



fenicsR13

Release v1.4
Extended gas dynamics using
FEniCS platform.

Navigation

fenicsR13

- fenicsR13 package
 - Submodules
 - fenicsR13.fenicsR13 module
 - fenicsR13.geoToH5 module
 - fenicsR13.input module
 - fenicsR13.meshes module
 - fenicsR13.postprocessor module
 - fenicsR13.solver module
 - fenicsR13.tensoroperation module
 - Module contents
- 2d_heat
 - test_heat_convergence module
- 2d_stress
 - test_stress_convergence module

Last update: Oct 13 22 at 09:48

fenicsR13: A Tensorial Mixed Finite Element Solver for the Linear R13 Equations Using the FEniCS Computing Platform

pipeline passed coverage unknown version 1.4 DOI 10.5281/zenodo.3673039
doc: <https://lamboo.pages.rwth-aachen.de/fenicsR13/>

#extendedGasDynamics #using #FEniCS

@@@

See the paper for a full explanation: *Lambert Theisen and Manuel Torrilhon. 2021. fenicsR13: A Tensorial Mixed Finite Element Solver for the Linear R13 Equations Using the FEniCS Computing Platform. ACM Trans. Math. Softw. 47, 2, Article 17 (April 2021), 29 pages. 10.1145/3442378. Preprint @ arXiv (Free PDF) Final Version @ Journal*

@@@

$$\begin{aligned} \nabla \cdot u &= s^i \\ \nabla p + \nabla \cdot \sigma &= b \\ \nabla \cdot u + \nabla \cdot a &= r \\ \frac{4}{5}(\nabla u)_{ef} + 2(\nabla v)_{ef} + \nabla \cdot m &= -\frac{1}{K} \sigma \\ \frac{5}{2} \nabla \theta + \nabla \cdot \sigma + \frac{1}{2} \nabla \cdot R &= \frac{1}{6} \nabla \Delta \sigma - \frac{1}{K} \frac{2}{3} \sigma^2 \\ m &= -2K(\nabla v)_{ef} \\ R &= \frac{24}{5} K(\nabla v)_{ef} \\ \Delta &= -12K(\nabla \cdot \sigma) \end{aligned}$$

Publishing Reproducible Numerics: A Student's Perspective

RDM Workshop @ SFB 1481: Sparsity and Singular Structures

Lambert Theisen

NMH/IANS @ University of Stuttgart

Aachen, 05 September 2023



University of Stuttgart
Germany



Applied and
Computational
Mathematics



Outline

1. Motivation
2. What Does “Reproducible Numerics” Mean?
3. Tools & Best Practices
 - Reproducible Progress (with Git)
 - Reproducible Environments (with Docker)
 - Reproducible Workflows (with Gitlab/-hub)
4. Archiving Code for Publication (with Zenodo)
 - Archiving Code for Publication (with Zenodo)
5. Conclusion

Outline

1. Motivation
2. What Does “Reproducible Numerics” Mean?
3. Tools & Best Practices
 - Reproducible Progress (with Git)
 - Reproducible Environments (with Docker)
 - Reproducible Workflows (with Gitlab/-hub)
4. Archiving Code for Publication (with Zenodo)
 - Archiving Code for Publication (with Zenodo)
5. Conclusion

Motivation

Status Quo: Many use software for research¹

Relevant Situations:

- You are given a paper as starting point for your project. First step: Understand and reproduce (my M.Sc. thesis)
- You want to continue an old project with a new idea (also as PI)
- You want to change an experiment later (for the PhD thesis?!)

¹[Flemisch Gläser '23] ²[Baker '16']

Motivation

Status Quo: Many use software for research¹

Relevant Situations:

- You are given a paper as starting point for your project. First step: Understand and reproduce (my M.Sc. thesis)
- You want to continue an old project with a new idea (also as PI)
- You want to change an experiment later (for the PhD thesis?!)

Problem: Reproducibility hard in practise²

- >70% failed to reproduce others
- >50% failed to reproduce **own** experiments -,-

⇒ We need reproducability strategies! But: No lab, only “computer”...

¹[Flemisch Gläser '23] ²[Baker '16']

nature

Explore content ▾ About the journal ▾ Publish with us ▾

[nature](#) > [news feature](#) > [article](#)


Published: 25 May 2016

1,500 scientists lift the lid on reproducibility

[Monya Baker](#)

[Nature](#) 533, 452–454 (2016) | [Cite this article](#)

147k Accesses | 2011 Citations | 5169 Altmetric | [Metrics](#)

 This article has been [updated](#)

Survey sheds light on the 'crisis' rocking research.

More than 70% of researchers have tried and failed to reproduce another scientist's experiments, and more than half have failed to reproduce their own experiments. Those are some of the telling figures that emerged from *Nature's* survey of 1,576 researchers who took a brief online questionnaire on reproducibility in research.

Outline

1. Motivation
2. What Does “Reproducible Numerics” Mean?
3. Tools & Best Practices
 - Reproducible Progress (with Git)
 - Reproducible Environments (with Docker)
 - Reproducible Workflows (with Gitlab/-hub)
4. Archiving Code for Publication (with Zenodo)
 - Archiving Code for Publication (with Zenodo)
5. Conclusion

What Does “Reproducible Numerics” Mean?

Definition:

Reproducing the exact numerical outcomes using the same setup.

Why?

- Establishes trust and ensures accuracy
- Building on previous work: standing on the shoulders of giants (avoid reinventing the wheel)

Typical TOCs

▶ 1. Introduction
▶ 2. The Gross–Pitaevskii equation and gradient flows
▶ 3. Adaptive gradient flow finite element discretization
▶ 4. Numerical Experiments
5. Conclusions
References

▶ 1 Introduction
▶ 2 Factorization and Homogenization of the Model Problem
▶ 3 Spatial Discretization and Iterative Eigensolvers
▶ 4 Numerical Experiments
5 Conclusion
References

▶ 1. Introduction
▶ 2. Unfitted DG and Trefftz DG Methods With Exact Geometry
▶ 3. Stabilisation Techniques
▶ 4. Error Analysis
▶ 5. Implementational Aspects
▶ 6. Numerical Examples
Data availability statement
References

What Does “Reproducible Numerics” Mean?

Definition:

Reproducing the exact numerical outcomes using the same setup.

Why?

- Establishes trust and ensures accuracy
- Building on previous work: standing on the shoulders of giants (avoid reinventing the wheel)

What can go wrong?

- Scripts not available, only method “description”
- Missing information, e.g. parameters N , h , η_{learn} , etc
- Unknown software versions, e.g. `julia v1.9.3` vs. `julia v1.6.7 LTS`
- Missing datasets (`mesh.msh` , `training-images.zip` , etc)
- Use of proprietary or unpublished, in-house software
- Other differences (Windows, macOS, Linux, etc) (M1, x64, etc)

Typical TOCs

▶ 1. Introduction
▶ 2. The Gross–Pitaevskii equation and gradient flows
▶ 3. Adaptive gradient flow finite element discretization
▶ 4. Numerical Experiments
5. Conclusions
References

▶ 1 Introduction
▶ 2 Factorization and Homogenization of the Model Problem
▶ 3 Spatial Discretization and Iterative Eigensolvers
▶ 4 Numerical Experiments
5 Conclusion
References

▶ 1. Introduction
▶ 2. Unfitted DG and Trefftz DG Methods With Exact Geometry
▶ 3. Stabilisation Techniques
▶ 4. Error Analysis
▶ 5. Implementational Aspects
▶ 6. Numerical Examples
Data availability statement
References

Outline

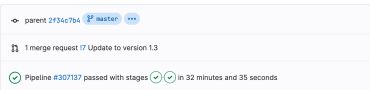
1. Motivation
2. What Does “Reproducible Numerics” Mean?
3. Tools & Best Practices
 - Reproducible Progress (with Git)
 - Reproducible Environments (with Docker)
 - Reproducible Workflows (with Gitlab/-hub)
4. Archiving Code for Publication (with Zenodo)
 - Archiving Code for Publication (with Zenodo)
5. Conclusion

Reproducible Progress (with Git)

Version control with git

- **Problem:** paper-v1.tex → paper-v3-final.tex → paper-v3-FINALLL.tex
- Solution: Track only differences
- Workflow: git add test.py → git commit -m "Add convergence test" → git push
- Allows to go back in time, also allows for asynchronous collaboration
- Nowadays: DevOps platform to track issues, run automated tests, store dependencies, create roadmaps, etc
- Git LFS: For large binary files track are hard to “diff” (mesh.msh, tomography.dat, etc)

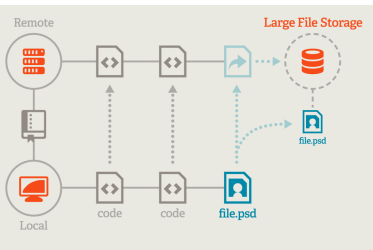
Fix lid example



Changes 1 Pipelines 1

Showing 1 changed file with 1 addition and 1 deletion Hide whitespace changes Inline Side-by-side

```
examples/lid_driven_cavity/input.yml
+1 -1 View file @bf678388
@@ -104,7 +104,7 @@ LidDrivenCavityStudy
 enable: False
 exact_solution: esola/81_coeffs.cpp
 plot: False # to avoid error exit code due to $DISPLAY
+ write_systemmatrix: True
- write_systemmatrix: False
 rascals_pressure: True
 relative_error: True
```



Reproducible Environments (with Docker)

Problem: People have different setups (student has macOS+python3.11 but PI has Windows+python3.7)

⇒ First step: Specify all dependencies

- e.g. `requirements.txt`, `setup.py` for Python
- e.g. `package.yml` and `manifest.yml` for Julia

But what about external deps (gmsh, TensorFlow, FEniCS, ...) or other cfg?

Reproducible Environments (with Docker)

Problem: People have different setups (student has macOS+python3.11 but PI has Windows+python3.7)

⇒ First step: Specify all dependencies

- e.g. requirements.txt, setup.py for Python
- e.g. package.yml and manifest.yml for Julia

But what about external deps (gmsH, TensorFlow, FEniCS, ...) or other cfg?

Virtualization with  **docker:**

- Create common “Cooking recipe” (⇔ Dockerfile): Use *this*, install *that* ...
- Workflow: docker build -tag myimg . → docker run -i -t myimg /bin/bash to open a shell inside container
- IDE Coupling: Possible in VSCode
- Automation: Can be used for testing of code!

Example Dockerfile:

```
Dockerfile
1
2 # Use FEniCS base image
3 FROM quay.io/fenicsproject/stable:2019.1.0.r3
4
5 # Descriptions
6 LABEL maintainer="Lambert Theisen <Lambert.theisen@rwth-aachen.de>"
7 LABEL description="Linearized R13 Equations Solver Environment"
8
9 # Specify software versions
10 ENV GMSH_VERSION 4.4.0
11
12 # Download Install Gmsh SDK with dependencies from Github's dolfinx Dockerfile
13 RUN export DEBIAN_FRONTEND=noninteractive && \
14 apt-get -qq update && \
15 # apt-get -yq --with-new-pkgs -o Dpkg::Options::="--force-confold" upgrade && \
16 apt-get -y install \
17     libgl1 \
18     libxcursor-dev \
19     libxinerama1 && \
20 apt-get clean && \
21 rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
22 RUN cd /usr/local && \
23 wget -nc http://gmsh.info/bin/Linux/gmsh-$(GMSH_VERSION)-Linux64-sdk.tgz && \
24 tar -xzf gmsh-$(GMSH_VERSION)-Linux64-sdk.tgz
25 ENV PATH=/usr/local/gmsh-$(GMSH_VERSION)-Linux64-sdk/bin:$PATH
26
27 # Install additional programs
28 RUN \
29 apt-get update && \
30 apt-get install -y \
31     nundiff \
32     htop \
33     imagemagick
34
35 # Install any needed packages specified in requirements.txt
36 # RUN pip install --trusted-host pypi.python.org -r requirements.txt
37 WORKDIR /fenicsR13
38 ADD ./requirements.txt /fenicsR13/requirements.txt
39 RUN pip install -r /fenicsR13/requirements.txt
40
```

Reproducible Workflows (with Gitlab/-hub)

Explain usage: README.md for installation/execution

Documentation (automated)

- e.g. Sphinx, DocuMenter.jl, doxygen, ...
- Transform code comments (docstrings) to documentation, ideally host via Git pages

```

end{bmatrix}
\begin{bmatrix}
\alpha_{1,2} \\
\alpha_{1,2} \\
\alpha_{1,2} \\
\alpha_{1,2} \\
\alpha_{1,2}
\end{bmatrix}
...

and returns ``\beta_{opt} = \alpha_{0,2} / \alpha_{0,1}`` which is the optimal
stepwidth in the algorithm ``x = \beta_{opt} p``.

...

function RRq(
  A :: AbstractArray{T} where T::Number,
  B :: AbstractArray{T} where T::Number,
  subspace
)

```

ddEigenLab.jl

Home

Examples

2D Lactator plus Prey

Plotting Example

Array Example

The steepest results from the eigenvector components of the gradient state $\mathbf{x} = (x_1, x_2, \dots, x_n)$ and use $\beta_{opt} = \alpha_{0,2} / \alpha_{0,1}, \dots, \alpha_{n,2} / \alpha_{n,1}$ such that the linear combination $\mathbf{x} = \beta_{opt} \mathbf{p}_1 + \dots + \beta_{opt} \mathbf{p}_n$ is the desired solution.

Example

For the Steepest Descent method, it usually yields

$$\begin{bmatrix} p_1^T A x \\ p_2^T A x \\ \vdots \\ p_n^T A x \end{bmatrix} = \lambda \begin{bmatrix} p_1^T x \\ p_2^T x \\ \vdots \\ p_n^T x \end{bmatrix}$$

and returns $\beta_{opt} = \alpha_{0,2} / \alpha_{0,1}$ which is the optimal stepwidth in the algorithm $\mathbf{x} = \beta_{opt} \mathbf{p}$.

Reproducible Workflows (with Gitlab/-hub)

Explain usage: README.md for installation/execution
Documentation (automated)

- e.g. Sphinx, Docstring.jl, doxygen, ...
- Transform code comments (docstrings) to documentation, ideally host via Git pages

Testing with  **GitLab** (RWTH instance) or  **GitHub**

- CI/CD: Function from 6 month ago, still work?
- ⇒ Write tests for your methods
- Ideally: Also test the numerical examples (e.g. numdiff to captured output)
 - **A good team:** Gitlab/-hub with automated tests. Every commit triggers the tests to run. Specifications via, e.g., .gitlab-ci.yml or .github/workflows/[name].yml
 - Useful: Automatic \LaTeX compilation (these slides)

```

\end{matrix}
\begin{matrix}
\alpha_{(1,2)} \\
\alpha_{(1,2)} \\
\vdots
\end{matrix}

and returns "\beta_2 = \alpha_{(0,2)}/\alpha_{(0,2)}" which is the optimal
stepwidth in the algorithm "x = \beta_2 p".

function RRq(
  A :: AbstractArray{T} where T::Number,
  B :: AbstractArray{T} where T::Number,
  subspace
)

```



Home

Examples

2D Eigensolver

Plotting Example

Array Example

$$A \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

The eigenpairs result from the eigenpairs components of the general state $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = 0$ and use $\beta_2 = \alpha_{(0,2)}/\alpha_{(0,2)}$, $\lambda = \alpha_{(0,2)}/\alpha_{(0,2)}$ such that the linear combination $x = \beta_2 p_1 + \dots + \beta_n p_n$ is the desired solution.

Example
For the Chebyshev method, it usually yields
and returns $\beta_2 = \alpha_{(0,2)}/\alpha_{(0,2)}$ which is the optimal stepwidth in the algorithm $x = \beta_2 p$

```

~(gitlab-2-homogenized-reproducible)
File: .gitlab-ci.yml
1 stage: merge/lint [2020-08-05]
2
3 stages:
4   - test
5   - build
6   - verify
7   - CFC_CHECKOUT_STRATEGY: recursive
8
9 test:
10  stages: test
11  tags:
12  - >docker
13  script:
14  - ./lint.sh
15
16 build:
17  stages: build
18  tags:
19  - >docker
20  script:
21  - compile.sh
22  - ./run_diff.sh -n -pdiff -do -doctest -@{[name,?]} | @{[name,?]} | @{[name,?]} | @{[name,?]}
23  - ./run_build.sh -n -pdiff -do -doctest -@{[name,?]} | @{[name,?]} | @{[name,?]} | @{[name,?]}
24  - ./run_verify.sh -n -pdiff -do -doctest -@{[name,?]} | @{[name,?]} | @{[name,?]} | @{[name,?]}
25  - ./run_diff.sh -n -pdiff -do -doctest -@{[name,?]} | @{[name,?]} | @{[name,?]} | @{[name,?]}
26  - ./run_verify.sh -n -pdiff -do -doctest -@{[name,?]} | @{[name,?]} | @{[name,?]} | @{[name,?]}
27
28 artifacts:
29  paths:
30  - ./run_diff_SHA.pdf
31
~(gitlab-2-homogenized-reproducible)

```

Outline

1. Motivation
2. What Does “Reproducible Numerics” Mean?
3. Tools & Best Practices
 - Reproducible Progress (with Git)
 - Reproducible Environments (with Docker)
 - Reproducible Workflows (with Gitlab/-hub)
4. Archiving Code for Publication (with Zenodo)
 - Archiving Code for Publication (with Zenodo)
5. Conclusion

Archiving Code for Publication (with Zenodo)

Setup: Code and paper ready for publication

Problem: Papers have a very long “lifecycle” but internet moves fast (Website links might change, people move)

¹[Uekerman '23]

Archiving Code for Publication (with Zenodo)

Setup: Code and paper ready for publication

Problem: Papers have a very long “lifecycle” but internet moves fast (Website links might change, people move)

Digital Object Identifier (**DOI**) System:

- Organized by DOI Foundation, ISO standard, managed by registration agencies, prefix/suffix¹
- Example: 10.1137/21M1456005
- Resolver: doi.org/10.1137/21M1456005 links to “correct” location

¹[Uekerman '23]

Archiving Code for Publication (with Zenodo)

Setup: Code and paper ready for publication

Problem: Papers have a very long “lifecycle” but internet moves fast (Website links might change, people move)

Digital Object Identifier (**DOI**) System:

- Organized by DOI Foundation, ISO standard, managed by registration agencies, prefix/suffix¹
- Example: 10.1137/21M1456005
- Resolver: doi.org/10.1137/21M1456005 links to “correct” location

Archiving using **zenodo** (or RWTH Publications in AC, DaRUS in STU)

- From CERN, creates DOI (citeable, unchangeable)
- Create versions (v1.0, v1.1, etc) of dataset/software
- Strategy: Git tag, download tarball, upload, cite DOI



¹[Uekerman '23]

Archiving Code for Publication (with Zenodo)

Setup: Code and paper ready for publication

Problem: Papers have a very long “lifecycle” but internet moves fast (Website links might change, people move)

Digital Object Identifier (**DOI**) System:

- Organized by DOI Foundation, ISO standard, managed by registration agencies, prefix/suffix¹
- Example: 10.1137/21M1456005
- Resolver: doi.org/10.1137/21M1456005 links to “correct” location

Archiving using **zenodo** (or **RWTH Publications** in AC, DaRUS in STU)

- From CERN, creates DOI (citeable, unchangeable)
- Create versions (v1.0, v1.1, etc) of dataset/software
- Strategy: Git tag, download tarball, upload, cite DOI



Data availability

As we think that there is even greater potential in the database that we have compiled, we are making it openly accessible for further studies. The vector-based site dataset (shapefile, .shp) with all 4194 occupations and the result of the spatial queries with the environmental variables as attributes is available for download at: <https://doi.org/10.18154/RWTH-2023-06762>. All used environmental geodatasets are already publicly available, and we refer to the respective website for individual download.

¹[Uekerman '23]

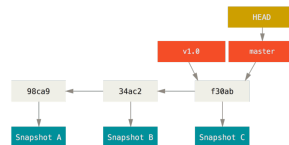
Outline

1. Motivation
2. What Does “Reproducible Numerics” Mean?
3. Tools & Best Practices
 - Reproducible Progress (with Git)
 - Reproducible Environments (with Docker)
 - Reproducible Workflows (with Gitlab/-hub)
4. Archiving Code for Publication (with Zenodo)
 - Archiving Code for Publication (with Zenodo)
5. Conclusion

Conclusion

Reproducible ...progress with git

- Backup, collaboration, DevOps platform



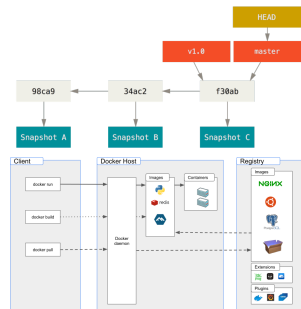
Conclusion

Reproducible ...progress with git

- Backup, collaboration, DevOps platform

...environments with docker

- Specify all dependencies, create “cooking recipe”
- Hardware/software independence



Conclusion

Reproducible . . . progress with git

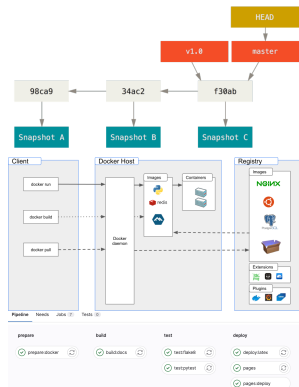
- Backup, collaboration, DevOps platform

. . . environments with docker

- Specify all dependencies, create “cooking recipe”
- Hardware/software independence

. . . workflows with GitLab or GitHub

- CI/CD automation (docu, testing, hosting, . . .)
- Confidence by automatic testing



Conclusion

Reproducible . . . progress with git

- Backup, collaboration, DevOps platform

. . . environments with docker

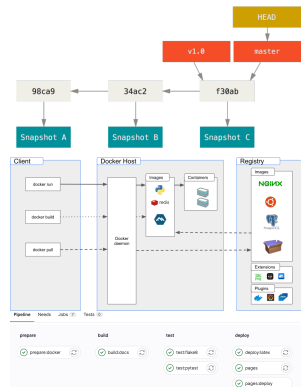
- Specify all dependencies, create “cooking recipe”
- Hardware/software independence

. . . workflows with GitLab or GitHub

- CI/CD automation (docu, testing, hosting, . . .)
- Confidence by automatic testing

Permanent archiving using zenodo

- Get permanent DOI identifier (citeable)



The previously described methods are implemented using `NGStreffitz` [53] and `ngaxfes` [54], add-on packages to the finite element library `NGSoive/Netgen` [55,51]. The python scripts implementing the methods discussed in this paper and the full numerical results presented below are freely available in the zenodo repository [24]. The stabilisation parameters for the interior penalty and the Nitsche method are fixed in all examples to $\beta = 10k^2$ and the ghost penalty scaling to $\gamma = 0.01$ if not mentioned otherwise. For the solution of linear systems arising in the numerical examples, we used sparse direct solvers.

[24] F. Heimann, C. Lehrenfeld, F. Stoeckl, and H. von Wahl. Unfitted Trefftz discontinuous Galerkin methods for elliptic boundary value problems - Reproduction scripts, doi: 10.5281/zenodo.3025934, 2022.

Conclusion

Reproducible ...progress with git

- Backup, collaboration, DevOps platform

...environments with docker

- Specify all dependencies, create “cooking recipe”
- Hardware/software independence

...workflows with GitLab or GitHub

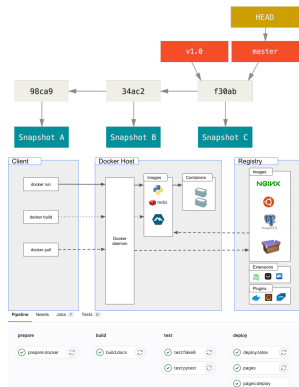
- CI/CD automation (docu, testing, hosting, ...)
- Confidence by automatic testing

Permanent archiving using zenodo

- Get permanent DOI identifier (citeable)

Other aspects

- Interactivity (Jupyter notebooks), ...



The previously described methods are implemented using `NGStreffitz` [53] and `ngaxfex` [54], add-on packages to the finite element library `NGsolve/Netgen` [55,51]. The python scripts implementing the methods discussed in this paper and the full numerical results presented below are freely available in the zenodo repository [24]. The stabilisation parameters for the interior penalty and the Nitsche method are fixed in all examples to $\beta = 10\epsilon^2$ and the ghost penalty scaling to $\gamma = 0.01$ if not mentioned otherwise. For the solution of linear systems arising in the numerical examples, we used sparse direct solvers.

[24] F. Heimann, C. Lehrenfeld, F. Stöckli, and H. von Wahl. Unfitted Trefftz discontinuous Galerkin methods for elliptic boundary value problems - Reproduction scripts, doi: 10.5281/zenodo.3022934, 2022.

References I

- [1] M. Baker. “1,500 Scientists Lift the Lid on Reproducibility”. In: *Nature* 533.7604 (2016), pp. 452–454. (Visited on 09/04/2023).
- [2] B. Flemisch and D. Gläser. *Sustainable-Simulation-Software / Course-Material* · *GitLab*. <https://gitlab.com/sustainable-simulation-software/course-material>. 2023. (Visited on 09/04/2023).
- [3] B. Uekermann. *Simulation-Software-Engineering/Lecture-Material: Material for the Simulation Software Engineering Lecture*. <https://github.com/Simulation-Software-Engineering/Lecture-Material/tree/main>. (Visited on 09/04/2023).

Questions?



`lambert.theisen@ians.uni-stuttgart.de`



`lambert.theisen@rwth-aachen.de`



@lamB000



<https://ths.n.dev>



NMH @ Institute of Applied Analysis and Numerical Simulation



University of Stuttgart



German Research Foundation (DFG) project 411724963

Thank You.