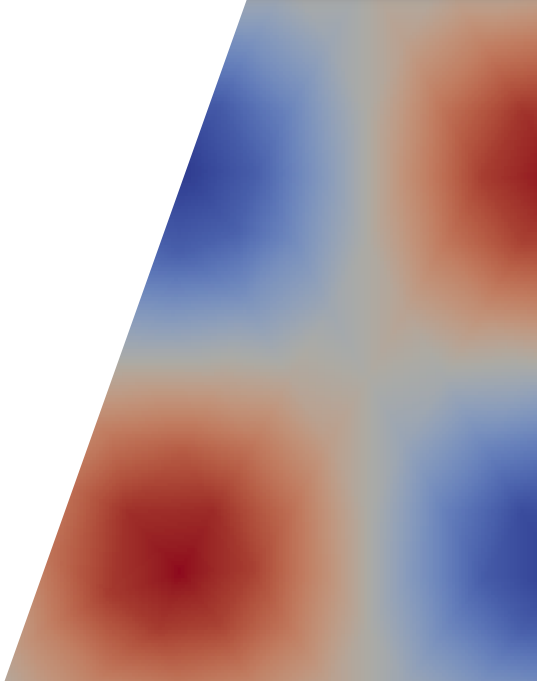
UNIVERSITÉ DE STRASBOURG

Coupling ScimBa and Feel++

Helya Amiri **Rayen Tlili**

May 27, 2024



UNIVERSITÉ DE STRASBOURG

Table of Contents

Project Overview

1 Introduction

This report outlines the goals, strategy, and plan for integrating the Scimba and Feel++ libraries.

- ScimBa focuses on combining machine learning with traditional scientific computing,
- while Feel++ is a C++ library for solving PDEs using Galerkin methods.

Their integration aims to enhance both libraries, enabling more effective solutions to complex scientific problems.

Project Overview

1 Introduction

- Use Docker, Python, and Git to establish a reproducible project environment, applying machine learning, solving PDEs, and managing source code.

UNIVERSITÉ DE STRASBOURG

Table of Contents

2 Objectives

Objectives

- **Generate Multiple Results using Feel++ Toolboxes**: Using the CFDE toolbox to solve Poisson equations with varying parameters and visualizing them on varying geometries.
- **Understanding and Exporting Results using ScimBa**: Explore the ScimBa repository and use examples from Pinns (Physics-informed neural networks) to solve a Poisson equation and visualize the results.

Objectives

2 Objectives

- **Creating a Program that is Able to Use Both as Solvers**: One of the primary objectives to reach for this project is to create a program that is able to call upon the Feel++ CFPDE toolbox and the ScimBa machine learning algorithms to solve various PDEs.
- **Comparing the Results of Both Solvers**: The results of both solvers will be able to be visualized and compared in terms of efficiency and accuracy.

Objectives

- **Expand Application Scope**: After successfully solving Poisson equations, expand the application of the program to solve other types of PDEs, further demonstrating the versatility of the integrated system.
- **Optimize Performance**: Continually optimize the performance of the program, ensuring that it runs efficiently and effectively on various hardware configurations.

Objectives

2 Objectives

User-Friendly Interface: Develop a user-friendly interface that allows users with varying levels of technical expertise to utilize the program effectively.

Documentation and Training: Provide comprehensive documentation and training materials to help users understand how to use the program and interpret the results.

Community Engagement: Engage with the user community to gather feedback, identify areas for improvement, and guide future development efforts.

UNIVERSITÉ DE STRASBOURG

Roadmap

2 Objectives

1. Explore Feel++ and ScimBa documentation.
2. Create a docker container with a Feel++ base.
3. Install dependencies and clone Scimba repository in docker image.
4. Enable image generation in docker.
5. Create a devcontainer.
6. Solve PDEs using Feel++.
7. Solve PDEs using ScimBa using PINNs.
8. Create a Poisson class you can call to solve using Feel++.
9. Add ScimBa as a solver for the class.
10. Update the Poisson2d class to handle ScimBa with parametrized f and g.
11. Add a diffusion tensor to the Poisson2d class.
12. Solve different types of PDEs with both solvers with the same parameters.
13. Compare the results of both solvers.

UNIVERSITÉ DE STRASBOURG

Table of Contents

3 Feel++

Getting familiar with Feel++

3 Feel++

Feel++ is a library for solving Partial Differential Equations (PDEs) and includes toolboxes for physics-based models and their coupling. These toolboxes include applications for:

- Fluid mechanics
- Solid mechanics
- Heat transfer and conjugate heat transfer
- Fluid-structure interaction
- Electro and magnetostatics
- Thermoelectrics

Exploring Feel++ Toolboxes

1. Getting started with toolboxes in Python

Feel++ toolboxes are availabe as python modules. The following toolboxes are available:

Toolbox	Python Module
coefficient form	feelpp.toolboxes.cfpdes
fluid mechanics	feelpp.toolboxes.fluid
heat transfert	feelpp.toolboxes.heat
solid mechanics	feelpp.toolboxes.solid
electric	feelpp.toolboxes.electric
hdg	feelpp.toolboxes.hdg

UNIVERSITÉ DE STRASBOURG

Table of Contents

4 ScimBa

Getting familiar with ScimBa

4 ScimBa

We started using examples from the ScimBa repository that apply Physics-Informed Neural Networks (PINNs). PINNs integrate PDEs into neural networks by creating a loss function that penalizes errors in fitting data and violating physical laws.

```
from lap2D_pinns import Run_laplacian2D, Poisson_2D
from scimba.equations import domain


# Define a square domain
xdomain = domain.SpaceDomain(2, domain.SquareDomain(2, [[0.0, 1.0],
                                                         [0.0, 1.0]]))

# Create an instance of the Poisson problem
pde = Poisson_2D(xdomain)

# Run the training
Run_laplacian2D(pde)
```

Figure: This is achieved by constructing a loss function that penalizes the network for failing to fit known data and for violating the given physical laws.

The class Poisson 2D is initialized with a given spatial domain (space domain) and sets up the problem with one unknown variable and one parameter. The parameter domain is narrowly defined, likely to enforce precise boundary conditions or to stabilize the solution.

```
class Poisson_2D(pdes.AbstractPDEx):
    def __init__(self, space_domain):
        super().__init__(
            nb_unknowns=1,
            space_domain=space_domain,
            nb_parameters=1,
            parameter_domain=[[0.50000, 0.500001]],
        )
```

The Run laplacian2D function encapsulates the entire process of setting up, training, and evaluating a neural network to solve the Laplacian PDE using PINN. This includes data sampling, network configuration, loss calculation, and optimization.

```
def Run_laplacian2D(pde, bc_loss_bool=False, w_bc=0, w_res=1.0):
    x_sampler = sampling_pde.XSampler(pde=pde)
    mu_sampler = sampling_parameters.MuSampler(
        sampler=uniform_sampling.UniformSampling, model=pde
    )
    sampler = sampling_pde.PdeXCartesianSampler(x_sampler, mu_sampler)
```

We will talk further about the files and visuals generated in both cases in the ”Results” section.

UNIVERSITÉ DE STRASBOURG

Table of Contents

5 Docker Setup

Docker

5 Docker Setup

Creating a Docker container and image for the project offers these key advantages:

1. **Portability:** Run the project on any platform supporting Docker.
2. **Isolation:** Avoid conflicts with other software on the host system.
3. **Reproducibility:** Recreate the exact same environment whenever needed.
4. **Dependency Management:** Package all dependencies within the Docker image.

Getting Started

Creating a Docker container and image

- This Dockerfile creates a docker image with Feel++ as a base and installs the dependencies and libraries needed to run ScimBa in that environment.
- It copies the public ScimBa repository into the `scimba` folder and installs it.
- We have also added command lines to automate a script that lets us run the program `solve_lap.py`, which uses Feel++ libraries to solve a Laplacian problem and generates visuals.

Getting Started

5 Docker Setup

To start working : Creating the Docker container

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

# Set labels for metadata
LABEL maintainer="Helya Amiri <helya.amiri@etu.unistra.fr>,
                  Rayen Tlili <rayen.tlili@etu.unistra.fr>"
LABEL description="Docker image with Feel++, Scimba, and Python libraries."

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


# Clone the Scimba repository
RUN git clone https://gitlab.inria.fr/scimba/scimba.git /scimba

# Install Scimba and its dependencies
WORKDIR /scimba
RUN pip3 install .

# Copy the xvfb script into the container
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", "python3"]
```

Listing 1: Dockerfile for Feel++, Scimba, and Python libraries.

UNIVERSITÉ DE STRASBOURG

Table of Contents

6 Methodology

Methodology

6 Methodology

Given the Feel++ documentation and the Poisson class prototype that gives access to results from the Feel++ solver. Create the right environment for using the CFPDE toolbox:

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

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

Methodology

6 Methodology

Solving the Poisson equation with different parameters using the CFPDE toolbox:

```
P = Poisson(dim = 2)
P(h=0.08, rhs='-1.0-1*y*x+y*y', g='0', order=1, geofile='geo/disk.geo',
    plot='2d.png')
P(h=0.1, rhs='-1.0-2*y*x+y*y', g='0', order=1, plot='f2.png')

P = Poisson(dim = 2)
P(h=0.1, diff='{1.0,0,0,x*y}', rhs='1', plot='d1.png')
P(h=0.1, diff='{1+x,0,0,1+y}', rhs='1', plot='d2.png')

P = Poisson(dim = 3)
P(h=0.08, diff='{1,0,0,0,x+1,0,0,0,1+x*y}', g = 'x', rhs='x*y*z',
geofile = 'geo/cube.geo', plot='3d.png')
```

Methodology

6 Methodology

Adding the option to use a different solver when calling the Poisson Class:

```
def __call__(self,
             h,                                          # mesh size
             order=1,                                    # polynomial order
             name='Potential',                           # name of the variable
             rhs='8*pi*pi*sin(2*pi*x)*sin(2*pi*y)',      # right hand side
             diff='{1,0,0,1}',                           # diffusion matrix
             g='0',
             geofile=None,
             plot=None,
             solver='feelpp'):                           # or solver ='scimba'
    """
```

Methodology

6 Methodology

Solving using Feel++ and ScimBa:

```
if solver == 'feelpp':
  print(f"Solving the laplacian problem for hsize = {h}...")
  feelpp_mesh = feelpp.load(feelpp.mesh(dim=self.dim, realdim=self.dim),
                                                    fn, h)
  self.pb.setMesh(feelpp_mesh)
  self.pb.setModelProperties(self.model)
  self.pb.init(buildModelAlgebraicFactory=True)
  self.pb.printAndSaveInfo()
  self.pb.solve()
  self.pb.exportResults()

elif solver == 'scimba':
  print("Solving using Scimba")

  # Define a disk domain

  if geofile == 'geo/disk.geo' :
    xdomain = domain.SpaceDomain(2,
                                 domain.DiskBasedDomain(2,
                                 center=[0.0, 0.0], radius=1.0))
    pde_disk = PoissonDisk2D(xdomain,  rhs= rhs, g= g)
    Run_laplacian2D(pde_disk)

  # Define a square domain

  elif geofile == None:
    xdomain = domain.SpaceDomain(2,
                                 domain.SquareDomain(2,
                                 [[0.0, 1.0], [0.0, 1.0]]))
    pde = Poisson_2D(xdomain,  rhs= rhs, g= g)
    Run_laplacian2D(pde)
```

Methodology

6 Methodology

Solving using ScimBa:

```
P(h=0.08, diff='{1,0,0,0,x+1,0,0,0,1+x*y}', g = 'x', rhs='x*y*z',
    geofile ='geo/cube.geo', plot='3d.png', solver='scimba')
```

Methodology

6 Methodology

- If `new training = False`, the code suggests that you might want to continue using a previously trained and saved model without starting the training from scratch.
- If `new training = True`, it indicates that you want to start fresh, ignoring any previously saved models.

```
new_training = False
#new_training = True
if new_training:
    (
        Path.cwd()
        / Path(training_x.TrainerPINNSpace.FOLDER_FOR_SAVED_NETWORKS)
        / file_name
    ).unlink(missing_ok=True)
```

UNIVERSITÉ DE STRASBOURG

Table of Contents

7 Results

Generating visuals using Feel++

7 Results

Using the `Poisson class` and the `Laplacian2D.py` program,

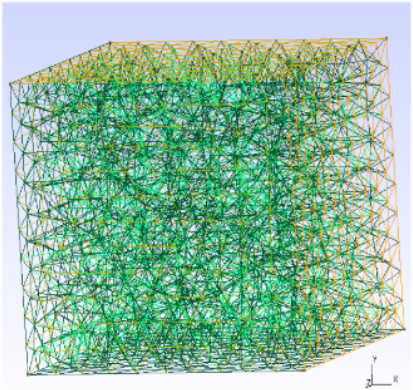
```
def genCube(self, filename, h=0.1):
    """
    Generate a cube geometry following the dimension  self.dim
    """


    geo="""SetFactory("OpenCASCADE");
    h={};
    dim={};
    """.format(h, self.dim)

    if self.dim==2 :
        geo+="""
        Rectangle(1) = {0, 0, 0, 1, 1, 0};
        Characteristic Length{ PointsOf{ Surface{1}; } } = h;
        Physical Curve("Gamma_D") = {1,2,3,4};
        Physical Surface("Omega") = {1};
        """
    elif self.dim==3 :
        geo+="""
        Box(1) = {0, 0, 0, 1, 1, 1};
        Characteristic Length{ PointsOf{ Volume{1}; } } = h;
        Physical Surface("Gamma_D") = {1,2,3,4,5,6};
        Physical Volume("Omega") = {1};
        """
    with open(filename, 'w') as f:
        f.write(geo
```

Generated 3D geometry and mesh viewed using gmsh:

7 Results



Calling the Poisson class specifying the parameters:

```
P = Poisson(dim = 2)
P(h=0.08,  rhs='-1.0-1*y*x+y*y', g='0', order=1, plot='f4.png')
```

```
P(h=0.08,  rhs='-1.0-1*y*x+y*y', g='0', order=1, solver='scimba')
```

Generating visuals using ScimBa

7 Results

```
xdomain = domain.SpaceDomain(2, domain.SquareDomain(2, [[0.0, 1.0],
                                                        [0.0, 1.0]]))
pde = Poisson_2D(xdomain, rhs='8*pi*pi*sin(2*pi*x)*sin(2*pi*y)', g='0')
Run_laplacian2D(pde)
```

Generating visuals using ScimBa

7 Results

```
xdomain = domain.SpaceDomain(2, domain.DiskBasedDomain(2, center=[0.0, 0.0],
                                                        radius=1.0))
pde_disk = PoissonDisk2D(space_domain=xdomain)
Run_laplacian2D(pde_disk)
```

Research and Analysis

7 Results

Key Findings

7 Results

UNIVERSITÉ DE STRASBOURG

Table of Contents

8 Conclusion

UNIVERSITÉ DE STRASBOURG

Conclusion

8 Conclusion

UNIVERSITÉ DE STRASBOURG

Future Work

8 Conclusion

UNIVERSITÉ DE STRASBOURG

Table of Contents

9 References

UNIVERSITÉ DE STRASBOURG

Thank You!

Thank you for listening!
Any questions?