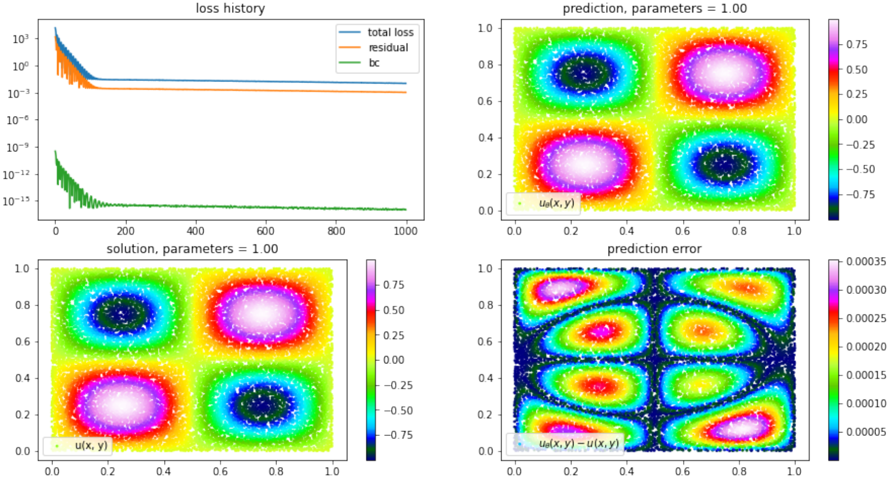
```
0  cfpdes.expr.grad_u_exact
1           cfpdes.expr.rhs
2       cfpdes.expr.u_exact
3          cfpdes.poisson.u
Number of features in coordinates: 3
Number of points: 517

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:



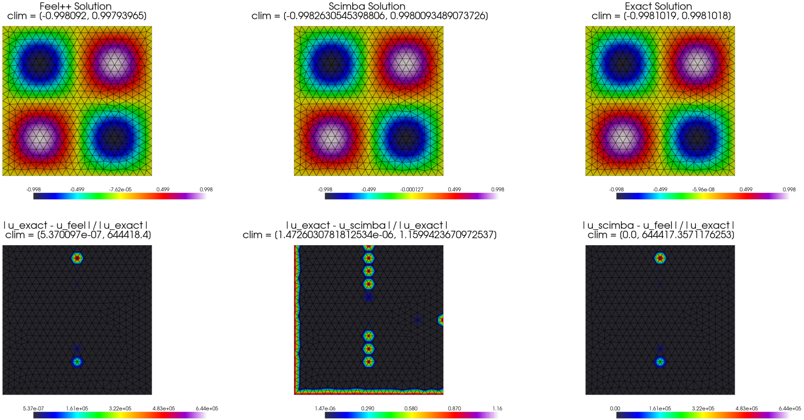
# Visualizing the solution for a Laplacian problem

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.

```
P = Poisson(dim = 2)

# for square domain
u_exact = 'sin(2*pi*x) * sin(2*pi*y)'
rhs = '8*pi*pi*sin(2*pi*x) * sin(2*pi*y)'

P(rhs=rhs, g='0', order=1, solver='feelpp', u_exact = u_exact)
P(rhs=rhs, g='0', order=1, solver ='scimba', u_exact = u_exact)
```

# Visualizing the solution for a Laplacian problem

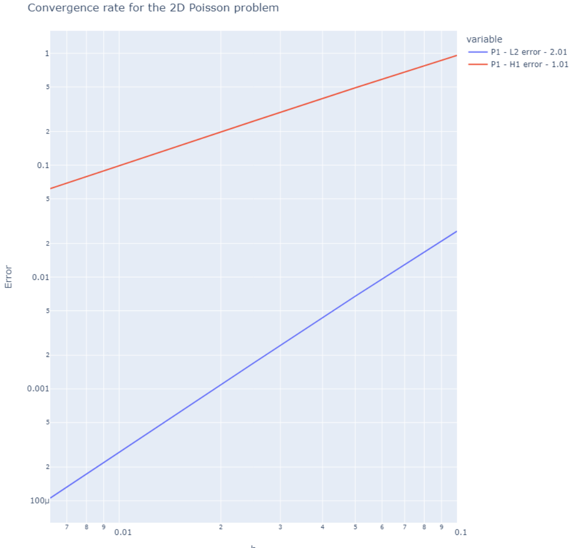
## Laplacian on square



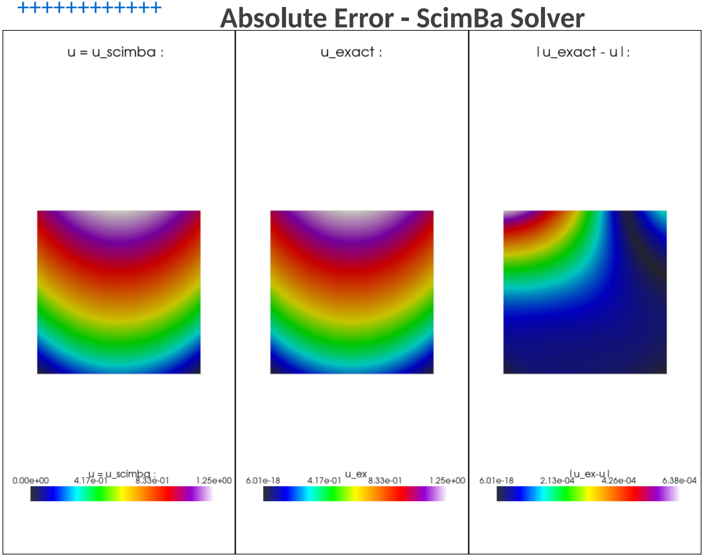
# Error convergence rate

```python
def runLaplacianPk(df, model, verbose=False):
    """generate the Pk case"""
    meas = dict()
    dim, order, json = model
    for h in df['h']:
        m = laplacian(hsize=h, json=json, dim=dim, verbose=verbose)
        for norm in ['L2', 'H1']:
            meas.setdefault(f'P{order}-Norm_laplace_{norm}-error', [])
            meas[f'P{order}-Norm_laplace_{norm}-error'].append(
                m.pop(f'Norm_laplace_{norm}-error'))
    df = df.assign(**meas)
    return df
```

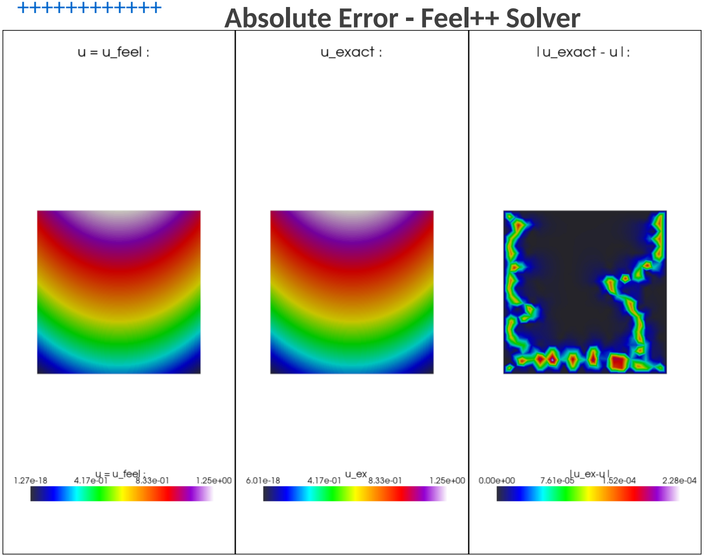
# Tracing the convergence rate



# Absolute Error - ScimBa Solver



# Absolute Error - Feel++ Solver



# Example: Poisson Equation on a Disk Domain

Consider the exact solution:

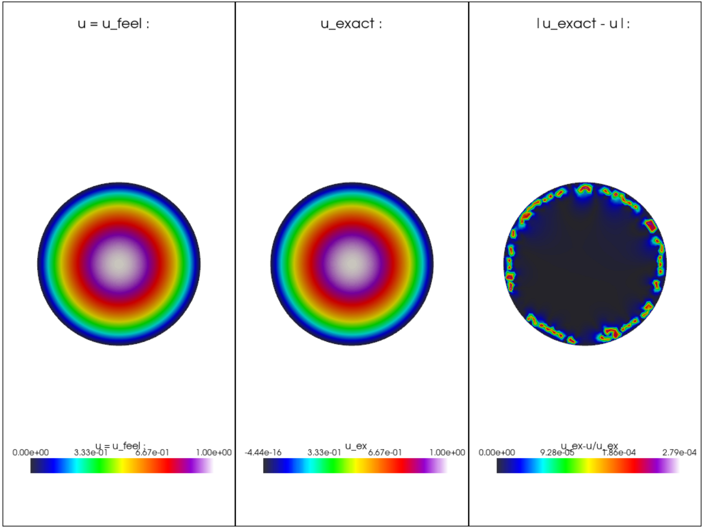
$$u_{\text{exact}} = 1 - x^2 - y^2$$

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

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

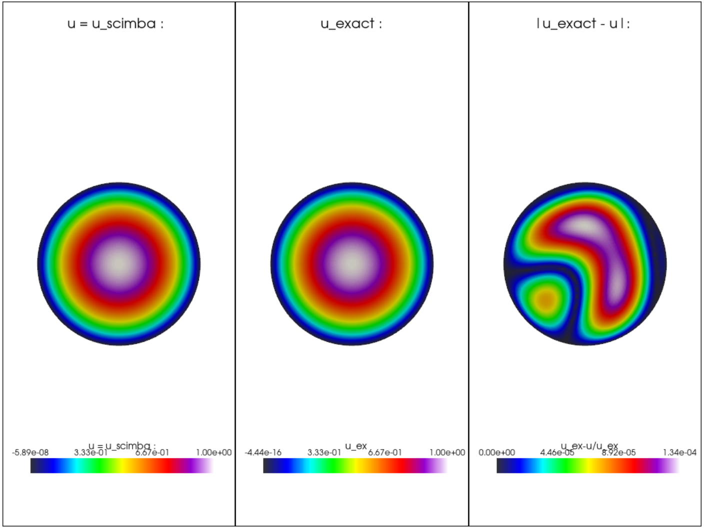
# Example: Poisson Equation on a Disk Domain

## Absolute Error - Feel++ Solver



# Example: Poisson Equation on a Disk Domain

## 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:

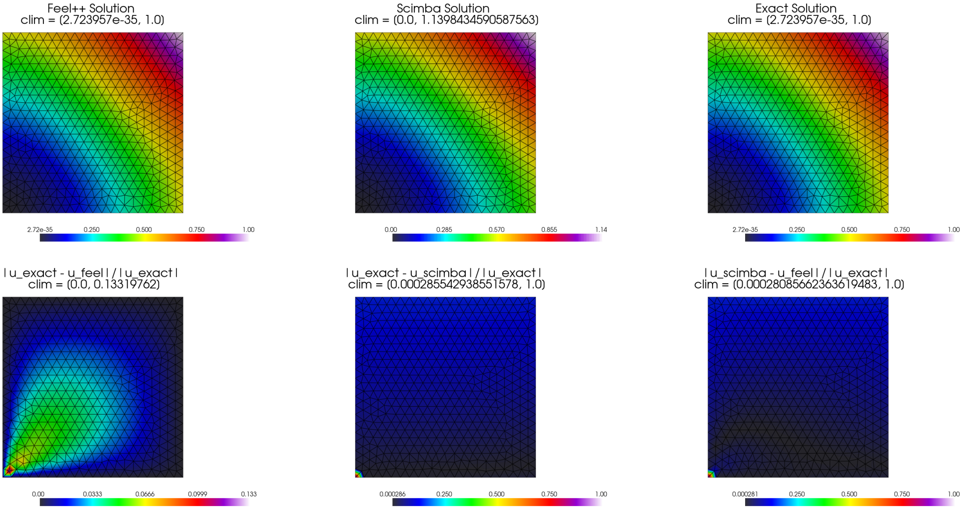
$$-\nabla \cdot (D \nabla u) = f \quad \text{in} \quad \Omega$$

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

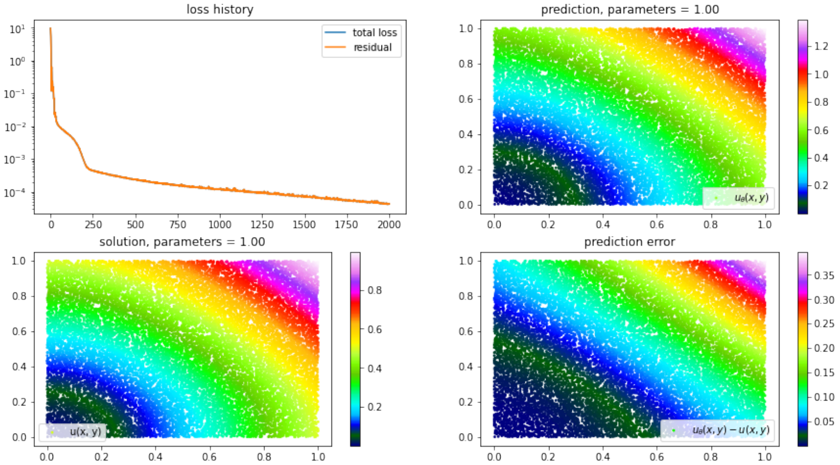
$$f = -\frac{4 + 2x + 2y}{(1+x)(1+y)}$$

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

# Example with Varying Anisotropy



# Example with Varying Anisotropy



# 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)


[1] Wikipedia. (n.d.). *Coupling (computer programming)*. Retrieved from `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)


[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?*