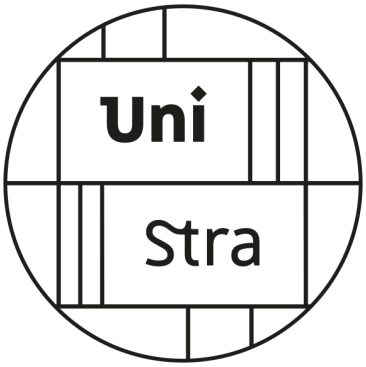


Project report: Coupling ScimBa and Feel++

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 General objectives

The primary objective of this project is to pioneer the integration of ScimBa and Feel++, two robust libraries renowned for their distinct strengths. The specific objectives are:

1. **Streamlined Data Exchange**: Develop a system that streamlines data exchange between ScimBa and Feel++, enabling seamless interaction between the two libraries.
2. **User Empowerment**: Create an interface that allows users to leverage the combined strengths of ScimBa and Feel++ effectively, thereby enhancing their computational research capabilities.
3. **Integration of Technologies**: Integrate various technologies such as Docker, Python, and Git to create a reproducible environment for the project, apply machine learning techniques, solve PDEs, and manage source code.
4. **Advancement of Computational Research**: Drive advancements in computational research by creating a platform that unifies machine learning techniques and PDE solving methods, thereby opening new avenues for innovation and discovery.

3 Project Status

This project encompasses a comprehensive overview meticulously detailed within each section, delineating the evolution of our efforts and elucidating the advancements made therein:

- Section V0
  - Create a Docker container
  - Make the presentation
  - Make the roadmap
  - Bibliography
  - Write initial report
- Section V1
  - Update Project Roadmap and Milestones
  - Refine Project Scope and Deliverables
  - Install git in docker image
  - Get familiar with scimba
  - Clone Scimba in docker image
  - Methodology Update
  - Update Project Objectives
  - First Results Analysis
  - Conduct Technical Review and Code Quality Check
- Section V2

  (The project is currently in the V1 phase.)

4 Section V0

4.1 Roadmap

Figure 1: Roadmap for V0

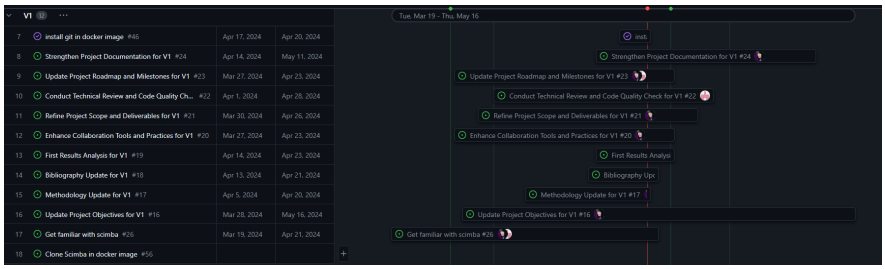


Figure 2: Roadmap for V1

Figure 3: Roadmap for V2

4.2 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

4.3 Feel++

Feel++ is a library that allows manipulation of mathematical objects to solve Partial Differential Equations (PDEs). It also provides 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
- Levelset and multifluid

4.4 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: .

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**:

4.5 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, such as Feel++, offer features for handling different PDEs. They make it easier to define coefficients, set boundaries, discretize problems, and use numerical methods. This helps users to solve complex PDEs, study physical phenomena, and simulate real-world situations more efficiently.

2. **System of PDEs**: Many PDEs can be expressed in a standard form, mainly based on the coefficients' definition. We use the following equation to find this form:
$u : \Omega \subset \mathbb{R}^d \longrightarrow \mathbb{R}^n$ 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 also need to follow certain limitations on coefficient shapes, as detailed in the table below.

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 Initial conditions set the initial values for each unknown variable in the equations. These conditions can be defined using expressions or fields.

   Boundary Conditions:

   - Dirichlet
   - Neumann
   - Robin

5. **Finite Element Approximation**
6. **Time scheme**:
   - Backward Differences Formula
   - Theta scheme
7. **Stabilized finite element methods**
   - GaLS
   - SUPG
   - Coefficient form PDE toolbox

5 Section V1

5.1 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.

5. **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.
6. **Optimize Performance**: Continually optimize the performance of the program, ensuring that it runs efficiently and effectively on various hardware configurations.
7. **User-Friendly Interface**: Develop a user-friendly interface that allows users with varying levels of technical expertise to utilize the program effectively.
8. **Documentation and Training**: Provide comprehensive documentation and training materials to help users understand how to use the program and interpret the results.
9. **Community Engagement**: Engage with the user community to gather feedback, identify areas for improvement, and guide future development efforts.

5.2 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.

```
1
2  # Start with the Feel++ base image
3  FROM ghcr.io/feelpp/feelpp:jammy
4
5  # Set labels for metadata
6  LABEL maintainer="Helya Amiri <helya.amiri@etu.unistra.fr>,
7                    Rayen Tlili <rayen.tlili@etu.unistra.fr>"
8  LABEL description="Docker image with Feel++, Scimba, and Python libraries."
9
10 USER root
11
12 # Install system dependencies
13 RUN apt-get update && apt-get install -y \
14     git \
15    xvfb
16
17 # Install Python libraries
18 RUN pip3 install torch xvfbwrapper pyvista plotly panel
19
20
21 # Clone the Scimba repository
22 RUN git clone https://gitlab.inria.fr/scimba/scimba.git /scimba
23
24 # Install Scimba and its dependencies
25 WORKDIR /scimba
26 RUN pip3 install .
27
28 # Copy the xvfb script into the container
29 COPY tools/load_xvfb.sh /usr/local/bin/load_xvfb.sh
30 RUN chmod +x /usr/local/bin/load_xvfb.sh
31
32 # Set the script to initialize the environment
33 CMD ["/usr/local/bin/load_xvfb.sh", "python3"]*
```

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

5.3 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')
```

5.4 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

6 Section V2

6.1 Future Work

The next phase of our project, referred to as V2, has several open issues that need to be addressed:

- Automatize the image generating issue in the docker (Issue #52)
- Solve in ScimBa (Issue #51)
- Return mesh and dofs in __call__ (Issue #6)
- Code Cleanup and Documentation (Issue #7)
- Advisor Code Review (Issue #9)
- Prepare Presentation Slides and Demo (Issue #11)
- Conduct Project Retrospective (Issue #12)
- Develop and Finalize the Project Presentation (Issue #13)
- Consolidate and Finalize the Project Bibliography (Issue #14)
- Compose the Comprehensive Final Project Report (Issue #10)
- Gather Feedback (Issue #8)
- Finalize README and Documentation (Issue #8)

These issues represent the key tasks that will be undertaken in the V2 phase of the project. Each issue is linked to its corresponding GitHub issue for easy tracking and reference.

7 Results

7.1 Generating visuals using Feel++

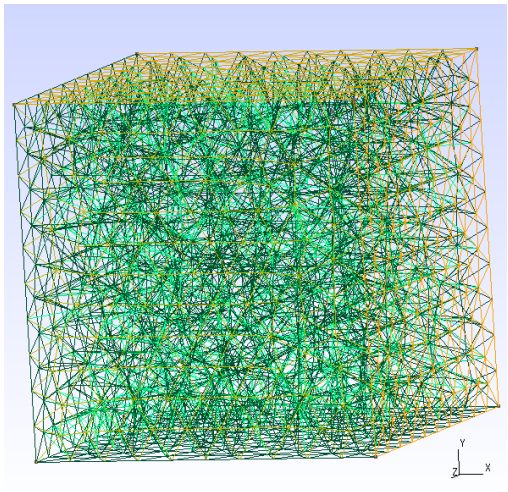
Using the Poisson class and the Laplacian2D.py program,

```
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
```

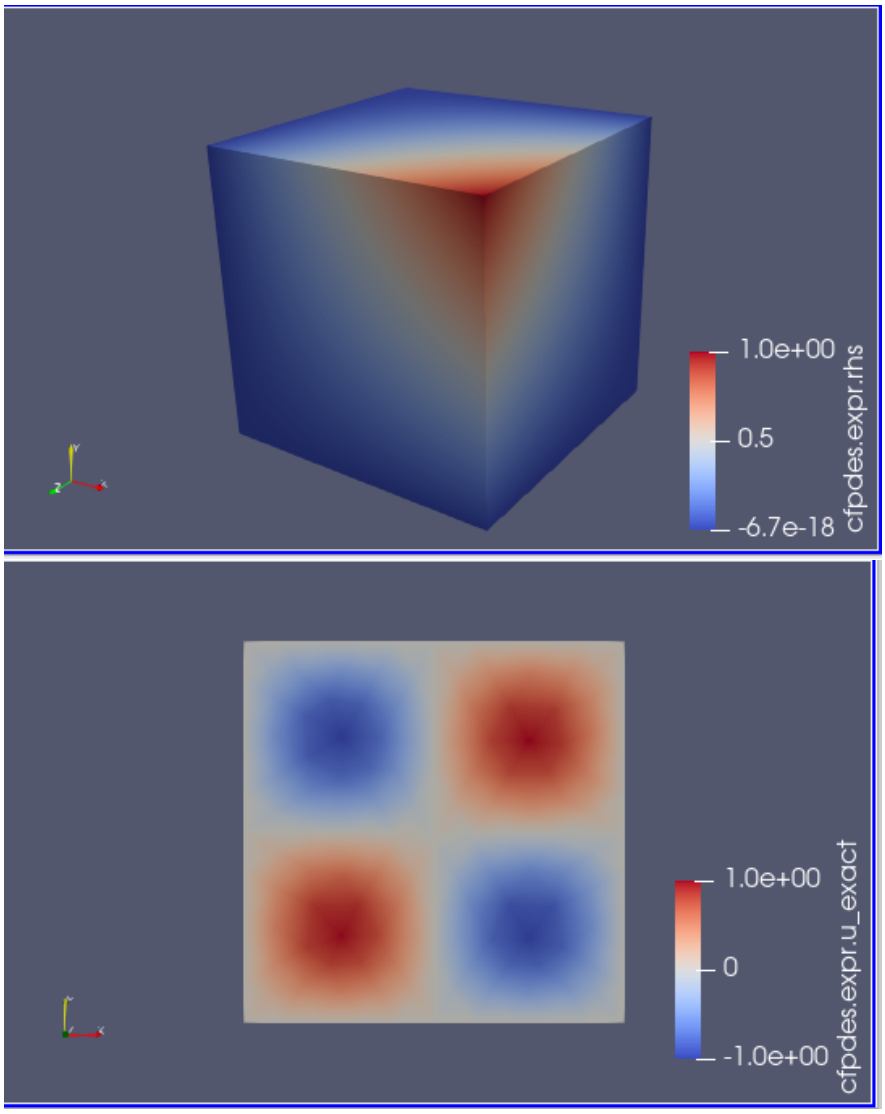
Generated 3D geometry and mesh viewed using gmsh:



.

Viewing 3D and 2D plotting viewed using paraview:

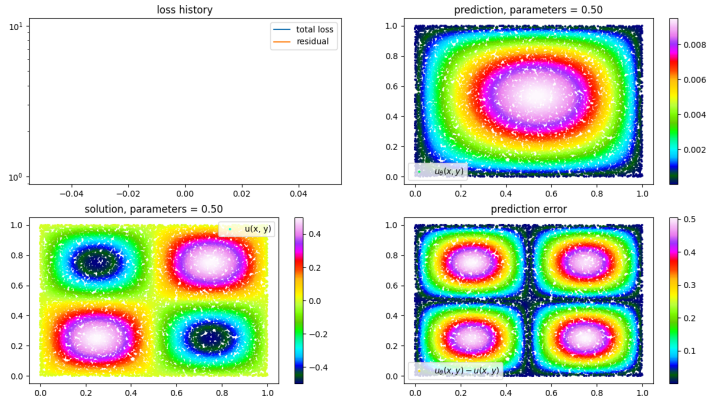
.



7.2 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)
```

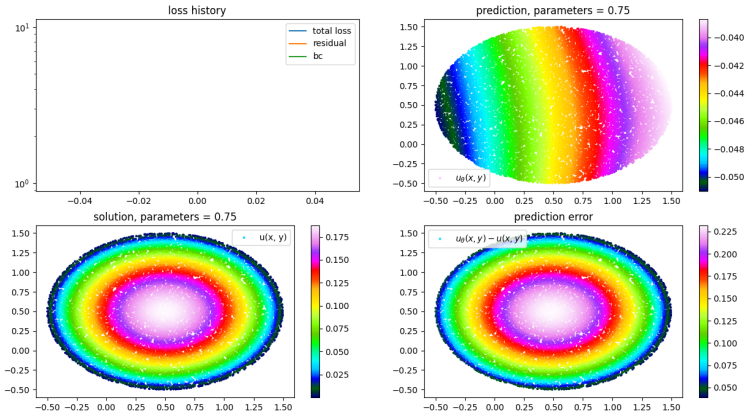
Laplacian strong Bc on Square with nn .



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)
```

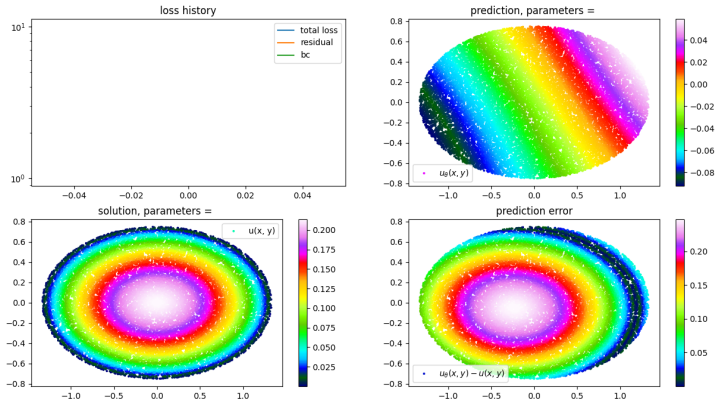
.



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)
```

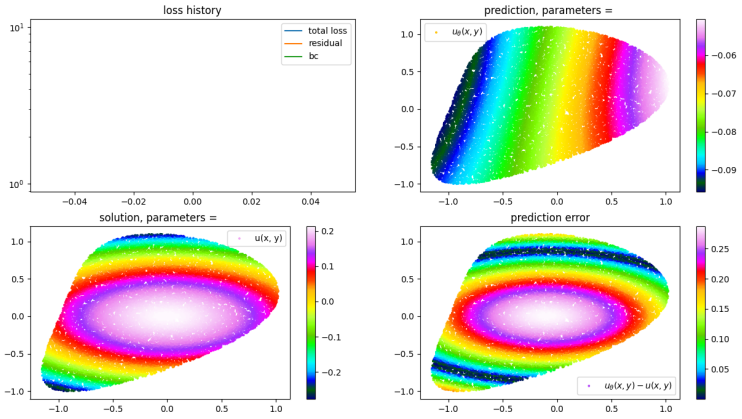
.



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
prediction, parameters =
$u_\theta(x, y)$
solution, parameters =
u(x, y)
prediction error
$u_\theta(x, y) - u(x, y)$

8 Conclusion

Bibliography

[1] Feel++. (n.d.). *Modeling and Quantitative Simulation*. Retrieved from https://feelpp.github.io/mqs/cases/0.109/index.html

[2] Feel++. (n.d.). *Python Feel++ Toolboxes*. Retrieved from https://docs.feelpp.org/user/latest/python/pyfeelpptoolboxes/index.html

[3] MSO4SC. (n.d.). *Poisson*. Retrieved from https://book.mso4sc.cemosis.fr/cases/0.109/poisson/README/

[4] Zenodo. (n.d.). *Zenodo Record*. Retrieved from https://zenodo.org/records/1231527

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

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

[7] Feel++. (n.d.). *Feel++ GitHub Repository*. Retrieved from https://github.com/feelpp/feelpp

[8] Cemosis. (n.d.). *Course: Solving PDEs with Feel++*. Retrieved from https://www.cemosis.fr/events/course-solving-pdes-with-feel/

[9] MSO4SC. (n.d.). *Feel++ Book*. Retrieved from https://book.mso4sc.cemosis.fr/feelpp/

[10] Wikipedia. (n.d.). *Coupling (computer programming)*. Retrieved from https://en.wikipedia.org/wiki/Coupling_(computer_programming)

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

[12] Feel++. (n.d.). *Feel++ Documentation*. Retrieved from https://docs.feelpp.org/user/latest/index.html

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

[14] Feel++. (n.d.). *Mixed Poisson*. Retrieved from https://docs.feelpp.org/user/latest/using/toolboxes/hdg_poisson.html