Project report: Coupling ScimBa and Feel++

Helya Amiri[*] & **Rayen Tlili**[†]

Supervised by Christophe Prud'homme, Joubine Aghili

Submitted April 23, 2024

[*]: , helya.amiri@etu.unistra.fr

[†]: , :rayen.tlili@etu.unistra.fr

Contents

Abstract

.

This project report details the integration efforts between ScimBa, emphasizing machine learning, and Feel++, known for its Galerkin methods in PDE solving. The aim is to establish seamless compatibility between the two libraries, fostering advanced computational techniques in scientific research. By combining machine learning with traditional PDE solvers, the project endeavors to propel computational science and engineering forward, enabling efficient data exchange and methodological synergy.

Main Content

1 Introduction

This report presents the objectives, approach, and roadmap for the coupling of ScimBa and Feel++ libraries. ScimBa is a project aimed at integrating machine learning techniques with traditional scientific computing methods, while Feel++ is a C++ implementation of Galerkin methods for solving partial differential equations (PDEs). The coupling of these two libraries is expected to enhance their capabilities and enable researchers to solve complex scientific problems more effectively.

2 Objectives

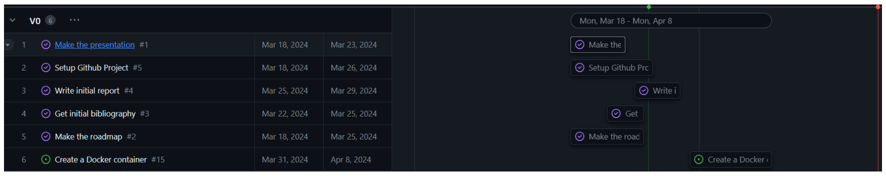
This project endeavors to seamlessly unite the realms of ScimBa and Feel++, fostering a symbiotic relationship that optimally harnesses their individual strengths. At its core, the project aims to achieve the following objectives:

1. **Strategic Analysis and Planning**: Delve into the functionalities of both ScimBa and Feel++ to discern strategic integration points. Develop a meticulous plan that charts the course for establishing loose coupling between these libraries, ensuring a harmonious blend of their capabilities.
2. **Implementation Excellence**: Execute the envisioned integration strategy with precision and finesse. Implement a seamless coupling mechanism between ScimBa and Feel++, meticulously crafting methods to generate datasets within Feel++ and seamlessly integrate them into the ScimBa ecosystem.
3. **Thorough Testing and Validation**: Subject the integrated system to rigorous testing and validation protocols, meticulously scrutinizing every aspect to ensure flawless communication between ScimBa and Feel++. Validate the efficacy of the coupling through the successful resolution of intricate scientific problems, affirming its transformative potential.
4. **Comprehensive Documentation and Reporting**: Document the integration process meticulously, capturing interface definitions and communication protocols with meticulous detail. This documentation serves as a beacon for future endeavors, providing invaluable insights into the integration journey. Additionally, prepare a comprehensive report that encapsulates the project's outcomes, distilling key learnings and insights gleaned along the way.

Through these objectives, the project aspires not merely to couple ScimBa and Feel++ but to forge a dynamic synergy that propels computational science into new realms of innovation and discovery.

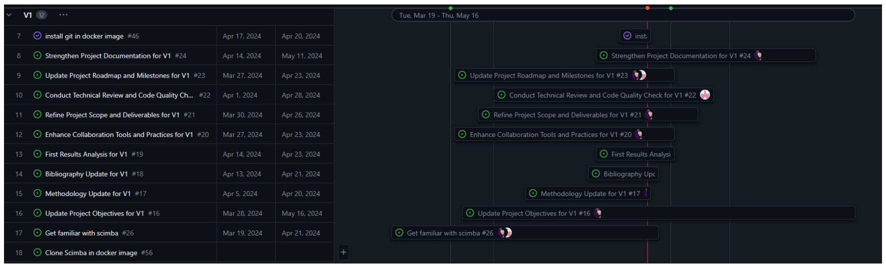
3 V0

3.1 Roadmap



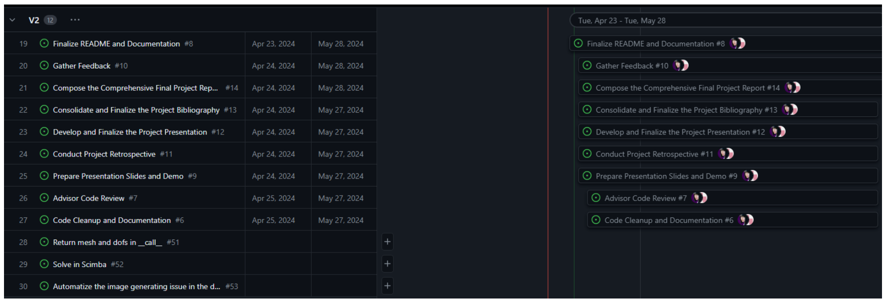
V0 6			
1	Make the presentation #1	Mar 18, 2024	Mar 23, 2024
2	Setup Github Project #5	Mar 18, 2024	Mar 26, 2024
3	Write initial report #4	Mar 25, 2024	Mar 29, 2024
4	Get initial bibliography #3	Mar 22, 2024	Mar 25, 2024
5	Make the roadmap #2	Mar 18, 2024	Mar 25, 2024
6	Create a Docker container #15	Mar 31, 2024	Apr 8, 2024

Figure 1: Roadmap for V0



V1 12			
7	install git in docker image #46	Apr 17, 2024	Apr 20, 2024
8	Strengthen Project Documentation for V1 #24	Apr 14, 2024	May 11, 2024
9	Update Project Roadmap and Milestones for V1 #23	Mar 27, 2024	Apr 23, 2024
10	Conduct Technical Review and Code Quality Ch... #22	Apr 1, 2024	Apr 28, 2024
11	Refine Project Scope and Deliverables for V1 #21	Mar 30, 2024	Apr 26, 2024
12	Enhance Collaboration Tools and Practices for V1 #20	Mar 27, 2024	Apr 23, 2024
13	First Results Analysis for V1 #19	Apr 14, 2024	Apr 23, 2024
14	Bibliography Update for V1 #18	Apr 13, 2024	Apr 21, 2024
15	Methodology Update for V1 #17	Apr 5, 2024	Apr 20, 2024
16	Update Project Objectives for V1 #16	Mar 28, 2024	May 16, 2024
17	Get familiar with scimba #26	Mar 19, 2024	Apr 21, 2024
18	Clone Scimba in docker image #56		

Figure 2: Roadmap for V1



V2 12			
19	Finalize README and Documentation #8	Apr 23, 2024	May 28, 2024
20	Gather Feedback #10	Apr 24, 2024	May 28, 2024
21	Compose the Comprehensive Final Project Rep... #14	Apr 24, 2024	May 28, 2024
22	Consolidate and Finalize the Project Bibliography #13	Apr 24, 2024	May 27, 2024
23	Develop and Finalize the Project Presentation #12	Apr 24, 2024	May 27, 2024
24	Conduct Project Retrospective #11	Apr 24, 2024	May 27, 2024
25	Prepare Presentation Slides and Demo #9	Apr 24, 2024	May 27, 2024
26	Advisor Code Review #7	Apr 25, 2024	May 27, 2024
27	Code Cleanup and Documentation #6	Apr 25, 2024	May 27, 2024
28	Return mesh and dofs in __call__ #51		
29	Solve in Scimba #52		
30	Automatize the image generating issue in the d... #53		

Figure 3: Roadmap for V2

3.2 Creating a Docker container

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.

We're using Feel++ as the base for the Docker container, adding the requirements and dependencies for Scimba and PyTorch.

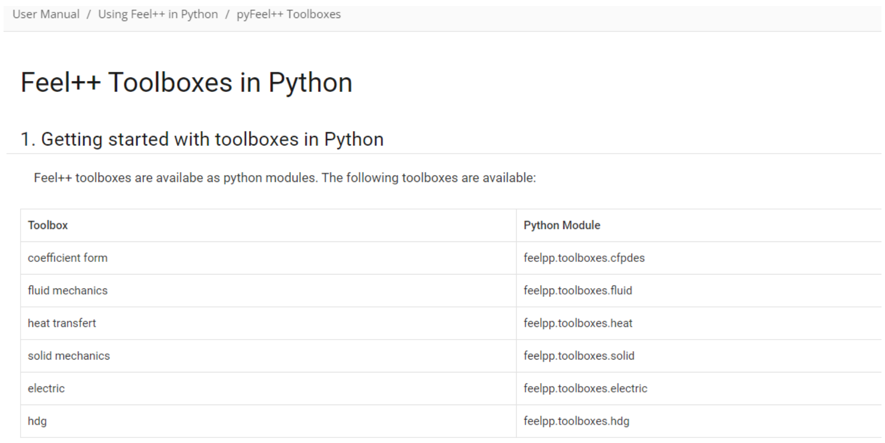
This Dockerfile contains the latest version of Feel++, Scimba, and PyTorch, and should be able to run these commands without error:

```
# 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 PyTorch."

# Install PyTorch.
RUN pip3 install torch

# Copy the local Scimba directory to the container.
COPY scimba/ /scimba/

# Install Scimba and its dependencies
WORKDIR /scimba
USER root

RUN pip3 install .

# Set the default command to launch Python.
CMD ["python3"]
```

Listing 1: Dockerfile for Feel++, Scimba, and PyTorch

3.3 Exploring Feel++ toolboxes

As of the first meeting with the project supervisors, we've taken a look at the different toolboxes Feel++ has to offer in Python: .



User Manual / Using Feel++ in Python / pyFeel++ Toolboxes

Feel++ Toolboxes in Python

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

An interesting toolbox to start with is the **Coefficient Form PDEs**:

3.4 Coefficient Form Toolbox

1. **What are Coefficient Form PDEs?**: The coefficient forms in PDE (Partial Differential Equation) toolboxes encapsulate crucial properties like diffusion, convection, and reaction coefficients. These coefficients are vital for characterizing diverse PDEs such as elliptic, parabolic, or hyperbolic equations, each with its unique coefficient form. For instance, in the Poisson equation, a common elliptic equation, the coefficient form is often expressed as:

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

   - c : represents the diffusion coefficient,
   - a : represents the reaction coefficient,
   - u : is the unknown function, and
   - f : is the source term.

   PDE toolboxes, like Feel++, provide extensive features for managing coefficient forms in diverse PDEs. They simplify tasks such as defining coefficients, setting boundaries, discretizing problems, and employing numerical methods. These capabilities enable researchers and engineers to efficiently tackle complex PDEs, analyze various physical phenomena, and simulate real-world scenarios with ease.

2. **System of PDEs**: A lot of PDE(s) can be written in a generic form, and depends mainly on the definition of coefficients. The generic form that we use is described by the following equation, to find

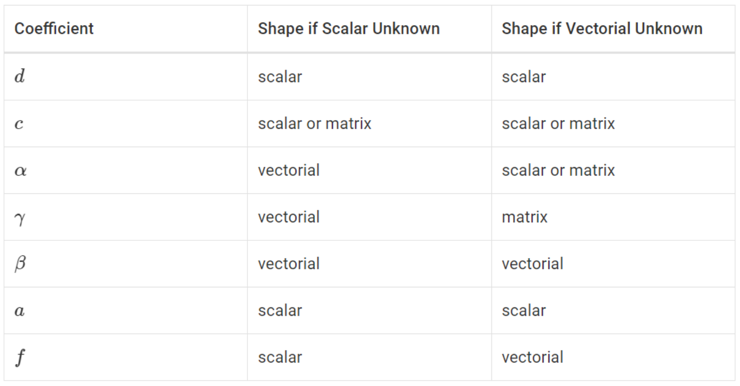
$u : \Omega \subset \mathbb{R}^d \longrightarrow \mathbb{R}^n$ with $d = 2, 3$ and $n = 1$ (u is a scalar field) or $n = d$ (u is a vector field) such that

$$d\frac{\partial u}{\partial t} + \nabla \cdot (-c\nabla u - \alpha u + \gamma) + \beta \cdot \nabla u + au = f \text{ in } \Omega$$

- d : damping or mass coefficient
- c : diffusion coefficient
- α : conservative flux convection coefficient
- γ : conservative flux source term
- β : convection coefficient
- a : absorption or reaction coefficient
- f : source term

Parameters μ may depend on the unknown u and on the space variable x, time t, and other unknowns $u_1, \ldots, u_N$.

3. **Coefficients**: We must also adhere to specific constraints regarding the shape of coefficients, outlined in the following table.



Coefficient	Shape if Scalar Unknown	Shape if Vectorial Unknown
d	scalar	scalar
c	scalar or matrix	scalar or matrix
α	vectorial	scalar or matrix
γ	vectorial	matrix
β	vectorial	vectorial
a	scalar	scalar
f	scalar	vectorial

Figure 4: Shape required by the coefficients

4. **Initial Conditions**: Initial conditions specify the starting values for each unknown variable in the equations. These conditions can be defined using expressions or fields, as detailed in the JSON specifications.
5. **Boundary Conditions**:
    - Dirichlet
    - Neumann
    - Robin
6. **Finite Element Approximation**
7. **Time scheme**:
    - Backward Differences Formula
    - Theta scheme
8. **Stabilized finite element methods**
    - GaLS
    - SUPG
    - Coefficient form PDE toolbox

4 V1

4.1 Objectives for V1

Once the proper environment was set up in the docker image, we started working on a program that will solve various PDEs using the Feel++ method and the Scimba method to visualize and compare the results.

1. **Generate multiple results using Feel++ toolboxes**: Using the CFDE toolbox to solve Poisson equations with varying variables and visualizing them on varying geometries.
2. **Understanding and exporting results using ScimBa**: Explore the ScimBa repository and use examples from Pinns (Physics-informed neural networks) to solve a simple Poisson equation and visualize the results.
3. **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.
4. **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.

After comparing the results of the Poisson equation using both solvers, the work will be working on adapting the program to work on more complex PDEs and other ScimBa neural networks.

4.2 Updating the Docker container

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 script that let us run the program 'solvelap.py', that uses Feel++ libraries to solve a Laplacian problem.

```

# 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 2: Dockerfile for Feel++, Scimba, and Python libraries.

4.3 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'))
```

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')
```

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'
    """
```

Solve using feelpp tools and a space to return the same types of results using 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()

       #measures = self.pb.postProcessMeasures().values()

elif solver == 'scimba':
       print("Solving using Scimba")
      # the solution process using Scimba
        return
```

We would wanna use this do generate results exctracted from 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')
```

4.4 Getting familiar with ScimBa

We decided to start using the examples in the ScimBa repository of uses of the Physics-Informed Neural Networks (PINNs). PINNs integrate the underlying physical laws described by PDEs directly into the learning process of neural networks. 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.

```
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)
```

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]],
        )
```

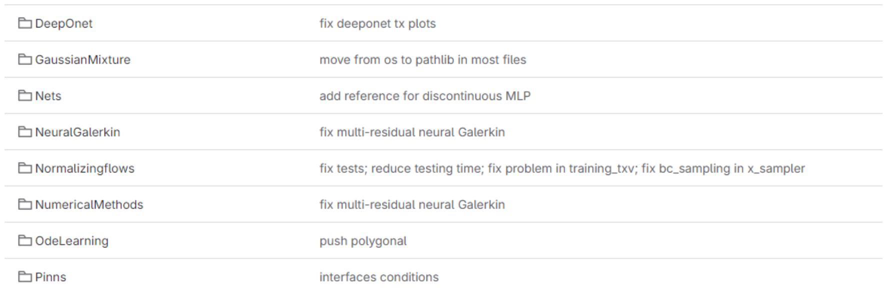
The Run laplacian2D function encapsulates the entire process of setting up, training, and evaluating a neural network to solve the Laplacian PDE using PINN. This includes data sampling, network configuration, loss calculation, and optimization.

```
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)
```

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)
```

We will talk further about the files and visuals generated in both cases in the "Results" section.
Other neural networks available on ScimBa: .



DeepOnet	fix deeponet tx plots
GaussianMixture	move from os to pathlib in most files
Nets	add reference for discontinuous MLP
NeuralGalerkin	fix multi-residual neural Galerkin
Normalizingflows	fix tests; reduce testing time; fix problem in training_txv; fix bc_sampling in x_sampler
NumericalMethods	fix multi-residual neural Galerkin
OdeLearning	push polygonal
Pinns	interfaces conditions

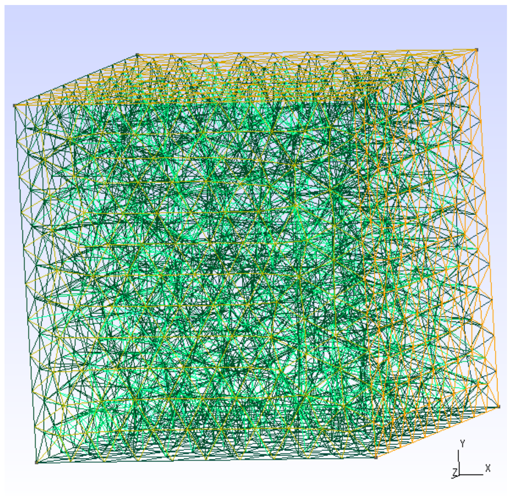
5 Results

5.1 Generating visuals using Feel++

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
```

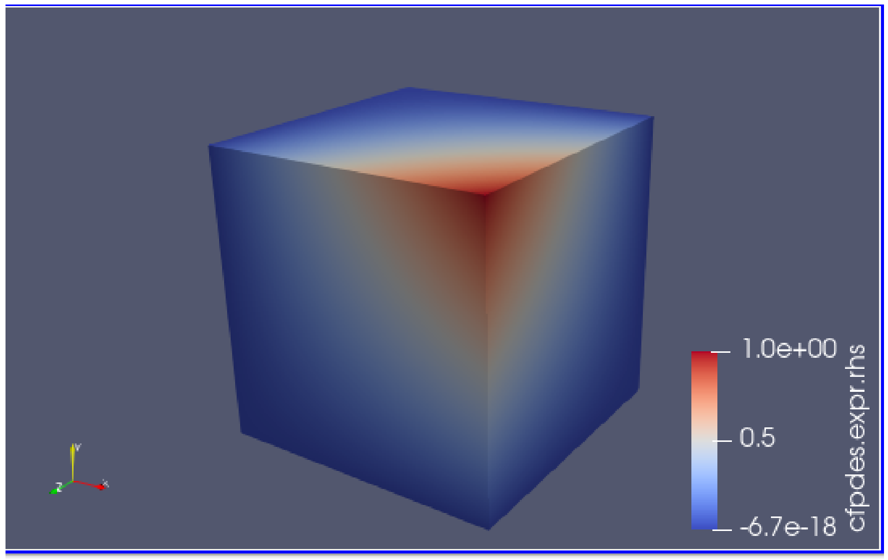
Generated 3D geometry and mesh viewed using gmsh:

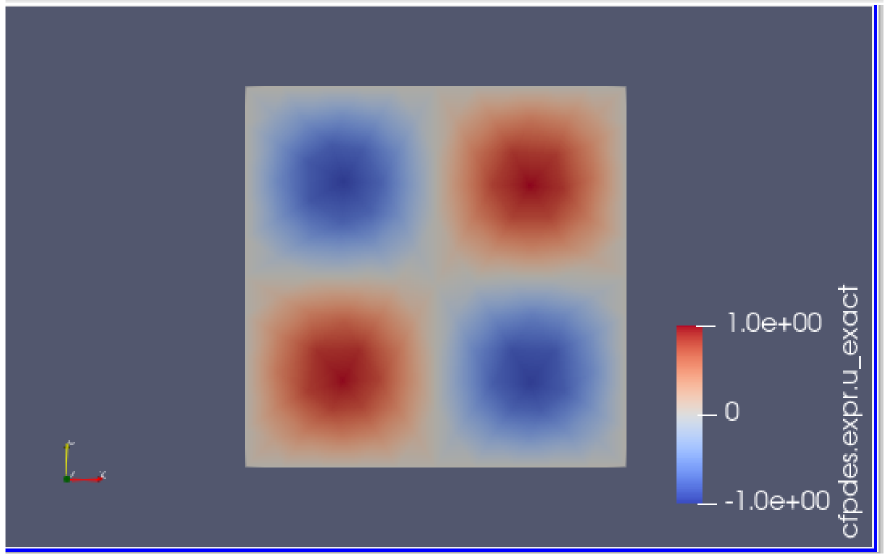


.

Viewing 3D and 2D plotting viewed using paraview:

.

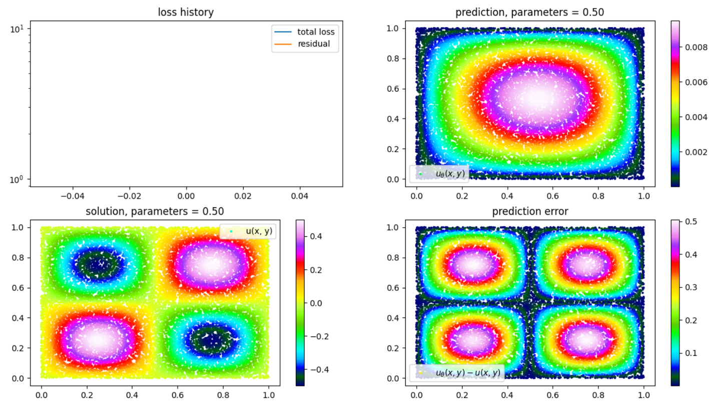




5.2 Generating visuals using ScimBa

```
    xdomain = domain.SpaceDomain(2, domain.SquareDomain(2, [[0.0, 1.0],
                                                           [0.0, 1.0]]))
    print(xdomain)
    pde = Poisson_2D(xdomain)
    Run_laplacian2D(pde)
```

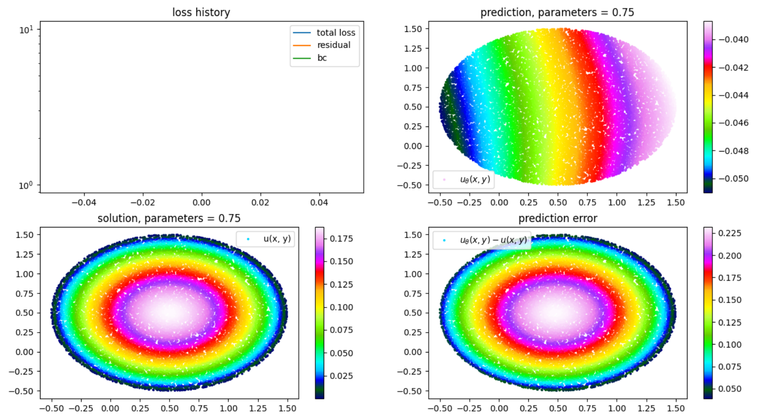
Laplacian strong Bc on Square with nn .



Laplacian on circle with nn

```
    xdomain = domain.SpaceDomain(2, domain.DiskBasedDomain(2, [0.5, 0.5], 1.0))
    pde = PoissonDisk2D(xdomain)
    Run_laplacian2D(pde, bc_loss_bool=True, w_bc=10, w_res=0.1)
```

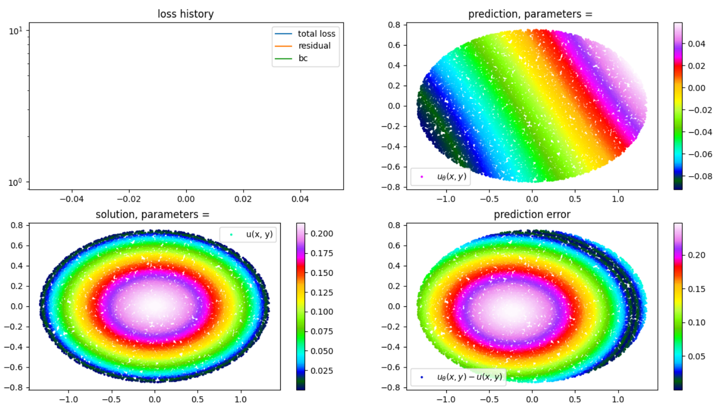
.



Laplacian on ellipse and mapping with nn

```
    xdomain = domain.SpaceDomain(
        2,
        domain.DiskBasedDomain(
            2,
            [0.0, 0.0],
            1.0,
            mapping=disk_to_ellipse,
            Jacobian=Jacobian_disk_to_ellipse,
        ),
    )
    pde = Poisson_2D_ellipse(xdomain)
    Run_laplacian2D(pde, bc_loss_bool=True, w_bc=10, w_res=0.1)
```

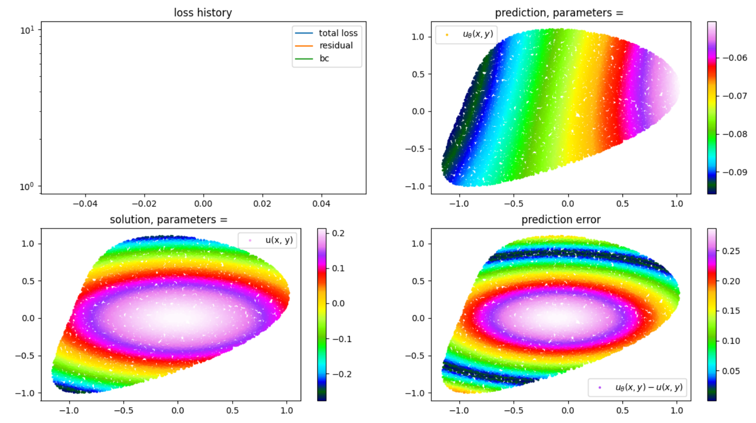
.



Laplacian on potato and mapping with nn

```
    xdomain = domain.SpaceDomain(2, domain.SquareDomain(2, [[0.0, 1.0],
                                                            [0.0, 1.0]]))
    print(xdomain)
    pde = Poisson_2D(xdomain)
    Run_laplacian2D(pde)
```

.


loss history
total loss
residual
bc
10^1
10^0
−0.04
−0.02
0.00
0.02
0.04
prediction, parameters =
$u_\theta(x, y)$
solution, parameters =
u(x, y)
prediction error
$u_\theta(x, y) - u(x, y)$

6 Conclusion

Bibliography

- Feel++ Documentation: `https://docs.feelpp.org/user/latest/index.html`
- ScimBa Documentation: `https://sciml.gitlabpages.inria.fr/scimba/`
- Coupling : `https://en.wikipedia.org/wiki/Coupling_(computer_programming)`
- Using feel++: `https://www.cemosis.fr/events/course-solving-pdes-with-feel/`
- Feel++ documentation: `https://book.mso4sc.cemosis.fr/feelpp`