UNIVERSITÉ DE STRASBOURG

Wrapping ScimBa and Feel++

Helya Amiri **Rayen Tlili**

June 4, 2024

Main objective:
Building an intermediary to ScimBa in Feel++

UNIVERSITÉ DE STRASBOURG

The main objective
Introduction

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

UNIVERSITÉ DE STRASBOURG

The main objectives

Introduction

- Create multiple results using Feel++ toolboxes.
- Using ScimBa to understand and share results
- Creating a program that can use both as solvers.
- Comparing the Results of Both Solvers
- Expand Application Scope

UNIVERSITÉ DE STRASBOURG

Roadmap

Introduction

1. Explore Feel++ and ScimBa documentation.
2. Create a container using docker with a Feel++ base and install ScimBa within it.
3. Solve PDEs using Feel++.
4. Solve PDEs using ScimBa PINNs.
5. Create a Poisson class you can call to solve using Feel++.
6. Add ScimBa as a solver for the class by updating the Poisson2d class to handle ScimBa with parametrized f, g and add a diffusion tensor to the Poisson2d class.
7. Compare the results of both solvers with exact solutions.
8. Compute L^2 and H^1 errors and trace their convergence for both solvers.

Introduction to Feel++

UNIVERSITÉ DE STRASBOURG

Getting familiar with Feel++

Feel++

- Library for solving PDEs
- Toolboxes for math and physics-based problems
- Coefficient Form PDEs toolbox (CFPDE)

Exploring Feel++ Toolboxes

Feel++

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

CFPDE

Feel++

The Coefficient Form PDEs toolbox:

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

- 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

Introduction to ScimBa

Getting familiar with ScimBa

ScimBa:

- Python library
- Merges machine learning with scientific computing
- Varying SciML (Scientific Machine Learning) methods for varying PDE problem
- Tools to build hybrid numerical methods

Getting familiar with ScimBa (PINNs)

We began utilizing examples from the ScimBa repository that employ Physics-Informed Neural Networks (PINNs)[1]

Reference: M. 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.

Using ScimBa to solve a Laplacian problem in 2D

ScimBa

Solving the Poisson equation on a unit square domain:

```
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 Poisson2D class

ScimBa

The parameter domain is carefully defined, to enforce specific boundary conditions or ensure solution stability.

```
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 Runlaplacian2D function

ScimBa

The Run laplacian2D covers data sampling, network setup, loss calculation, and optimization.

```python

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)
```

Training

ScimBa

- If `new training = False`, it 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.

```python

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)
```

Setting up the Container

Why use Docker?

Docker

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

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

Creating a Docker container and image

Docker

Creating the Docker container

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

# 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
```

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-m1-scimba-feelpp/scimba

# Install Scimba and its dependencies
WORKDIR /workspaces/2024-m1-scimba-feelpp/scimba
RUN pip3 install scimba

# 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"]
```

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

Container limitations

Docker

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

Methodology

Setting the environment
Github

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

Launch

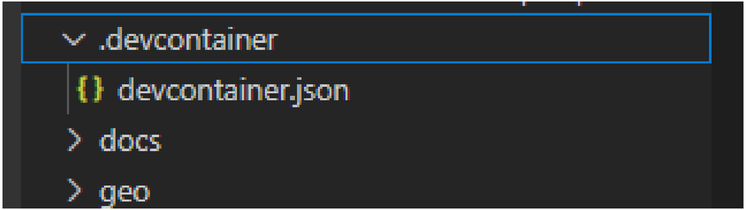
Follow these steps to get the project up and running on your local machine:

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
Github

Setting the container:



Setting the environment

Github

Inside the '.devcontainer' folder:

```
{
"name": "ScimBa-Feel++ 22.04",
"image": "feelpp_scimba:latest",
// Add the IDs of extensions we want installed
"extensions": [
    "ms-vscode.cpptools",
    "ms-vscode.cmake-tools",
    "josetr.cmake-language-support-vscode",
    "asciidoctor.asciidoctor-vscode",
    "ms-python.python",
    "ms-toolsai.jupyter"
]
}
```

Setting the environment

Feel++

Create the right environment for using the CFPDE toolbox:

```
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'))
```

The Poisson class

Feel++

Inside that environment we want to call upon a Poisson class to solve 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')
```

Calling the class

Feel++ ++ ScimBa

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='scimba'):                     # or solver='feelpp'
    """
```

Calling the class

++Fe+el++ ++++ S++ci+m+++Ba

Solving using Feel++ and ScimBa:

```python
P( rhs='-1.0-4*y*x+y*y', g='x', order=1, solver='feelpp')
P( rhs='-1.0-4*y*x+y*y', g='x', order=1, solver ='scimba')
```

Calling the class

Solving using Feel++ and ScimBa and comparing with an exact solution:

```
P( rhs='-1-4*y*x+y*y', g='x', order=1, solver='feelpp', u_exact= u_exact)
P( rhs='-1-4*y*x+y*y', g='x', order=1, solver ='scimba', u_exact= u_exact)
```

Results

Generating visuals using Feel++

Feel++

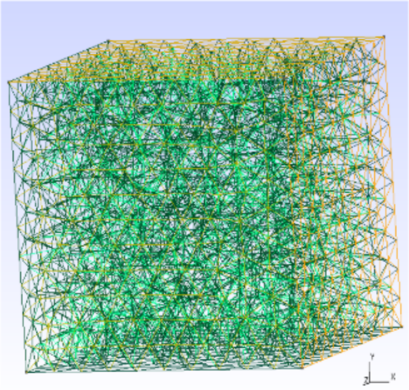
Feel++ generates geometry files for either a 2D rectangle or a 3D box, compatible with Gmsh.

```python
def getMesh(filename, hsize=0.05, dim=2, verbose=False):
"""create mesh
Args:
    filename (str): name of the file
    hsize (float): mesh size
    dim (int): dimension of the mesh
    verbose (bool): verbose mode"""
                .
                .
                .
generateGeometry(filename=filename, dim=dim, hsize=hsize)
mesh = feelpp.load(feelpp.mesh(dim=dim, realdim=dim), filename, hsize)
return mesh
```

Generating visuals using Feel++

Feel++

Generated 3D geometry and mesh viewed using gmsh:

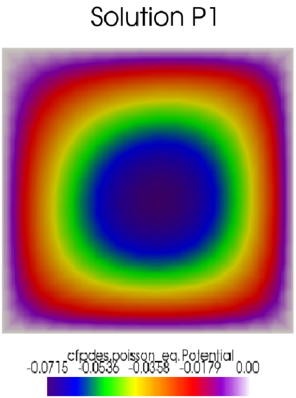


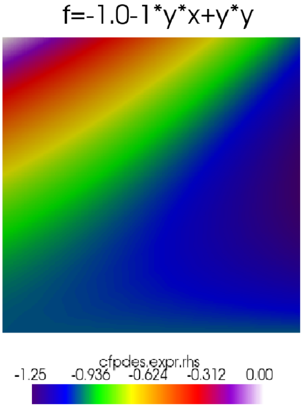
Generating visuals using Feel++

Feel++

We initiate the Poisson class instance P by specifying the dimension as 2:

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



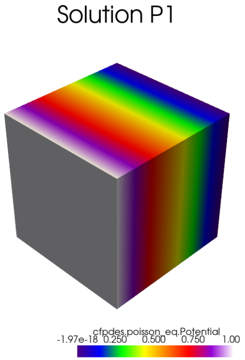


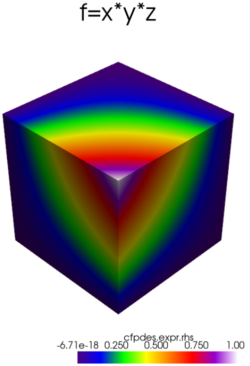
Generating visuals using Feel++

Feel++

We initiate the Poisson class instance P by specifying the dimension as 3:

```
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')
```





Generating visuals using ScimBa

We start by defining the spatial domain `xdomain` with ScimBa's SpaceDomain module, setting a two-dimensional square domain from $(0.0, 0.0)$ to $(1.0, 1.0)$.

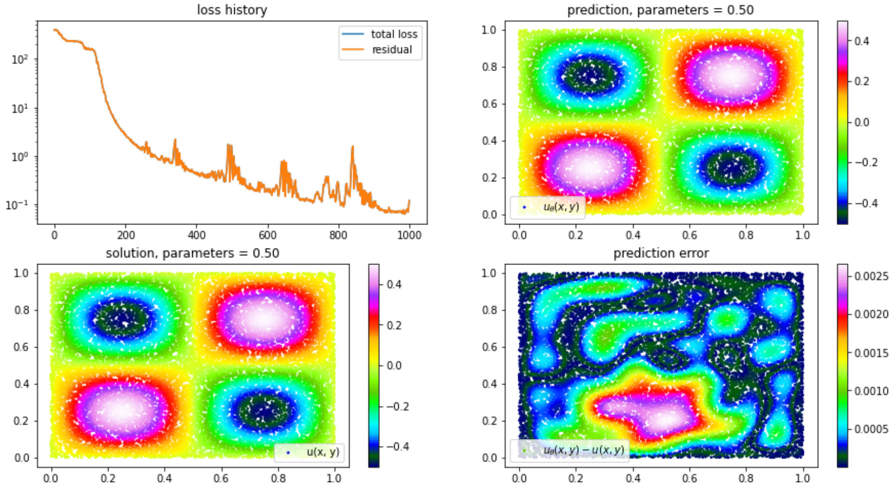
Lastly, we run the `Run_laplacian2D` function to solve the Poisson equation defined by `pde`.

```
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

ScimBa

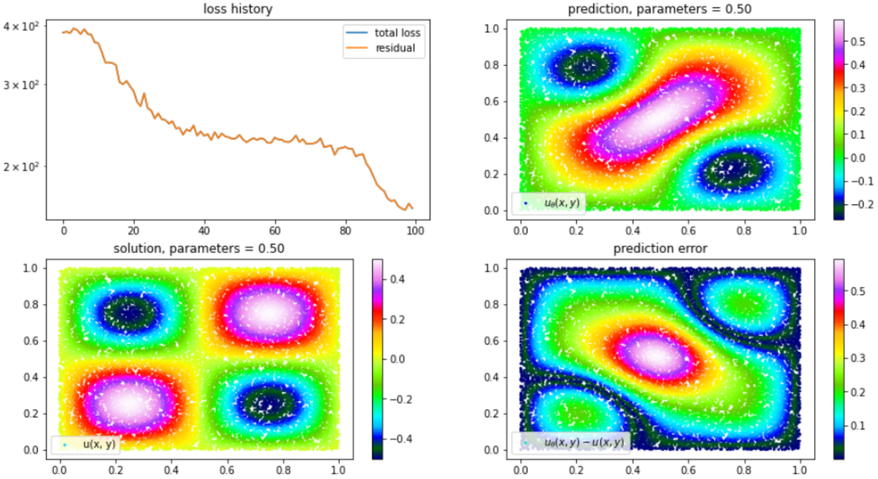
The code snippet above produces the following visual representation:



Generating visuals using ScimBa

ScimBa

The same problem with 100 epochs of training:



Laplacian on disk mapping

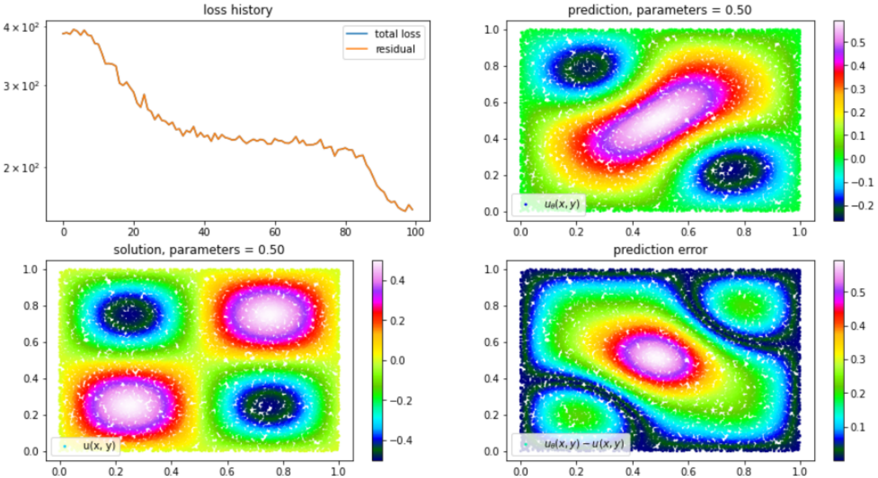
In this instance, we specify a two-dimensional domain utilizing a disk-based configuration with a center at $(0.0, 0.0)$ and a radius of 1.0.

```
xdomain = domain.SpaceDomain(2, domain.DiskBasedDomain(
                             2, center=[0.0, 0.0], radius=1.0))
    u_exact = ' (1 - x*x - y*y)'
    rhs = '4'
    pde_disk = PoissonDisk2D(xdomain, rhs= rhs, g= '0', u_exact=u_exact)
    Run_laplacian2D(pde_disk)
```

Generating visuals using ScimBa

ScimBa

The code snippet above produces the following visual representation:



UNIVERSITÉ DE STRASBOURG

Comparing the visuals for a Laplacian problem

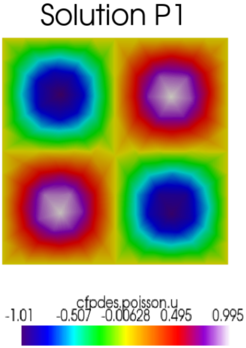
This segment focuses on visualizing the solutions to the Laplacian problem on a square domain. We compare the numerical accuracy and visual fidelity of the solutions using both Feel++ and Scimba solvers.

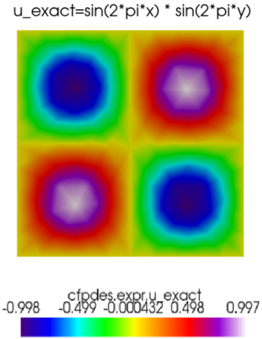
```
# 2D on different domains
P = Poisson(dim = 2)

# 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)
```

Comparing the visuals for a Laplacian problem

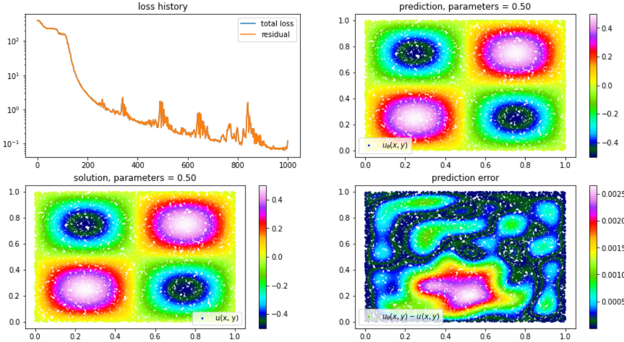
++++++++++++

Feel++





ScimBa



Comparing the visuals for a Laplacian problem

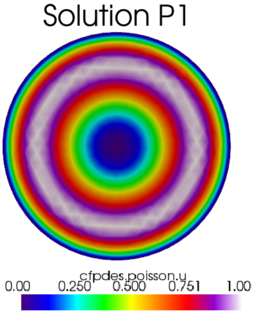
Testing the solvers' capabilities in more complex geometrical contexts.

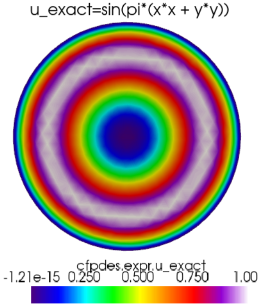
```python

# for disk domain
u_exact = 'sin(pi*(x*x + y*y))'
rhs = '4*pi*sin(pi*(x*x + y*y)) - 4*pi*pi*(x*x + y*y)*cos(pi*(x*x + y*y))'

P(rhs=rhs, g='0', order=1, geofile='geo/disk.geo', plot='2d.png',
    u_exact = u_exact)
P(rhs=rhs, g='0', order=1, geofile='geo/disk.geo', solver='scimba',
    u_exact = u_exact)
```

Comparing the visuals for a Laplacian problem

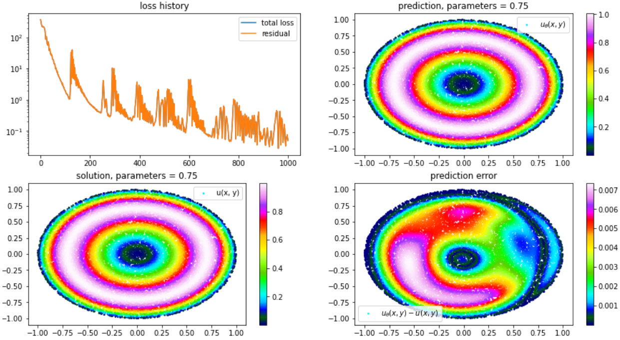
++++++++++++

Feel++





ScimBa



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
```

Computing L2 and H1 errors (Computing the errors)

++++++++++++

This function iterates over a set of mesh sizes h, computes the solution using a specified computational model, and appends the L2 and H1 errors to the dataframe.

```
df= runLaplacianPk(P, df=df, model=model, verbose=True)
```

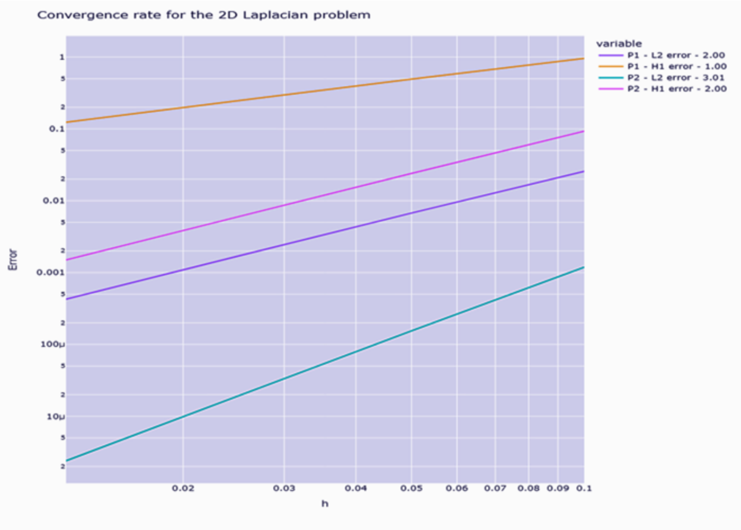
Plotting the convergence rate

We conduct the convergence analysis for various mesh sizes and polynomial orders. We generate and display a plot of convergence rates across mesh sizes for each polynomial order.

```

df=runConvergenceAnalysis(json=laplacian_json ,dim=2,verbose=True)
fig= plot_convergence(P, df ,dim=2)
fig.show()
```

Tracing the convergence rate

++++++++++++



Bibliography (Part 1)

o

[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)

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`

Conclusion

++++++++++++

Wrapping Feel++ with ScimBa meaningfully is a challenging task but the project was successful in certain areas yet there are still some current setbacks and potential for future work.

Thank you for listening!

Any questions?