Report for the course Simulation Software Engineering

The open-source simulation software FEniCSx and my contribution to it in the winter term 2022/23

Julius Herb

09.02.2023

Contents

FEniCS(x): Automated solution of PDEs by the FEM **2**
What is FEniCS(x)? . . . 2
How does it work? . . . 2
Installation . . . 3
Getting started . . . 4
Tutorials . . . 4
Further information . . . 4

RSE infrastructure of FEniCSx **4**
Git workflow and external contributions . . . 4
Virtualization and containers . . . 5
Building and packaging . . . 5
Documentation . . . 6
Testing and CI . . . 6

Contribution to FEniCSx **6**
Overview . . . 6
Contribution process . . . 7
Motivation . . . 7
Issue selection . . . 7
Implementation . . . 7
Review and further course . . . 8
Learnings . . . 9

FEniCS(x): Automated solution of PDEs by the FEM

What is FEniCS(x)?

FEniCS(x) (FE: Finite Elements, CS: Computational Software) is a software project that aims for the automated solution of partial differential equations (PDEs) by using the finite element method (FEM). The project targets mainly mathematicians that have already experience with PDEs and different FEM approaches, but do not want to develop an efficient FEM solver from scratch. An user can simply describe the weak formulation of the PDE and the desired Finite Element specifications and then FEniCS(x) handles the entire assembly and solution process in an optimized way.

FEniCS(x) also supports advanced techniques like saddle point problems, Brezzi-Douglas-Marini Finite Elements or enriched/extended Finite Elements, which are not available in commonly used FEM solvers. FEniCS(x) is actively used in research as Google Scholar lists about 3000 citations to the FEniCS book. Applications range from solid mechanics, fluid mechanics and FSI to electromagnetics - basically any subject where PDEs are involved. A great advantage of FEniCS(x) over commercial FEM solvers is that the user always has full control over the PDE models, the Finite Element discretization and the solver configuration.

Development of FEniCS started in 2003 as a collaboration between the Chalmers University of Technology and the University of Chicago. Nowadays, the development is mainly driven by the FEniCS(x) steering council around long-term maintainer Garth Wells.

In 2019, the FEniCS(x) team started a refactoring process of the entire software. The new edition is called FEniCSx and developed on GitHub. The last stable release of the legacy FEniCS is version 2019.1.0, while FEniCSx has currently the version 0.5.2 (as of February 2023) and is still in active development. Some features of legacy FEniCS are missing in FEniCSx or still experimental and the documentation is not complete yet. In particular, an overview of changes from FEniCS to FEniCSx is not available and an open issue on GitHub. However, the developers recommend to use FEniCSx:

> Now that development is focussed on FEniCSx, updates are made very rarely to the legacy FEniCS library. We recommend that users consider using FEniCSx instead of the legacy library.

How does it work?

FEniCS(x) has the goal to automate the entire workflow of the numerical solution of a PDE. The user provides the weak formulation and defines the solution spaces. Then, FEniCS automatically generates efficient C++ code to assemble the Finite Element system and solves it using a linear algebra backend like PETSc. The generated code supports parallelization using MPI.

FEniCSx is not a monolithic software system, but rather a combination of several building blocks that need to be used together. It consists of the following libraries:

- UFL: Unified Form Language that is used to express the weak formulation of a PDE in Python code
- FFCx: FEniCSx Form Compiler that compiles the weak formulation in UFL to efficient C++ code
- Basix: Provides the Finite Element basis functions for a wide range of Finite Elements (in legacy FEniCS this was provided by FIAT)
- DOLFINx: Contains all necessary data structures, connects the several parts of FEniCS and provides a unifying Python interface

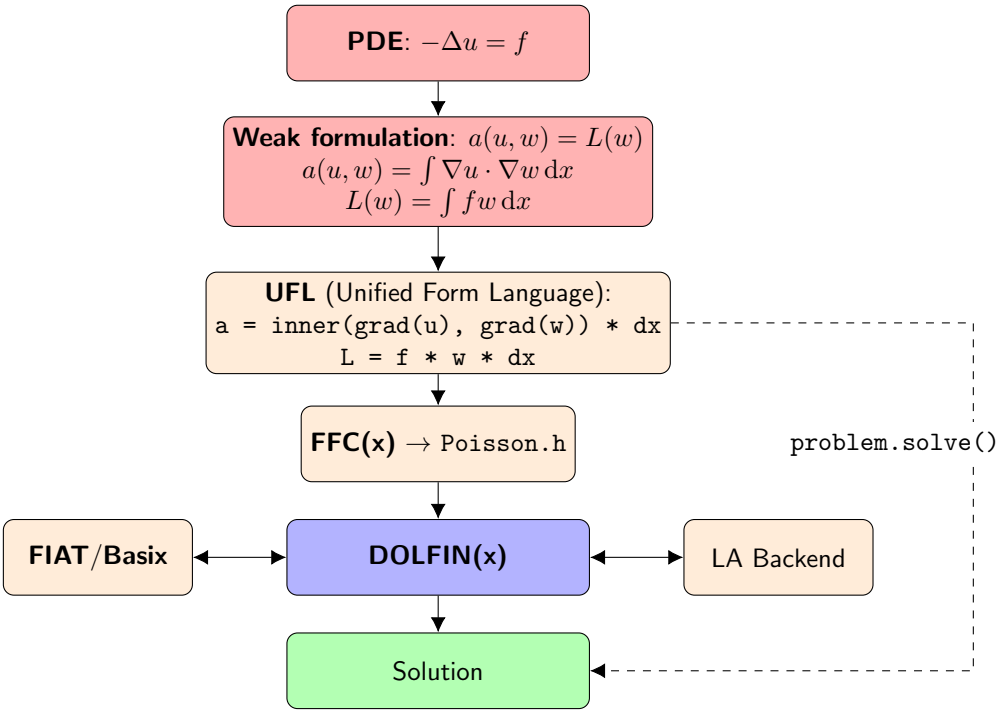

Figure 1: Overview of the FEniCS software project

FEniCSx employs mixed language programming by providing a convenient Python interface and performing the costly computations in C++. The user only needs to call the `solve` function of the Python interface and a JIT compiler handles everything under the hood.

Installation

There are multiple options to obtain a FEniCSx installation:

- Docker container: easiest option

```
docker run -ti dolfinx/dolfinx:stable
```

- Binary packages: easy, but usually not the latest version
- Built from source: This is a bit cumbersome since all parts of FEniCSx have to be built from source separately before compiling DOLFINx. I collected all necessary steps on Ubuntu in an installation script.

Getting started

There are really good tutorials for FEniCSx and legacy FEniCS. I would recommend starting with the Poisson problem and then solving real world problems (e.g. linear elasticity).

Tutorials

- "The FEniCSx Tutorial" by Jørgen S. Dokken (for FEniCSx)
- Solving PDEs in Python: The FEniCS Tutorial I by Hans P. Langtangen and Anders Logg (for legacy FEniCS)

Further information

- Documentation of the DOLFINx Python interface
- Documentation of the DOLFINx C++ API
- Automated Solution of Differential Equations by the Finite Element Method - "The FEniCS Book" by Anders Logg, Kent-Andre Mardal and Garth Wells (for legacy FEniCS)
- Roadmap of the FEniCSx project

RSE infrastructure of FEniCSx

Git workflow and external contributions

The FEniCSx team employs a forking workflow using feature branches on the developer's forks. Even maintainers work on their forks except when finalizing a release. The contribution process starts by creating a fork of the repository. It is recommended to announce new contributions via GitHub issues. After implementing the new feature, a pull request can be made to the main branch of the origin repository. When the test and build pipeline succeeds and the review of a maintainer is successful,

the pull requests can be merged. Whenever a new version is released, the current code on the main branch is merged into the release branch.

The FEniCSx ecosystem consists of various building blocks. Besides the repository for DOLFINx there are separate repositories for ffcx, ufl and basix. Their RSE infrastructure is similar to the DOLFINx repository. In addition, there are further repositories, such as for the documentation and the website.

Possibilities to contribute are explained in CONTRIBUTING.md. Contributions from new developers are welcome but should follow the code of conduct and the style guides of the project such as the C++ style guide. There are also issues targeting new developers that are labeled as "good first issue".

The long-term goals of the project are formulated in the road map on the website. Questions about the usage of FEniCS can be asked on Discourse. The communication between developers takes place on Slack.

Virtualization and containers

The Docker platform is used extensively to test and ship the code. On the one hand, Docker containers are used by the testing and CI pipeline to create a standardized environment. On the other hand, Docker containers can be used easily to run the latest stable release of DOLFINx

```
docker run -ti dolfinx/dolfinx:stable
```

to run the latest nightly build of DOLFINx

```
docker run -ti dolfinx/dolfinx:nightly
```

or to run a Jupyter Lab with the latest stable release of DOLFINx

```
docker run --init -ti -p 8888:8888 dolfinx/lab:stable
```

Building and packaging

The C++ code is built using `cmake` based on `CMakeLists.txt` files. The Python code is packaged using `build` based on the `setup.py` file.

Spack packages are available and can be installed via

```
spack env create fenicsx-env
spack env activate fenicsx-env
spack add py-fenics-dolfinx cflags="-O3" fflags="-O3"
spack install
```

There are also distribution packages, e.g. for Ubuntu and Debian, but they are rather outdated.

Documentation

The documentation of FEniCSx consists of the DOLFINx C++ API reference, the DOLFINx Python API reference, and the API references of the other building blocks of FEniCSx. Besides that, there are some tutorials on the website of a maintainer that are treated as semi-official tutorials.

According to the README.md of the C++ API reference

> The C++ API documentation is generated from the source code using Doxygen, and the Doxygen output is curated and rendered using reStructured text with Sphinx and Breathe.

The Python API reference is created using reStructured text with Sphinx.

Testing and CI

Different tools are used to automate the testing and CI tasks. Besides GitHub Actions, the FEniCSx team also uses CircleCI and SonarCloud for automation purposes.

According to the configuration, GitHub Actions are used to build Docker images, build the software in different configurations, and update the documentation on the website. When you make a pull request, GitHub actions are automatically triggered. CircleCI is used to run the tests for different build configurations. Details can be found in the configuration file.

For C++, unit tests using the test framework Catch2 and some demos are executed automatically. For Python, unit tests are run using pytest.

Contribution to FEniCSx

Overview

I have made the following contributions to improve the demos of FEniCSx:

- Adding a demo for the Stokes equation with various stable pairs of finite elements in PR #2505 that aims to resolve issue #2312
- Adding a Python demo that solves the Poisson equation using matrix-free CG methods in PR #2517 to resolve issue #1776
- Reimplementing Python and C++ demos from legacy FEniCS for FEniCSx that solve the biharmonic equation using a discontinuous Galerkin method in PR #2508
- Adding a DOLFIN to DOLFINx cheat sheet in issue #2330

Apart from that I also made some minor contributions on the way:

- Fixing isort checks for some demos in PR #2506 that has been merged
- Updating the Slack invite link in the `README.md` in PR #2485 that has been merged and on the website in issue FEniCS/web#121
- Slightly improving the semi-official DOLFINx tutorial in PR jorgensd/dolfinx-tutorial#110 that has been merged

Contribution process

Motivation

I have noticed that many users have not yet switched from legacy FEniCS to the new FEniCSx. In my opinion, this is mainly because FEniCSx is not backward compatible and there are few demos available for reference. That makes it difficult for new users to get started with FEniCSx and to migrate code from legacy FEniCS to FEniCSx.

Issue selection

I discovered the following issues that seemed interesting for my contribution:

- Issue #2312: Add a demo using an Enriched Element
- Issue #1776: Add matrix-free solver demos
- Issue #2330: Create a DOLFIN to DOLFINx “cheat sheet”
- Issue #28: Reimplement PointSource

Then I decided as follows:

Issue #28 is one of the oldest issues of FEniCSx and deals with the reimplementation of an important feature. However, the FEniCSx maintainers are not sure yet how exactly the feature should be implemented. Hence, I did not consider this issue further.

Therefore I selected the three other issues for my contributions.

Implementation

I started with issue #2312, i.e. adding a demo using an Enriched Element. A popular Enriched Element is the MINI element which can be used to solve the Stokes equations. Hence, I created a Python demo

for the Stokes equation where various stable pairs of finite elements are compared in PR #2505. The author of the issue and another maintainer appreciated the effort and also suggested extending the demo, which I did then.

After that, I looked through the demos of legacy FEniCS and found the following demos that seemed interesting also for FEniCSx, which I started to prepare:

- Biharmonic C++ demo
- Nonlinear Poisson C++ demo
- Geometric Multigrid Poisson C++ demo

Later I found out that there is already a demo similar to the nonlinear Poisson demo outside the repository. Thus, I did not add my reimplementation as a PR.

Unfortunately, it was not possible to get the multigrid demo to work for FEniCSx without "hacking" since the necessary interpolation operator has been removed.

Instead, I added my reimplementations of Python and C++ demos from legacy DOLFIN that solve the biharmonic equation using a discontinuous Galerkin method in PR #2508.

Then, I turned my focus to the matrix-free solver demo for Python that was requested in issue #1776. There was already a C++ demo from PR #1959, which should be reimplemented for Python. I found several ways to implement matrix-free solvers for FEniCSx in Python. I created PR #2517 to add the implemented demo to FEniCSx.

The first version of my PR did not support MPI parallelization like the C++ demo. Achieving this was not straightforward, as the C++ and Python APIs of DOLFINx are inconsistent about when and how exactly to communicate between processes. After some time of debugging, I was able to solve the problems, making the Python demo behave the same as the C++ demo.

Finally, I have attached a DOLFIN to DOLFINx cheat sheet that I created along the way to issue #2330.

Review and further course

As far as I can tell, my demos are working as intended and the pipeline is running successfully. While some maintainers already reacted positively to my Stokes demo in PR #2505 and the matrix-free solver demo in PR #2517, there was unfortunately no detailed review yet. As soon as this is available, I will adjust the demos accordingly. There is also still no feedback on my biharmonic demos in PR #2508. If the maintainers decide not to merge the demos, I will try to add them to a FEniCSx tutorial outside the repository instead.

Learnings

- Maintainers of large open-source projects tend to be busy, resulting in delayed reviews. Still, they put in a lot of effort and are supportive.
- Even if some maintainers indicate approval, this does not imply that all of them are behind it.
- Don't forget to test your code on a FEniCSx build with `PETSC_ARCH=linux-gnu-complex-32`, since the DOLFINx pipeline runs the demos also on this architecture and with complex numbers things can be different!
- For `petsc4py` it is easier to look in the source code than in the incomplete documentation.
- `jupytext` is a great tool to create a `git diff`-able format for Jupyter notebooks.