Interfaces

Thomas Wick

[1] Institut für Angewandte Mathematik (IfAM)
[2] Cluster of Excellence PhoenixD
Leibniz Universität Hannover, Germany
[3] Guest researcher at Laboratoire de Mécanique et Technologie
École Normale Supérieure Université Paris-Saclay, France

`https://thomaswick.org`

Jan 7-8, 2021
Paris online, WCCM ECCOMAS 2020 congress
Link to CSMA Junior section workshop

Preface

This course is designed for students, doctoral researchers and peers who had basic classes in calculus (analysis), introduction to numerical methods, differential equations, and finite element discretizations. Classes in continuum mechanics and multiphysics problems are additionally useful.

The **goal** is to provide an overview about **interfaces**. Practical exemplification is provided by considering fluid-structure interaction.

This course is organized into two parts:

A) **What I do:** 30-minute presentation on interfaces

→ **What you learn:** mathematical concepts and challenges, numerical realization, types, and coupling strategies

B) **What I do:** 90-minute joint cloud coding, online discussions on the open-source code step-fsi.cc on fluid-structure interaction implemented in deal.II (www.dealii.org, C++) and running with qarnot and repl.it (cloud computing)

→ **What you learn:** illustration of the interface between fluid and solid, programming code realization of interface coupling conditions with variational-monolithic coupling, possible change of finite elements, time-stepping schemes, implementation of Newton's method, black-box direct solver UMFPACK, evaluation of quantities of interest, comparison of qualitative (graphical) and quantitative results with benchmark values, several questions concerning the interface

Thomas Wick

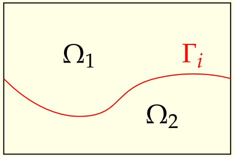
Paris online, January 2021

Overview

What is an interface?

- Notation in this class Γ_i, where the i stands for interface
- Interfaces are also known as internal boundaries
- Interface divides medium $\Omega \subset \mathbb{R}^d$ (where d is the dimension) into at least two subdomains Ω_1 and Ω_2
- Lower-dimensional manifold $\mathbb{R}^{d-1}$

Ω_1 : subdomain 1
Ω_2 : subdomain 2
Γ_i : interface



- Example: $d = 2$. Domain $\Omega \subset \mathbb{R}^2$ (plane) and interface $\Gamma \subset \mathbb{R}$ (line).

Typical situation and first challenges

- Task: **Solve boundary value problem** (BVP) or initial-boundary value problem (IBVP) on a domain Ω with interface Γ_i
- Coefficients may have discontinuities of the first kind (jump discont.)

$\rightarrow$ Diffraction problem[1] [2], generalized problems[3], coupled (interface) problems

- Theory of PDEs request to work with Hilbert and Sobolev spaces

$\rightarrow$ Due to properties of the Lebesgue integral, Γ_i is of Lebesgue measure zero in Ω

$\rightarrow$ Need for trace theorems, extension theorems and embedding results in order to ensure $u|_{\Gamma_i} \in H^s(\Gamma_i)$ [4][5]

[1]Ladyzhenskaya, Springer, 1985

[2]Ladyzhenskaja/Solonnikov/Uralceva; AMS Vol. 23, 1968

[3]Lions/Magenes, tome 1, chapitre 3, section 4, 1968

[4]Lions/Magenes, Dunod, 1968

[5]Hackbusch, Springer, 2017

Further mathematical challenges

- Mathematical regularity of Γ_i such that normal vectors, tangential vectors, interface conditions, and models are well-defined
- Interface Γ_i touches other boundaries (e.g., outer boundary $\partial\Omega$) with different interface/boundary conditions: singularities in the solution
- Identification/derivation of interface coupling conditions: kinematic and dynamic (lateral) conditions
- Jumping model and material coefficients or entire change of constitutive laws (e.g., fluid-structure interaction where continuity of normal fluid/solid stresses must be realized)

Further numerical challenges

- Interface evolution: curvature, speed, parametrization, shape (specifically in higher dimensions)
- Numerical treatment of interface discretization
- Design of coupling strategies
- Implementation of interface coupling conditions
- Numerical analysis (stability, convergence) of the coupling
- Types: Fixed, moving, propagating interfaces
- A **zoo of methods** have been proposed to cope with these challenges; please see next slide

$\rightarrow$ **Focus in this class on such computational/scientific computing aspects**

Some key concepts and literature

1. Interface-tracking (see later for precise definition)

→ arbitrary Lagrangian-Eulerian, Deforming-Spatial-Domain/Stabilized Space-Time Technique

2. Interface-capturing (see later for precise definition)

→ Diffusive approaches such as level-set, phase-field methods

3. Fitted methods

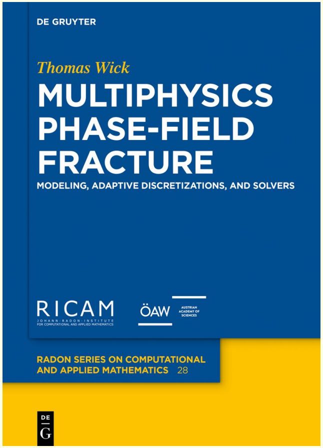
→ Locally finite elements, interface-fitted subspace projection methods or historical work by Babuska 1970

4. Unfitted methods

→ XFEM/GFEM, cut-cell methods, fictious domain methods, immersed boundary methods, finite cell methods

Main literature of this class [6]

- Section 1.3 Interfaces
- Section 3.3 Coupled problems and multiphysics PDEs
- Section 3.4 Fixed interface: Biot (porous media) coupled with elasticity
- Section 8 Propagating interfaces (fracture)
- Section 11 Multiphysics/fluid-structure interaction (partially)
- Section 11.8 / 12.4 Moving interfaces


DE GRUYTER
Thomas Wick
MULTIPHYSICS PHASE-FIELD FRACTURE
MODELING, ADAPTIVE DISCRETIZATIONS, AND SOLVERS
RICAM
ÖAW
RADON SERIES ON COMPUTATIONAL AND APPLIED MATHEMATICS 28

[6]Figures used in this presentation are drawn from the raw data of published figures used in this book. References to original studies are made therein.

Definition

Definition (Interface)

Let $\Omega_1 := \Omega_1(t) \subset \mathbb{R}^d$ and $\Omega_2 := \Omega_2(t) \subset \mathbb{R}^d$ be domains that may vary in time and with Lipschitz boundary $\partial\Omega$. Let the total domain be defined as $\Omega := \Omega_1 \cup \Omega_2$. The interface is then defined by

$$\Gamma(t) := \overline{\Omega}_1(t) \cap \overline{\Omega}_2(t).$$

Often, we use the short hand notation $\Gamma := \Gamma(t)$. The interface Γ is an evolving hypersurface in $\mathbb{R}^{d-1}$ if there exists a time $T > 0$ such that

1. $\Gamma(t, x)$ is a smooth hypersurface in $\mathbb{R} \times \mathbb{R}^d$
2. there exist smooth hypersurfaces $\Gamma(t)$ in $\mathbb{R}^d$ such that
$$\Gamma = \{(t, x) |\, t \in (0, T), x \in \Gamma(t)\}$$
3. the tangential spaces $Tang_{(t,x)}(\Gamma)$ are not space-like, i.e.,
$$Tang_{(t,x)}(\Gamma) \neq \{0\} \times \mathbb{R}^d \quad \forall (t, x) \in \Gamma$$

Interface conditions for second-order PDEs

Definition (Interface coupling conditions; scalar-valued case)

Let $u_1 : \Omega_1 \to \mathbb{R}$ and $u_2 : \Omega_2 \to \mathbb{R}$. For second-order-in-space equations (for instance two Poisson problems $-\nabla \cdot (\alpha \nabla u_1) = f_1$ and $-\nabla \cdot (\beta \nabla u_2) = f_2$), we need to impose two interface conditions on Γ:

$$u_1 = u_2,$$
$$\sigma_1(u_1)n = \sigma_2(u_2)n,$$

with the normal vector n, e.g., $n := n_1$ and the relation $n_1 = -n_2$. The first condition is a **kinematic condition** and the second is the continuity of normal stresses, also known as balance of contact forces, the so-called **dynamic (lateral) condition**. The first condition is a **Dirichlet-like condition**, which needs to be imposed in the function space. The second one is a **Neumann-type condition** and appears through integration by parts in the PDE.

Interface conditions for second-order PDEs

Example

The dynamic coupling condition reads for two Poisson operators:

$$\alpha \nabla u_1 n = \beta \nabla u_2 n,$$

i.e.,

$$\sigma_1(u_1) := \alpha \nabla u_1,$$
$$\sigma_2(u_2) := \beta \nabla u_2.$$

Interface types

Definition (Types of interfaces)

Interfaces can be divided into three types with increasing level of difficulty:

1. **Fixed interfaces** that are fixed in space, time, and area (length). Examples are domain decomposition methods and problems in subsurface modeling with over- and underburden;
2. **Moving interfaces** that vary in space/time and do not (or only slightly) change their area. Examples are two-phase flow problems and fluid-structure interaction;
3. **Propagating interfaces** that vary in space/time and significantly extend their area (length). Examples are damage and fracture problems.

Interface types

Ω_1 : Domain 1
Ω_2 : Domain 2
$\Gamma_i, i = 1, 2, 3$: interfaces

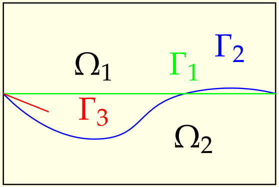


Figure: Interface types: Γ_1: fixed interface, Γ_2: moving interface in time, Γ_3: propagating interface.

Example of a fixed interface

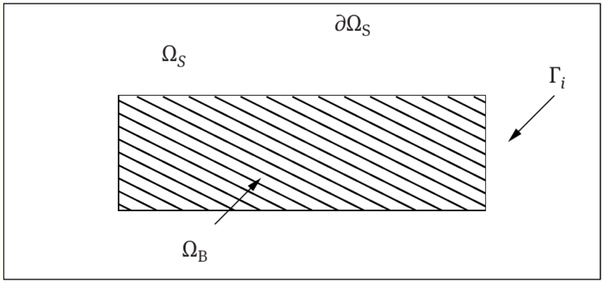


Figure: Augmented Mandel's problem: coupling elasticity in Ω_S with poro-elasticity in Ω_B. The interface Γ_i between both subdomains is fixed.

- In addition domain decomposition (DD)[7] methods are famous examples

[7] Toselli/Widlund, Springer, 2005

Example of a moving interface

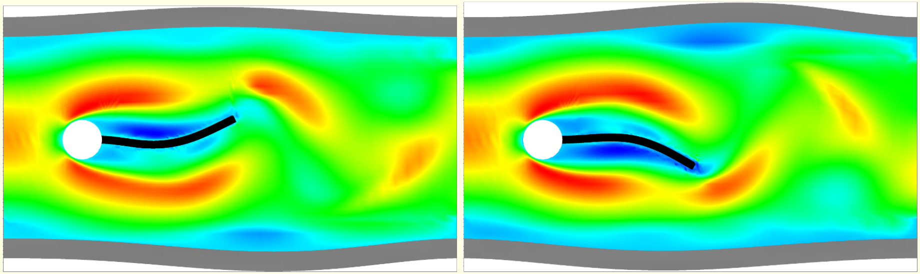


Figure: Coupling the incompressible Navier-Stokes equations with nonlinear elasticity: fluid-structure interaction. Here, the black and grey interfaces are moving in space and time.

Example of a propagating interface

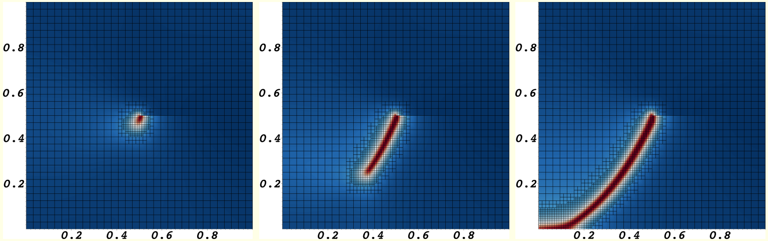


Figure: Propagating interface (in red) in terms of a fracture.

Applications

- Domain decomposition
- Two-phase flow / multiphase flow
- Gas-solid, gas-fluid, fluid-solid interactions
- Multiphysics problems
- Phase transition processes such as solidification
- Fracture propagation
- Free boundary problems
- Contact problems

Interface tracking and interface capturing

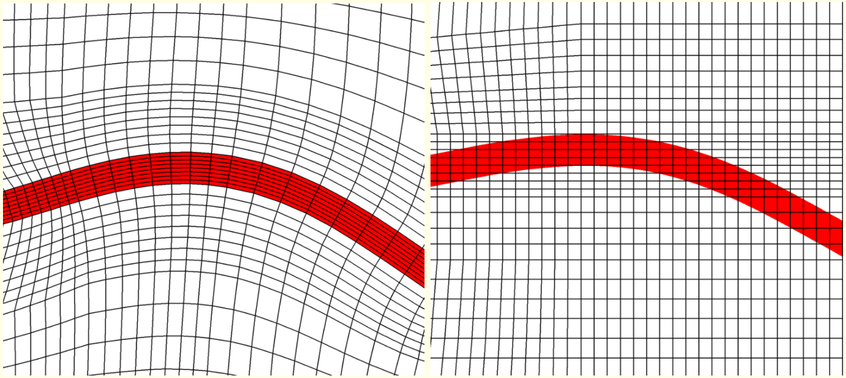


Figure: Comparison of the Lagrangian perspective with interface-tracking of the red object (left) and an Eulerian fixed-mesh approach with interface-capturing of the red object (right). In the former, the interface is located on mesh edges. In the latter, the interface cuts through mesh elements and is captured with an additional marking function.

Interface tracking and interface capturing

Definition (Interface-tracking)

In interface-tracking methods, the interface is explicitly described. Here, we need meshes that enable us to track the interface. Consequently, the mesh needs to be updated as the interface evolves further in time. One shortcoming is that mesh elements may be deformed too much, such that the approach fails, if re-meshing is not taken care of in a proper way.

Definition (Interface-capturing)

In interface-capturing methods, the computations are done on a fixed spatial domain and the interface is implicitly defined through additional functions. The interface function captures the interface and marks its location. Mesh degeneration is not a problem, but conserving well-posedness of the interface marker function is important.

Sharp versus diffusive interface representations

- In interface capturing, sharp and diffusive approximations can be further distinguished
- Smeared zones/mushy zone/transition zone arise in the latter

→ Need to find zero level set (taking right figure and arriving at the left one) if interface needs to be explicitly known by reconstruction.

→ Model error $O(\varepsilon)$ where ε is the size of the diffusive zone.

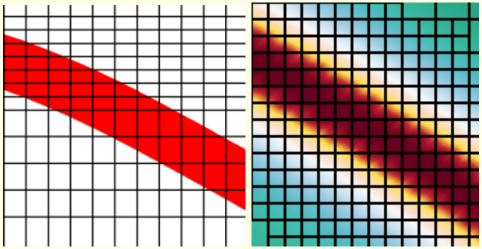


Figure: Comparison of a sharp interface resolution and a diffusive interface (yellow-light-red transition zone).

Pros and cons

- The decision whether to work with interface-tracking or interface-capturing techniques is a compromise between algorithmical, implementational, and computational efforts in terms of accuracy of the desired interface approximation and their mathematical rigorous justification.
- While in general, interface-capturing approaches can deal more easily with moving and evolving interfaces up to topology changes, the accuracy is lower than comparable interface-tracking approaches.
- The latter ones are, however, more challenging when interfaces are propagating because the mesh must be adapted. This holds in particular true for 3D.

Strategies for coupling equations on interface domains

Two principles algorithms exist:

- Iterative coupling, also known as partitioned or staggered approaches: iterate between given PDEs/CVISs[8] while only solving for the actual solution variable in a specific PDE/CVIS and fixing all others.
- Monolithic approaches: solving equations simultaneously for the given unknowns.
- In the following, we consider the abstract problem: Find $U = (u_1, u_2) \in V_1 \times V_2$ such that:

$$A_1(U)(\psi_1) = F(\psi_1) \quad \forall \psi_1 \in V_1, \tag{1}$$

$$A_2(U)(\psi_2) = F(\psi_2) \quad \forall \psi_2 \in V_2. \tag{2}$$

[8]CVIS: coupled variational inequality system.

Partitioned schemes for coupled problems

For coupled problems, often, partitioned schemes are employed. The idea is to iterate between the different equations per time step.

Algorithm

Let initial iteration guesses u_1^0 and u_2^0 be given. For $j = 1, 2, 3, \dots$ iterate:

$$
\begin{aligned}
\textit{Given } u_2^{j-1}, \textit{find } u_1^j: &\quad A_1(u_1^j, u_2^{j-1})(\psi_1) = F(\psi_1),\\
\textit{Given } u_1^{j}, \textit{find } u_2^j: &\quad A_2(u_1^j, u_2^{j})(\psi_2) = F(\psi_2).
\end{aligned}
$$

Check a stopping criterium:

$$\max(\|u_1^j - u_1^{j-1}\|, \|u_2^j - u_2^{j-1}\|) < TOL$$

If the stopping criterion is fulfilled, stop. If not, increment $j \to j + 1$.

Variational-monolithic coupling

Definition (Variational-monolithic coupling)

In variational-monolithic coupling the coupling conditions are realized in an exact fashion in the weak (i.e., variational) formulation.

- Common function space is the first task:

$$X = V_1 \times V_2.$$

Formulation

Let $U := (u_1, u_2) \in X$. Sum-up both equations (1) *and* (2) *such that*

$$\textit{Find } U \in X: \quad A(U)(\Psi) = F(\Psi) \quad \forall \Psi \in X,$$

where $\Psi = (\psi_1, \psi_2) \in X$ *and*

$$A(U)(\Psi) := A_1((u_1, u_2))(\psi_1) + A_2((u_1, u_2))(\psi_2).$$

Objectives

- **Fluid-structure interaction** as an example of an **interface** problem
- Interface divides incompressible flow (Navier-Stokes) and (geometrically) nonlinear elasticity
- Interface type No. 2 (moving interface)

Software and cloud computing

Software:

- `C++`, e.g., `https://www.cplusplus.com/`
- deal.II `www.dealii.org` : open-source finite element library
- Fluid-Structure interaction code in ANS (Archive of Numerical Software) `https://media.archnumsoft.org/10305/`
- Updates of this code to newer deal.II versions (9.x.x), please see `https://thomaswick.org/gallery_engl.html`

Cloud computing and interactive code:

- repl.it `https://repl.it/team/CSMAJunior`

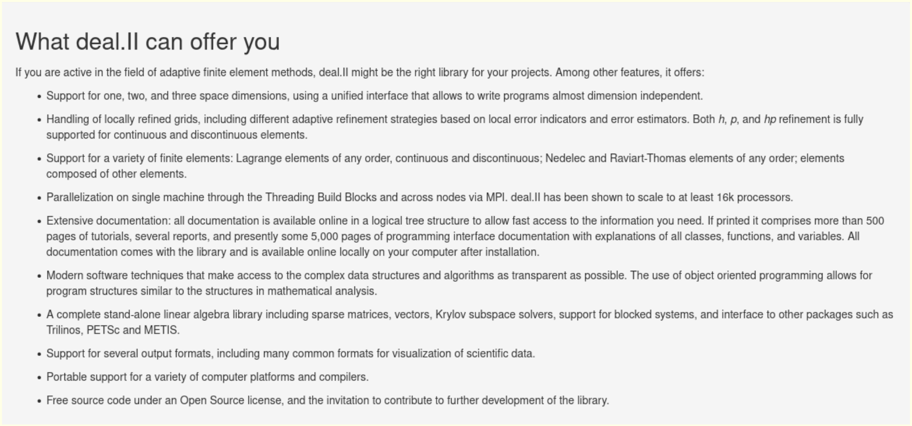
$\rightarrow$ Cloud computing platform with software and importing qarnot

- qarnot `https://console.qarnot.com/`

$\rightarrow$ Cloud computing importing deal.II installation as a docker container

- Interactive discussions via chat function in repl.it and Microsoft Teams

deal.II: differential equations analysis library [9] [10]



What deal.II can offer you

If you are active in the field of adaptive finite element methods, deal.II might be the right library for your projects. Among other features, it offers:

- Support for one, two, and three space dimensions, using a unified interface that allows to write programs almost dimension independent.
- Handling of locally refined grids, including different adaptive refinement strategies based on local error indicators and error estimators. Both *h*, *p*, and *hp* refinement is fully supported for continuous and discontinuous elements.
- Support for a variety of finite elements: Lagrange elements of any order, continuous and discontinuous; Nedelec and Raviart-Thomas elements of any order; elements composed of other elements.
- Parallelization on single machine through the Threading Build Blocks and across nodes via MPI. deal.II has been shown to scale to at least 16k processors.
- Extensive documentation: all documentation is available online in a logical tree structure to allow fast access to the information you need. If printed it comprises more than 500 pages of tutorials, several reports, and presently some 5,000 pages of programming interface documentation with explanations of all classes, functions, and variables. All documentation comes with the library and is available online locally on your computer after installation.
- Modern software techniques that make access to the complex data structures and algorithms as transparent as possible. The use of object oriented programming allows for program structures similar to the structures in mathematical analysis.
- A complete stand-alone linear algebra library including sparse matrices, vectors, Krylov subspace solvers, support for blocked systems, and interface to other packages such as Trilinos, PETSc and METIS.
- Support for several output formats, including many common formats for visualization of scientific data.
- Portable support for a variety of computer platforms and compilers.
- Free source code under an Open Source license, and the invitation to contribute to further development of the library.

Figure: deal.II offering to us: copy and paste from the webpage.

[9] Arndt et al.; CAMWA, 2020
[10] `www.dealii.org`

qarnot

- Registering via `https://account.qarnot.com/register.`
- Vous recevez automatiquement 60 heures de calcul lors de l'inscription
- Main link `https://computing.qarnot.com/`
- Terminal `https://console.qarnot.com`
- Documentations:
  - `https://computing.qarnot.com/developers/overview/qarnot-computing-home`
  - Python SDK : `https://computing.qarnot.com/documentation/sdk-python/`

Running deal.II with qarnot

- In zip-folder: `c++_deal.ii`
- `config  input_resources  run.py`
- The main files to work with are in `input_resources`
- `CMakeLists.txt  fsi.inp  running_script.sh  step-fsi.cc`
- Therein:
    1. `CMakeLists.txt` cmake file to build code
    2. `fsi.inp` mesh file
    3. `running_script.sh` shell execution script from qarnot
    4. the source file with the actual implementation is `step-fsi.cc`

Running deal.II via personal terminal or cloud repl.it

- Two options to run code from qarnot
- Personal computer (does NOT allow interactive exchange on code!)
- Running the code in terminal `user_wick>`:
  ```
  user_wick> ./run.py
  ```
- For **joint code development in the frame of this workshop**

$\rightarrow$ Use repl.it

- Work and run within repl.it together on the code
- Save way: run code via green **Run** button in the middle of the window
- Alternatively (after full installation only) : go into `Shell` and therein
  ```
  ~/InterfaceSession3$ python main.py
  ```
- My pre-installed code **step-fsi.cc will run approximately 8 minutes!**

Innocent information on the other interface types

- **Fixed interface**: Biot equations / Augmented Mandel's problem DOpElib `www.dopelib.net` (based on deal.II)

→ `dopelib/Examples/PDE/InstatPDE/Example6`

- **Moving interface**: Fluid-structure interaction

→ `http://www.thomaswick.org/gallery_engl.html`

→ Alternatively `dopelib/Examples/PDE/InstatPDE/Example2`

- **Propagating interface**: Phase-field fracture Heister/Wick 2018,2020

→ `https://github.com/tjhei/cracks`

→ pfm-cracks

Code for the online session (I)

- We concentrate on the FSI code `step-fsi.cc` originally published on `https://media.archnumsoft.org/10305/`
- Code in principle dimension-independent via template parameter
- Numerical ingredients:
  1. variational-monolithic formulation,
  2. spatial discretization on quadrilaterals with Galerkin finite elements, here $(v, u, p) \in Q_2^c \times Q_2^c \times P_1^{dc}$
  3. time-stepping based on A-stable second-order time stepping schemes (Crank-Nicolson, shifted CN),
  4. nonlinear solution with Newton's method and line search,
  5. linear solution with UMFPACK (direct solver)

Code for the online session (II)

- Postprocessing and quantities of interest:
  1. Displacements of tip of elastic beam: u_x, u_y,
  2. drag and lift values
  3. Newton iteration numbers,
  4. comparison to benchmark values, computational convergence in space and time (robustness with respect to discretization parameters),
  5. variations in material parameters: robustness of solver

Biggest bottleneck:

- Major computational cost goes into the linear solver for many DoFs and 3D configurations
- Governing code was further extended in current in-house version by Daniel Jodlbauer in Jodlbauer, Langer, Wick; IJNME, 2019

$\rightarrow$ Again: a (block-)preconditioner for the iterative solution (we use GMRES) only works robustly, when interfacial degrees of freedom are explicitly considered ...

Disclaimer

In the following, you do not need to understand all details.
The materials are added for a self-contained presentation.
The main purpose is to discuss **interface** related aspects!

Fluids and solids in their standard systems

Equations for fluid flows (Navier Stokes) - Eulerian

$$\partial_t v + (v \cdot \nabla v) - \nabla \cdot \sigma(v,p) = 0, \quad \nabla \cdot v = 0, \quad \text{in } \Omega_f \times I,$$
$$+\text{bc and initial conditions}$$

with Cauchy stress tensor $\sigma(v,p) = -pI + \rho_f \nu_f (\nabla v + \nabla v^T)$.

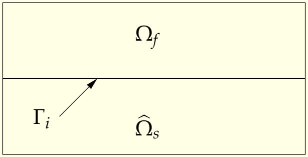
Equations for (nonlinear) elasticity - Lagrangian

$$\partial_t^2 \hat{u} - \hat{\nabla} \cdot (\hat{F}\widehat{\Sigma}(\hat{u})) = 0 \quad \text{in } \widehat{\Omega}_s \times I,$$
$$+\text{bc and initial conditions}$$

with the stress $\widehat{\Sigma}(\hat{u}) = \widehat{F}^{-1}\widehat{P} = 2\mu_s \hat{E} + \lambda_s \text{trace}(\hat{E})I$, the strain $\hat{E} = \frac{1}{2}(\widehat{F}^T\widehat{F} - I)$ and $\widehat{F} = I + \widehat{\nabla}\hat{u}$.

Coupling conditions on $\Gamma_i, \hat{\Gamma}_i$

$$v_f = \hat{v}_s \quad \text{and} \quad \sigma(v,p)n_f = -\widehat{F}\widehat{\Sigma}(\hat{u})\hat{n}_s.$$



Coupling conditions on interface Γ_i

- Kinematic coupling condition (physics):
$$\hat{v}_f = \hat{v}_s \quad \text{on } \hat{\Gamma}_i.$$
- Dynamic coupling condition (physics):
$$\hat{J}\hat{\sigma}_f \widehat{F}^{-T}\hat{n}_f = \widehat{F}\widehat{\Sigma}\hat{n}_f \quad \text{on } \hat{\Gamma}_i.$$
- Geometric coupling condition:
$$\hat{u}_f = \hat{u}_s \quad \text{on } \hat{\Gamma}_i.$$

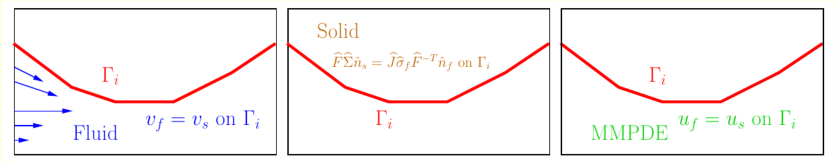

Γ_i
Fluid
$v_f = v_s$ on Γ_i
Solid
$\widehat{F}\widehat{\Sigma}\hat{n}_s = \widehat{J}\widehat{\sigma}_f\widehat{F}^{-T}\hat{n}_f$ on Γ_i
Γ_i
Γ_i
MMPDE
$u_f = u_s$ on Γ_i

Figure: Illustration of the three coupling conditions on Γ_i (respectively its corresponding definition on the fixed $\hat{\Gamma}_i$): v_f is required to solve the fluid system, $\widehat{F}\Sigma_s\hat{n}_s$ is required for the solid system, and u_f is necessary for the MMPDE (mesh motion PDE).

Variational-monolithic coupling

- The continuity of flow conditions,
$$v_f = v_s,$$
and the geometric coupling
$$u_f = u_s,$$
are incorporated directly in the Sobolev spaces as usually done for **Dirichlet conditions**:
$$\hat{V}^0_{f,\hat{v}} := \{\hat{v}_f \in H^1_0(\widehat{\Omega}_f) : \hat{v}_f = \hat{v}_s \text{ on } \hat{\Gamma}_i\},$$
$$\hat{V}^0_{f,\hat{u}} := \{\hat{u}_f \in H^1_0(\widehat{\Omega}_f) : \hat{u}_f = \hat{u}_s \text{ on } \hat{\Gamma}_i\}.$$
- In variational-monolithic coupling, **Neumann-like** interface conditions are fulfilled in a weak sense on the continuous level:
$$\langle \hat{J}\hat{\sigma}_f \widehat{F}^{-T}\hat{n}_f, \hat{\varphi}\rangle - \langle \widehat{F}\widehat{\Sigma}\hat{n}_f, \hat{\varphi}\rangle = 0 \quad \forall \hat{\varphi} \in \hat{V}^0_{f,\hat{v}}.$$
- They are realized through interface integrals (but actually **disappear** in the later model because of their **continuity in the weak (variational) sense**), provided that $\varphi_f = \varphi_s$ on $\hat{\Gamma}_i$, as it is guaranteed with the space $\hat{V}^0_{f,\hat{v}}$ here.

Variational-monolithic ALE fluid-structure interaction

Formulation

Find vector-valued velocities, vector-valued displacements and a scalar-valued fluid pressure, i.e., $\hat U := \{\hat v_f, \hat v_s, \hat u_f, \hat u_s, \hat p_f\} \in \widehat X_D^0 := \{\hat v_f^D + \hat V^0_{f,\hat v}\} \times \hat L_s \times \{\hat u_f^D + \hat V^0_{f,\hat u}\} \times \{\hat u_s^D + \hat V^0_s\} \times \hat L_f^0$, *such that* $\hat v_f(0) = \hat v_f^0$, $\hat v_s(0) = \hat v_s^0$, $\hat u_f(0) = \hat u_f^0$, *and* $\hat u_s(0) = \hat u_s^0$ *are satisfied, and for almost all times* $t \in I$ *holds:*

Fluid momentum $\begin{cases} \hat A_1(\hat U)(\hat\psi^v) := (\hat J \hat\varrho_f \partial_t \hat v_f, \hat\psi^v)_{\widehat\Omega_f} + (\hat\varrho_f \hat J (\widehat F^{-1}(\hat v_f - \hat w)\cdot\widehat\nabla)\hat v_f), \hat\psi^v)_{\widehat\Omega_f} \\ +(\hat J \hat\sigma_f \widehat F^{-T}, \widehat\nabla \hat\psi^v)_{\widehat\Omega_f} + \langle \hat\varrho_f \nu_f \hat J (\widehat F^{-T} \widehat\nabla \hat v_f^T \hat n_f)\widehat F^{-T}, \hat\psi^v\rangle_{\hat\Gamma_{out}} = 0 \quad \forall \hat\psi^v \in \hat V^0_{f,\hat\Gamma_i}, \end{cases}$

Solid, v-equation $\left\{ \hat A_2(\hat U)(\hat\psi^v) := (\hat\varrho_s \partial_t \hat v_s, \hat\psi^v)_{\widehat\Omega_s} + (\widehat F \widehat\Sigma, \widehat\nabla \hat\psi^v)_{\widehat\Omega_s} = 0 \, \forall \hat\psi^v \in \hat V_s^0 \right.$

Fluid mesh motion $\left\{ \hat A_3(\hat U)(\hat\psi^u) := (\hat\sigma_{mesh}, \widehat\nabla \hat\psi^u)_{\widehat\Omega_f} \quad = 0 \quad \forall \hat\psi^u \in \hat V^0_{f,\hat u,\hat\Gamma_i}, \right.$

Solid, u-equation $\left\{ \hat A_4(\hat U)(\hat\psi^u) := \hat\varrho_s(\partial_t \hat u_s - \hat v_s, \hat\psi^u)_{\widehat\Omega_s} \quad = 0 \quad \forall \hat\psi^u \in \hat L_s, \right.$

Fluid mass conservation $\left\{ \hat A_5(\hat U)(\hat\psi^p) := (\widehat{div}\,(\hat J \widehat F^{-1} \hat v_f), \hat\psi^p)_{\widehat\Omega_f} \quad = 0 \quad \forall \hat\psi^p \in \hat L_f^0. \right.$

- The green terms represents the respective terms in the ‘standard’ coordinate systems

→ True for all solid equations, because they remain in Lagrangian coordinates

→ The fluid has been transformed from Ω to $\widehat\Omega$ using the ALE technology.

- The **red** terms are novel due to the ALE transformations.

Time and spatially discretized abstract problem

Formulation (Abstract problem)

Given $\hat{U}^0 \in \widehat{X}$ *finding* $\hat{U} = (\hat{U}_h^n)_{n=1}^N \in \widehat{X}_h^N$ *solving*

$$\sum_{n=1}^{N} \Bigg(\hat{A}_{T,k}(\hat{U}_h^n, \hat{U}_h^{n-1})(\hat{\Psi}_h^n) + \theta k \hat{A}_E(\hat{U}_h^n)(\hat{\Psi}_h^n) \\ + k\hat{A}_P(\hat{U}_h^n)(\hat{\Psi}_h^n) + k\hat{A}_I(\hat{U}_h^n)(\hat{\Psi}_h^n) + (1-\theta)k\hat{A}_E(\hat{U}_h^{n-1})(\hat{\Psi}_h^n) \Bigg) = 0, \tag{3}$$

for all $(\hat{\Psi}_h^n)_{n=1}^N \in \widehat{X}_h^N$.

- $k = t_n - t_{n-1}$ time step size, h spatial discretization parameter
- $\theta \in [0,1]$, One-Step-θ time discretization
- $\hat{A}_{T,k}$ contains terms with time derivatives, $\hat{A}_E$ contains explicit terms (diffusion, convection, ...), $\hat{A}_P$ contains pressure terms (fully implicit in time), $\hat{A}_I$ contains further implicit terms such as ALE mesh motion
- This problem is still nonlinear and is treated with Newton's method

Nonlinear and linear solution

- Recall: we are sitting at time step t_n. To solve for U_h^{n+1} at t_{n+1} we utilize Newton's method; here defect-correct version
- At each Newton step (Index j), we have

$$\underbrace{A'(U_h^{n,j})(\delta U_h, \Psi_h)}_{=A\delta U} = \underbrace{-A(U_h^{n,j})(\Psi_h)}_{=B}$$

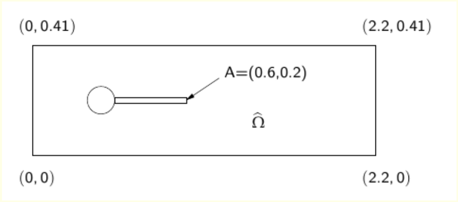
$$U_h^{n,j+1} = U_h^{n,j} + \lambda \delta U_h, \quad \lambda \in [0,1]$$

Thus, we obtain linear equation system:

$$A\delta U = B.$$

- Up to 10^5 unknowns (2D), 10^4 unknowns (3D), direct solvers work still okay and are used for this reason in this class.

Configuration



The computational domain has length $L = 2.5m$ and height $H = 0.41m$. The circle center is positioned at $C = (0.2m, 0.2m)$ with radius $r = 0.05m$. The elastic beam has length $l = 0.35m$ and height $h = 0.02m$. The right lower end is positioned at $(0.6m, 0.19m)$, and the left end is attached to the circle.

Boundary and initial conditions

- For the upper, lower, boundaries, the 'no-slip' conditions for velocity and no zero displacement for structure are given. At the outlet $\hat{\Gamma}_{out}$, the 'do-nothing' outflow condition is imposed leading to a zero mean value of the pressure at this part of the boundary.
- A parabolic inflow velocity profile is given on $\hat{\Gamma}_{in}$ by

$$v_f(0,y) = 1.5\bar{U}\frac{4y(H-y)}{H^2}, \quad \bar{U} = 1.0ms^{-1}.$$

  At the outlet $\hat{\Gamma}_{out}$ the 'do-nothing' outflow condition is imposed which lead to zero mean value of the pressure at this part of the boundary.
- For the non-steady tests one should start with a smooth increase of the velocity profile in time. We use

$$v_f(t;0,y) = \begin{cases} v_f(0,y)\frac{1-\cos(\frac{\pi}{2}t)}{2} & \text{if } t < 2.0s \\ v_f(0,y) & \text{otherwise.} \end{cases}$$

Parameters

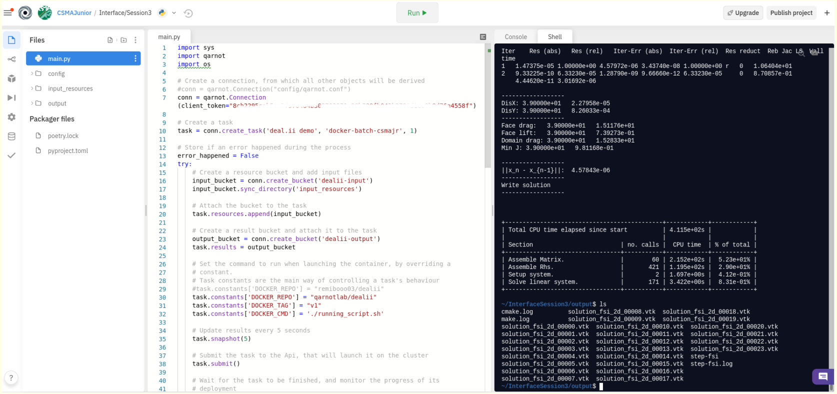
- For the fluid we use $\varrho_f = 10^3 kgm^{-3}$, $\nu_f = 10^{-3} m^2 s^{-1}$.
- The elastic solid is characterized by $\varrho_s = 10^4 kgm^{-3}$, $\nu_s = 0.4$, $\mu_s = 5 * 10^5 kgm^{-1}s^{-2}$.
- All these parameters can be modified within the cc-file.
- **Three different test cases:** FSI1, FSI2, FSI3 according to Hron and Turek, Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow, 2006. `https://www.springer.com/gp/book/9783540345954`

$\rightarrow$ Change in code in line

```
// Defining test cases
test_case = "FSI1";
```

Some results

- You find the results in the folder `output`
- `*.vtk` can be displayed with visit or paraview
- The log-file with the quantities of interest can be displayed via `cat step-fsi.log`



```python
import sys
import qarnot
import os

# Create a connection, from which all other objects will be derived
#conn = qarnot.Connection("config/qarnot.conf")
conn = qarnot.Connection
(client_token="8cb2295[redacted]a4558f")

# Create a task
task = conn.create_task('deal.ii demo', 'docker-batch-csmajr', 1)

# Store if an error happened during the process
error_happened = False
try:
    # Create a resource bucket and add input files
    input_bucket = conn.create_bucket('dealii-input')
    input_bucket.sync_directory('input_resources')

    # Attach the bucket to the task
    task.resources.append(input_bucket)

    # Create a result bucket and attach it to the task
    output_bucket = conn.create_bucket('dealii-output')
    task.results = output_bucket

    # Set the command to run when launching the container, by overriding a
    # constant.
    # Task constants are the main way of controlling a task's behaviour
    #task.constants['DOCKER_REPO'] = "remibooo03/dealii"
    task.constants['DOCKER_REPO'] = "qarnotlab/dealii"
    task.constants['DOCKER_TAG'] = "v1"
    task.constants['DOCKER_CMD'] = './running_script.sh'

    # Update results every 5 seconds
    task.snapshot(5)

    # Submit the task to the Api, that will launch it on the cluster
    task.submit()

    # Wait for the task to be finished, and monitor the progress of its
    # deployment
```

```
Iter    Res (abs)   Res (rel)   Iter-Err (abs)  Iter-Err (rel)  Res reduct  Reb Jac LS  Wall
time
1   1.47375e-05 1.00000e+00 4.57972e-06 3.43740e-08 1.00000e+00 r   0   1.06404e+01
2   9.33225e-10 6.33230e-05 1.28790e-09 9.66660e-12 6.33230e-05     0   8.70857e-01
    4.44620e-11 3.01692e-06

------------------
DisX: 3.90000e+01   2.27958e-05
DisY: 3.90000e+01   8.26033e-04
------------------
Face drag:   3.90000e+01   1.51176e+01
Face lift:   3.90000e+01   7.39273e-01
Domain drag: 3.90000e+01   1.52833e+01
Min J: 3.90000e+01   9.81168e-01

------------------
||x_n - x_{n-1}||:  4.57843e-06
------------------
Write solution
------------------



+---------------------------------------------+------------+------------+
| Total CPU time elapsed since start          | 4.115e+02s |            |
|                                             |            |            |
| Section                         | no. calls |  CPU time  | % of total |
+---------------------------------+-----------+------------+------------+
| Assemble Matrix.                |        60 | 2.152e+02s |  5.23e+01% |
| Assemble Rhs.                   |       421 | 1.195e+02s |  2.90e+01% |
| Setup system.                   |         2 | 1.697e+00s |  4.12e-01% |
| Solve linear system.            |       171 | 3.422e+00s |  8.31e-01% |
+---------------------------------+-----------+------------+------------+

~/InterfaceSession3/output$ ls
cmake.log            solution_fsi_2d_00008.vtk  solution_fsi_2d_00018.vtk
make.log             solution_fsi_2d_00009.vtk  solution_fsi_2d_00019.vtk
solution_fsi_2d_00000.vtk  solution_fsi_2d_00010.vtk  solution_fsi_2d_00020.vtk
solution_fsi_2d_00001.vtk  solution_fsi_2d_00011.vtk  solution_fsi_2d_00021.vtk
solution_fsi_2d_00002.vtk  solution_fsi_2d_00012.vtk  solution_fsi_2d_00022.vtk
solution_fsi_2d_00003.vtk  solution_fsi_2d_00013.vtk  solution_fsi_2d_00023.vtk
solution_fsi_2d_00004.vtk  solution_fsi_2d_00014.vtk  step-fsi
solution_fsi_2d_00005.vtk  solution_fsi_2d_00015.vtk  step-fsi.log
solution_fsi_2d_00006.vtk  solution_fsi_2d_00016.vtk
solution_fsi_2d_00007.vtk  solution_fsi_2d_00017.vtk
~/InterfaceSession3/output$
```

Some results

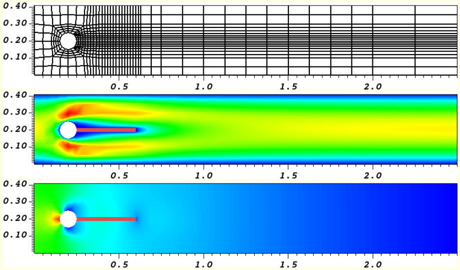


Figure: Displaying *.vtk files. FSI 1 benchmark: mesh, x-velocity v_x solution, and pressure p. The interface Γ_i is around the elastic obstacle (red/orange elastic bar) and the surrounding blue/green colored flow/pressure.

Quantities of interest (obtained from step-fsi.log):

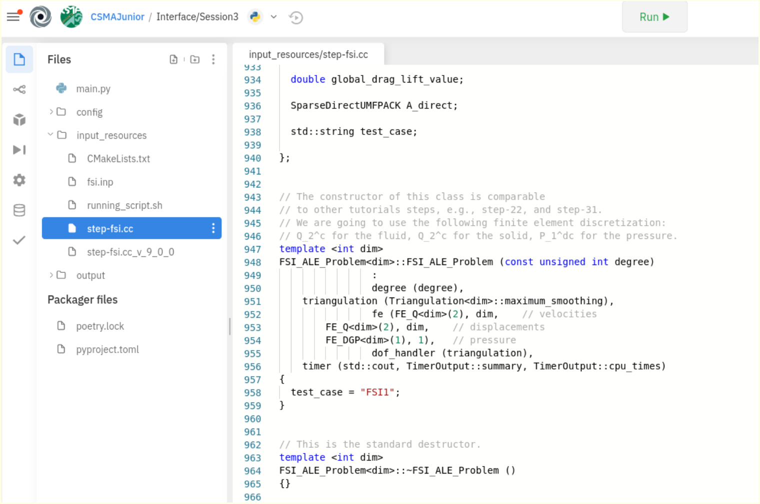
```
DisX:        3.90000e+01   2.27958e-05
DisY:        3.90000e+01   8.26033e-04
------------------
Face drag:   3.90000e+01   1.51176e+01
Face lift:   3.90000e+01   7.39273e-01
Min J:       3.90000e+01   9.81168e-01
```

Your turn ...

- In the following, I give some hints what to study with the help of the code.
- Specifically, you may compare with slide p. 54
- Your own ideas and further questions during our discussions are of course welcome!

Specific tasks (I)

- Make yourself familiar with the code `step-fsi.cc`



```
  double global_drag_lift_value;

  SparseDirectUMFPACK A_direct;

  std::string test_case;

};


// The constructor of this class is comparable
// to other tutorials steps, e.g., step-22, and step-31.
// We are going to use the following finite element discretization:
// Q_2^c for the fluid, Q_2^c for the solid, P_1^dc for the pressure.
template <int dim>
FSI_ALE_Problem<dim>::FSI_ALE_Problem (const unsigned int degree)
                :
                degree (degree),
    triangulation (Triangulation<dim>::maximum_smoothing),
                fe (FE_Q<dim>(2), dim,     // velocities
        FE_Q<dim>(2), dim,     // displacements
        FE_DGP<dim>(1), 1),    // pressure
                dof_handler (triangulation),
    timer (std::cout, TimerOutput::summary, TimerOutput::cpu_times)
{
  test_case = "FSI1";
}


// This is the standard destructor.
template <int dim>
FSI_ALE_Problem<dim>::~FSI_ALE_Problem ()
{}

```

Specific tasks (II)

- Run FSI 1 and compare the output functionals (slide p. 54) and computational performance
- You may speed-up a bit the code by changing in line 3313 `if (refinement_cycle <= 1)` from 1 to 0
- Change inflow velocity from 0.2 to 0.4 in `BoundaryCondition Inflow<dim>::value(...)`

$\rightarrow$ What do you observe?

- Change material parameters in `set_runtime_parameters ()` to some other values of your choice

$\rightarrow$ What do you observe?

- Change time-stepping scheme in `set_runtime_parameters ()` and/or the time step size

$\rightarrow$ What do you observe?

- Display solution (*.vtk)

$\rightarrow$ Need to copy to your personal computer/laptop

Specific tasks (III)

- Change the finite elements (be careful with the inf-sup condition required for the Navier-Stokes part)

```
fe (FE_Q<dim>(2), dim,    // velocities
    FE_Q<dim>(2), dim,    // displacements
    FE_DGP<dim>(1), 1),   // pressure
```

- On the **interface:**

  1. Where and how is the interface realized in the code?
  2. Is interface-tracking or interface-capturing used?
  3. What are the interface conditions?
  4. What are further observations or challenges w.r.t. the interface?

- If you still have time/motivation: Run FSI 2 (might not fully work out in 90 minutes ...)

Observations and learning effects

- Interface problems (specifically moving and propagating) often result in complex programming codes
- Extensive numerical modeling and debugging often necessary
- Code often will not run in real-time, but takes several minutes, hours, days, weeks[11]
- My decision to provide you this FSI code was on purpose: a simple Poisson problem in two domains with a (fixed) interface on mesh faces, would have shown the interface, but not really the true work amount that is behind!

[11]My longest computation of a similar FSI code ran about 2 months

Conclusions

Conclusions

1. Definition and types of interfaces
2. Algorithmic realizations and coupling strategies
3. Joint cloud coding using an open-source benchmark code in fluid-structure interaction

Key references

1. T. Wick; Multiphysics phase-field fracture, Radon Series on computational and applied mathematics, Band 28, 2020 `http://www.degruyter.com/books/978-3-11-049656-7`
2. Links to programming codes on `https://thomaswick.org/gallery_engl.html`

The end

Thanks for participating in this course and happy coding!

Thomas Wick
Institute of Applied Mathematics
Leibniz University Hannover
`https://thomaswick.org`