Project 01 – Open FEM Project with FEniCSx

Exercise Sheet

Dr. Lambert Theisen & Dr. Georgii Oblapenko

0.1 Overview

In this mini-project you will design, implement, verify, and analyze a finite element model for a **physical problem or PDE system of your own choice** using the Python/FEniCSx stack. The final submission is a **research-style minipaper** of around **6 pages** (excluding figures) together with a **Git repository** containing code, meshes, tests, project documentation, and all metadata needed to reproduce the results.

> **!** Important
>
> **Deadline (end of that date):** 01.06.2026
> **Submission:** Via Moodle as repository link and PDF. Add us as members (`@lambert.theisen` / `@georgii.oblapenko`) to the project repository.

> **i** Note
>
> **Own initiative is part of the task.** You should choose a physical setup that interests you, formulate the PDE model, and decide what numerical and physical questions are worth studying. Discuss your idea with us early if you are unsure whether the scope is realistic.

The project should combine:

- a clearly motivated physical problem and PDE model,
- derivation of the weak form including boundary conditions,
- a FEniCSx implementation with boundary conditions handled generically through tags or reusable functions,
- verification by a manufactured solution **or** comparison with another method/reference solution where appropriate,
- a mesh convergence study with error norms and convergence rates,
- a realistic geometry or scenario study comparing meshes, geometries, boundary conditions, or material/physical parameters,
- analysis of at least one physical quantity of interest,
- clear documentation of solver choices, software versions, testing, and reproducibility.

> **!** Important
>
> **Software-engineering requirement.** Treat this as a small research software project, not only as a report. Your work should live in a **Git repository** and be reproducible from the command line. Include code, a short project documentation, and lightweight but meaningful automated checks. CI/CD pipelines shall perform the automated testing and hosting of the project documentation,

for example via GitLab Pages.

Note

Scientific focus. Derivation, verification, comparison, interpretation, and reproducibility matter at least as much as the final visualization.

0.1.1 Learning Outcomes

After completing this project, you should be able to:

- select and justify a PDE model for a physical problem,
- derive and interpret the weak form including boundary terms,
- implement a finite element discretization in FEniCSx,
- verify a PDE solver by mesh refinement, error norms, and convergence rates,
- compare your FEM result with a manufactured solution, another method, or a documented reference where applicable,
- generate or import realistic meshes programmatically,
- analyze how geometry, mesh resolution, boundary conditions, or physical parameters affect meaningful quantities of interest.

0.1.2 Topic Selection

Choose a physical problem/PDE system that is interesting to you and feasible within the project time. Suitable examples include, but are not limited to:

- heat conduction or diffusion,
- nonlinear PDEs such as nonlinear diffusion, nonlinear elasticity, or reaction-diffusion models,
- linear elasticity,
- Stokes or incompressible flow,
- advection-diffusion or reaction-diffusion,
- eigenvalue problems,
- electrostatics or potential flow,
- coupled or parameter-dependent variants of the above, if the scope remains manageable.

Your project may be stationary or time-dependent. If you choose a time-dependent problem, the spatial FEM part must still be central, and the verification/convergence study must remain manageable.

Tip

Need ideas? Start from the official DOLFINx Python demos, Dokken's DOLFINx tutorial, the lectures, or your own literature research. Use external examples as learning material, but your submitted setup, implementation choices, discussion, and report must be your own work.

0.1.3 Requirements for the Project and the Minipaper

Other than your lecture notes, consult online resources, papers, textbooks, or software documentation related to your chosen topic. Reference all sources you use. Everyone needs to hand in their own

repository containing the minipaper. The minipaper must include a written statement that the work has been done individually. The statement may for example read:

> I hereby declare that I wrote this report on my own and without the use of any other than the cited sources and tools and all explanations that I copied directly or in their sense are marked as such.

Structure the report like a short paper, including, for example:

- introduction and physical setup,
- model and weak formulation,
- implementation and solver choices,
- verification or benchmark comparison,
- mesh convergence study,
- realistic scenario study,
- conclusion.

0.1.3.1 Submission Guidelines

- Submit one PDF report plus the Git repository used to generate the results.
- Submit via Moodle as Link to repo and PDF.
- The repository should have a clean, understandable structure and meaningful file names.
- Use Git throughout the project instead of collecting everything at the end in one folder.
- Label all plots and tables clearly.
- Use consistent notation and include units where appropriate.
- Keep the workflow reproducible and scriptable from start to finish.
- Avoid undocumented manual steps in meshing, postprocessing, or figure generation (or at least document them clearly in the project documentation).
- You may discuss ideas with classmates, but all submitted text, code, plots, and conclusions must be your own.

0.2 Mathematical Model and Weak Form

State your chosen strong-form PDE or PDE system on a domain $\Omega \subset \mathbb{R}^d$, $d = 1, 2$, or 3, and define all unknown fields, material parameters, source terms, and units.

The following scalar second-order equation is only a small illustrative example of the kind of notation one might write down:

$$-\nabla \cdot (A\nabla u) + b \cdot \nabla u + cu = f \quad \text{in } \Omega.$$

Do not treat this as the default project model. Your own project may look quite different: it may be nonlinear, vector-valued, mixed, time-dependent, an eigenvalue problem, or a coupled system.

Describe the boundary decomposition relevant to your model, for example

$$\partial\Omega = \Gamma_D \cup \Gamma_N \cup \Gamma_R \cup \Gamma_{\text{other}},$$

where the subsets are disjoint and may be empty. Explain the physical meaning of every boundary part and every boundary condition.

Typical boundary-condition types include:

$$u = u_D \quad \text{on } \Gamma_D,$$

$$(A\nabla u) \cdot n = g_N \quad \text{on } \Gamma_N,$$

$$(A\nabla u) \cdot n + h(u - u_\infty) = 0 \quad \text{on } \Gamma_R.$$

Adapt signs and notation to your own PDE. For systems such as Stokes or elasticity, formulate the corresponding vector or mixed boundary conditions instead.

In your report, derive the weak form by multiplying with test functions, integrating by parts where appropriate, and explaining:

1. which function spaces contain the trial and test functions,
2. how essential boundary conditions enter the space or are imposed algebraically,
3. which natural boundary conditions appear in the linear form,
4. which Robin, reaction, coupling, or stabilization terms appear in the bilinear or nonlinear residual form,
5. what finite element spaces you use, for example continuous Lagrange elements, Taylor-Hood elements, or another justified choice.

0.3 Required Software Stack

Use the same implementation stack as Lecture 03:

- Python with the `fenicsx` environment/kernel,
- `dolfinx`, `ufl`, `basix`, `mpi4py`, `petsc4py`,
- `dolfinx.fem.petsc.LinearProblem`, `NonlinearProblem`, explicit PETSc assembly, SLEPc, or another appropriate PETSc-backed workflow,
- `gmsh` and `dolfinx.io.gmshio.model_to_mesh` or `dolfinx.io.gmsh.read_from_msh` for generated geometries where applicable,
- `numpy` and `matplotlib` for convergence data and visualization.

ParaView may also be used for visualization if applicable. If you use ParaView, write XDMF/VTK output from your scripts and document the exact plotting workflow well enough that the figures are reproducible.

For reproducibility, your repository must document the environment and include commands that can be run from a clean checkout. A good minimal check is:

```
python -c "import dolfinx, gmsh, petsc4py, mpi4py; print('FEniCSx stack ok')"
```

Document the installation and execution instructions in the `README.md`.

> **Note**
>
> Use Python code and FEniCSx abstractions for the submitted FEM solver. Implementations from Project 00 or other methods may be useful as comparison baselines, but they are not the target stack for the main solver.

0.4 Exercise 1: Problem Choice, Model, and Weak Form

1. Choose a physical problem or PDE system and explain why it is interesting.
2. State the PDE, unknowns, material parameters, source terms, and physical assumptions.
3. Define the boundary decomposition and explain the role of each boundary subset.
4. Derive the weak form from the strong form, including boundary terms.
5. State the finite element spaces and polynomial orders you use.
6. Explain any simplifications, nondimensionalization, or assumptions needed to keep the project feasible.

Your write-up should be specific enough that another student could identify exactly which mathematical problem you solved.

0.5 Exercise 2: FEniCSx Implementation and Verification

Implement your chosen PDE first on a simple domain where verification is possible, such as an interval, unit square, rectangle, unit cube, or another geometry with a known solution.

1. Implement a reusable FEniCSx solver for your chosen PDE.
2. Use the Lecture 03 pattern: build function spaces, define UFL forms/residuals, impose boundary conditions, and solve with PETSc-backed linear algebra.
3. Accept geometry, material data, source terms, and boundary data in a way that lets you change test cases without rewriting the full solver.
4. Represent boundary subsets by facet tags whenever tags are available.
5. Verify the implementation by at least one of the following:
    - **manufactured solution:** choose an exact solution and derive matching source terms and boundary data,
    - **comparison with another method:** for example finite differences on a structured-grid problem if applicable,
    - **comparison with a trusted reference:** for example an analytical solution, benchmark value, or carefully documented literature result.
    - [**mesh convergence study:** show that the solver converges under mesh refinement]
6. Explain why your chosen verification method is appropriate for your PDE.
7. Include plots or tables that compare numerical and reference results.

0.6 Exercise 3: Mesh Convergence Study

Perform a mesh convergence study for the verification problem.

1. Use at least four mesh resolutions.
2. Define the mesh-size measure h you use.
3. Compute at least one error norm. Use more than one where meaningful, for example L^2, H^1 seminorm, energy norm, maximum norm, divergence error, eigenvalue error, or an application-specific quantity.
4. Plot error versus h on a log-log scale.
5. Estimate experimental convergence rates.
6. Compare the observed rates with theory or with a justified expectation for your element choice and PDE.
7. If exact error norms are not available, compare against a sufficiently refined reference solution and state the limitations.

8. If no theoretical rates are known for your problem, study the changes between consecutive mesh refinements. For example, compare quantities of interest or solution differences from mesh to mesh and explain whether the sequence appears to stabilize.

For many smooth scalar elliptic problems with P1 elements, one often expects approximately

$$\|u - u_h\|_{L^2} = \mathcal{O}(h^2), \qquad \|u - u_h\|_{H^1} = \mathcal{O}(h).$$

These rates are not universal. State the expected behavior for your own PDE, element, regularity, and boundary conditions.

0.7 Exercise 4: Realistic Scenario Study

Create a more realistic physical scenario for your chosen problem and analyze it with your FEM solver.

0.7.1 Reproducibility Requirement

Your geometry, mesh, boundary tags, solver settings, and postprocessing must be reproducible. Do not rely on manual GUI-only meshing steps. Use one of the following:

- the Gmsh Python API with `gmsh.model.occ`,
- a scripted import of an `.geo` geometry,
- a documented mesh-generation script from another tool,
- a simple programmatically generated FEniCSx mesh if that is sufficient for your physical setup.

Import external meshes into FEniCSx in a reproducible way, for example via `dolfinx.io.gmshio.model_to_mesh` or `dolfinx.io.gmsh.read_from_msh`. Represent boundary conditions through facet tags or another documented tagging mechanism so the same solver can be reused across variants.

0.7.2 Required Study

1. Compare at least **two variants** under controlled conditions. Variants may be different meshes, geometries, boundary-condition choices, material parameters, source terms, or physical configurations.
2. Define at least **one quantitative physical quantity of interest** before running the comparison.
3. Suitable quantities include maximum/minimum field value, average value in a region, total flux, lift/drag-like integral, displacement, stress concentration, pressure drop, eigenfrequency, decay rate, reaction yield, or another quantity appropriate to your PDE.
4. Visualize the mesh, boundary tags where useful, and the computed field(s). Matplotlib, PyVista, and ParaView are all acceptable if the workflow is documented.
5. Interpret the results physically: where are the bottlenecks, hotspots, boundary layers, stress concentrations, dominant paths, unstable regions, or other relevant structures?
6. Identify which variant performs better with respect to your metric and explain why.

0.8 Backup Option: Stationary Heat Conduction

The intended project is an own-initiative FEM study. If you really do not know what to do, you may use a stationary heat-conduction project only as a **backup option after talking to us**.

A possible backup setup is:

- solve a stationary heat or diffusion equation,
- derive the weak form with Dirichlet, Neumann, and Robin boundary conditions,
- verify by manufactured solution or finite-difference comparison,
- perform a mesh convergence study with L^2 and H^1-type errors,
- create a realistic thermal scenario such as a thermal bridge, heat sink, pipe insulation defect, or bracket-like geometry,
- compare at least two geometries or boundary-condition variants using a quantity such as total heat flux, maximum temperature, or effective thermal resistance.

This fallback is deliberately less open-ended. Use it only if you have discussed the topic choice with us and cannot identify a suitable own project.

0.9 Optional Extensions

If the mandatory scope is complete, you may extend the project with one of the following:

- time-dependent simulation,
- nonlinear material laws or nonlinear PDEs,
- spatially varying coefficients,
- multi-material subdomains,
- mixed finite element methods,
- eigenvalue problems,
- coupling between two physical fields,
- solver-performance comparison for larger meshes,
- a 3D geometry instead of a 2D cross-section,
- comparison of different numerical solvers and/or preconditioners for the linear system.

0.10 Suggested Repository Layout

This is only a suggestion. The final structure will look different depending on your PDE, geometry, solver workflow, and plotting pipeline. Choose a layout that makes your own project reproducible and understandable.

```
project01/
  pyproject.toml or environment.yml
  README.md
  src/
    solver.py
    geometry.py
    ... more Python package files
  tests/
    test_verification.py
  scripts/
    run_convergence.py
```

```
    run_scenario_study.py
  results/
    figures/
  report/
    report.qmd
```

Keep generated results separate from solver source code. Commit the scripts and small configuration files needed to regenerate figures; avoid committing large generated mesh or output files unless you have a clear reason.

1 TODO (for next year)

- Mention that for nonlinear Problems, residual checking and convergence monitoring is important and shall be included.