

Project report: Coupling ScimBa and Feel++

Helya Amiri^{*} & Rayen Tlili[†]

Supervised by Christophe Prud'homme, Joubine Aghili

Submitted April 22, 2024

^{*}: , `helya.amiri@etu.unistra.fr`

[†]: , `rayen.tlili@etu.unistra.fr`

Contents

<i>Abstract</i>	ii
Main Content	1
1 Introduction	1
2 Objectives	1
3 Roadmap	2
4 Getting started	3
4.1 Creating a Docker container	3
4.2 Exploring Feel++ toolboxes	4
4.2.1 Coefficient Form Toolbox	4
5 V1	7
5.1 Objectives for V1	7
5.2 updating the Docker container	8
5.3 Methodology	9
5.4 Getting familiar with ScimBa	11
6 Results	12
7 Conclusion	13
<i>Bibliography</i>	14

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 Roadmap



Figure 1: Roadmap for V0

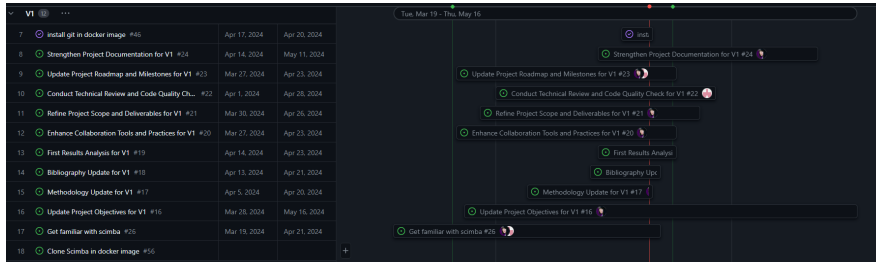


Figure 2: Roadmap for V1



Figure 3: Roadmap for V2

4 Getting started

4.1 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:

```
1 # Start with the Feel++ base image
2 FROM ghcr.io/feelpp/feelpp:jammy
3
4 # Set labels for metadata
5 LABEL maintainer="Helya Amiri <helya.amiri@etu.unistra.fr>,"
6         Rayen Tlili <rayen.tlili@etu.unistra.fr>"
7 LABEL description="Docker image with Feel++, Scimba, and PyTorch."
8
9 # Install PyTorch.
10 RUN pip3 install torch
11
12 # Copy the local Scimba directory to the container.
13 COPY scimba/ /scimba/
14
15 # Install Scimba and its dependencies
16 WORKDIR /scimba
17 USER root
18
19 RUN pip3 install .
20
21 # Set the default command to launch Python.
22 CMD ["python3"]
```

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

4.2 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 available 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**:

4.2.1 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

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

- 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, \dots, 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

5 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.

5.2 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.toolbox.core as tb

from tools.solvers import Poisson
sys.argv = ["feelpp_app"]
e = feelpp.Environment(sys.argv,
                       opts=tb.toolbox_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 extracted 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

6 Results

7 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_\(computer_programming\)](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>