A General Methodology for Symbolically Generating Manufactured Solutions Satisfying Prescribed Conditions – Application to Two-Phase Flows Equations

David Henneaux

Pierre Schrooyen

Philippe Chatelain

Thierry Magin

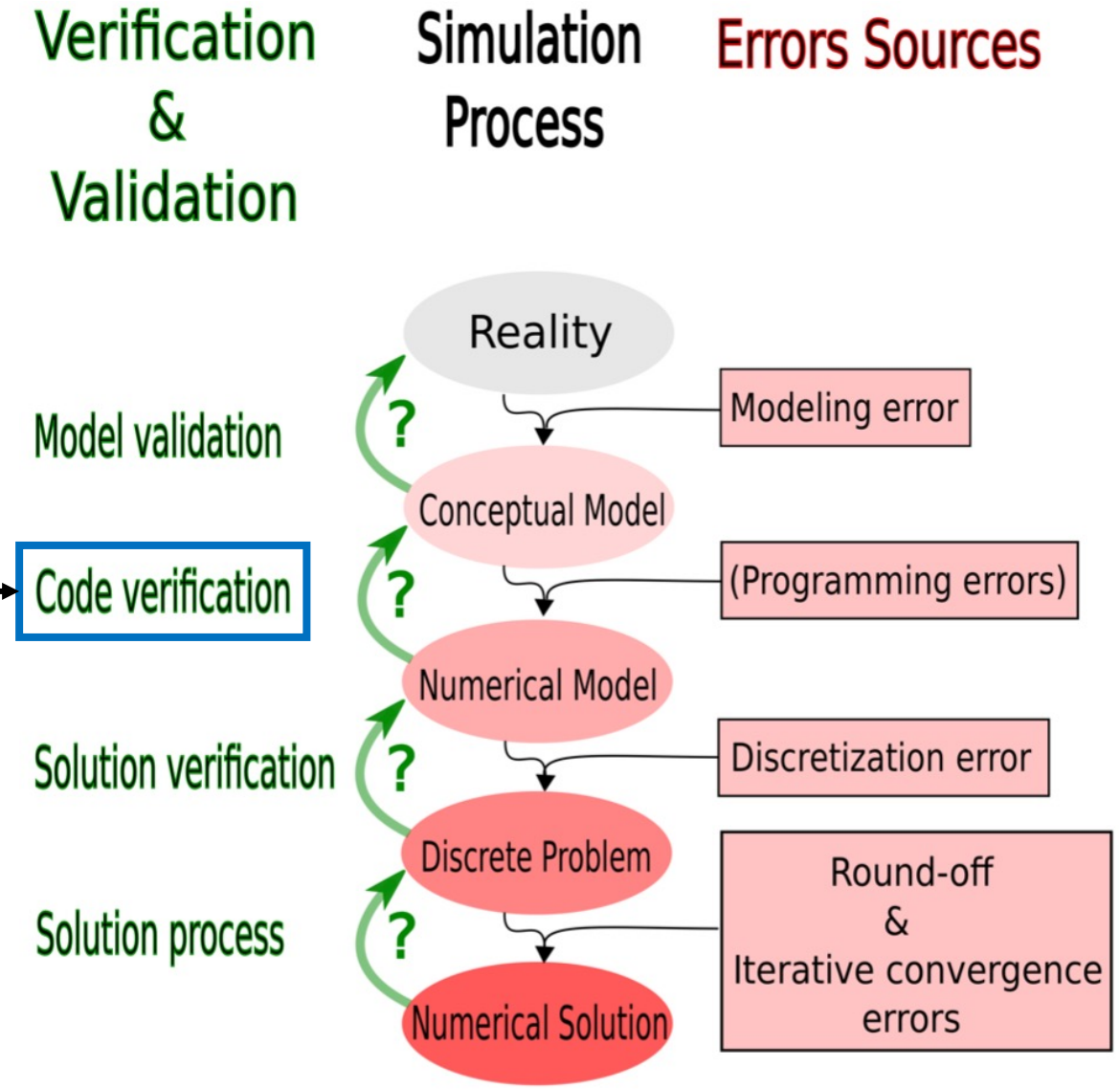
SYMCOMP 2023 Conference

30 March 2023

von KARMAN INSTITUTE FOR FLUID DYNAMICS

Verification and Validation in scientific simulation

Focus on code verification: a crucial step in the V&V process



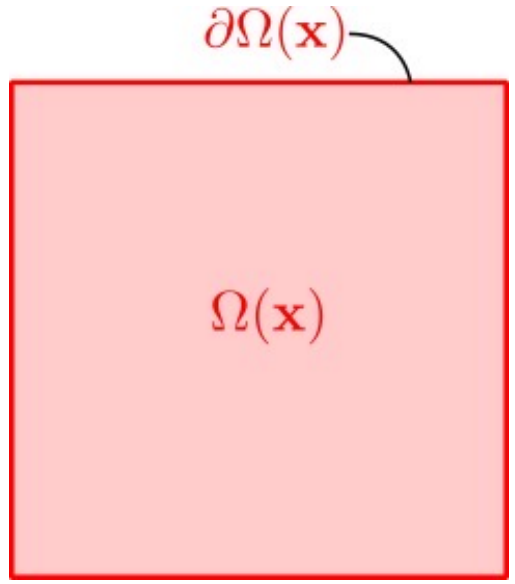
[1] Navah et al. (2018)

1. The Method of Manufactured Solution (MMS)
2. Building MS satisfying prescribed conditions with PyManufSol
3. Applications to two-phase Navier-Stokes equations
4. Conclusions and perspectives

1. The Method of Manufactured Solution (MMS)
2. Building MS satisfying prescribed conditions with PyManufSol
3. Applications to two-phase Navier-Stokes equations
4. Conclusions and perspectives

Building the reference solution and the residual source term

$$\begin{cases} \mathbf{F}(\mathbf{u}(\mathbf{x},t)) = 0 \ , & \forall \mathbf{x} \in \Omega(\mathbf{x}),\ \forall t > 0 \ , \\ \mathbf{B}(\mathbf{u}(\mathbf{x},t)) = 0 \ , & \forall \mathbf{x} \in \partial\Omega(\mathbf{x}),\ \forall t > 0 \ , \\ \mathbf{I}(\mathbf{u}(\mathbf{x},t)) = 0 \ , & \forall \mathbf{x} \in \Omega(\mathbf{x}),\ t = 0 \ . \end{cases}$$



1. Choose the form of the Manufactured Solution (MS)

$$\tilde{\mathbf{u}} = \tilde{\mathbf{u}}(\mathbf{x},t) \neq \mathbf{u}(\mathbf{x},t)$$

2. Insert the MS into the bulk governing equations to get source term

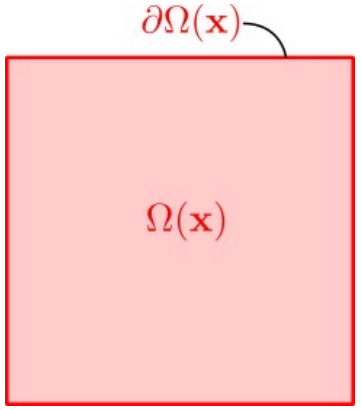
$$\mathbf{F}(\tilde{\mathbf{u}}(\mathbf{x},t)) = \mathbf{S}(\mathbf{x},t) \neq 0$$

[2] Steinberg & Roache (1985)

Code verification by discretizing the modified problem

- The problem to solve becomes

$$\begin{cases} \widetilde{\mathbf{F}}(\tilde{\mathbf{u}}(\mathbf{x},t)) = \mathbf{F}(\tilde{\mathbf{u}}(\mathbf{x},t)) - \mathbf{S}(\mathbf{x},t) = 0 \ , & \forall \mathbf{x} \in \Omega(\mathbf{x}),\ \forall t > 0 \ , \\ \widetilde{\mathbf{B}}(\tilde{\mathbf{u}}(\mathbf{x},t)) = \tilde{\mathbf{u}} - \tilde{\mathbf{u}}(\mathbf{x}_{\mathrm{BC}},t) = 0 \ , & \forall \mathbf{x} \in \partial\Omega(\mathbf{x}),\ \forall t > 0 \ , \\ \widetilde{\mathbf{I}}(\tilde{\mathbf{u}}(\mathbf{x},t)) = \tilde{\mathbf{u}} - \tilde{\mathbf{u}}(\mathbf{x},t=0) = 0 \ , & \forall \mathbf{x} \in \Omega(\mathbf{x}),\ t = 0 \ , \end{cases}$$

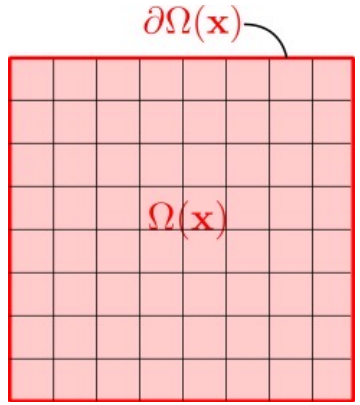


- Discretize it with the solver to be verified

$$\begin{cases} \widetilde{\mathbf{F}}(\tilde{\mathbf{u}}^h(\mathbf{x},t)) = 0 \ , & \forall \mathbf{x} \in \Omega(\mathbf{x}),\ \forall t > 0 \ , \\ \widetilde{\mathbf{B}}(\tilde{\mathbf{u}}^h(\mathbf{x},t)) = 0 \ , & \forall \mathbf{x} \in \partial\Omega(\mathbf{x}),\ \forall t > 0 \ , \\ \widetilde{\mathbf{I}}(\tilde{\mathbf{u}}^h(\mathbf{x},t)) = 0 \ , & \forall \mathbf{x} \in \Omega(\mathbf{x}),\ t = 0 \ , \end{cases}$$

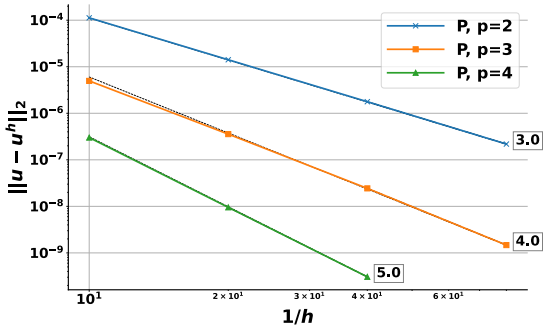
$\tilde{\mathbf{u}}$: continuous

$\tilde{\mathbf{u}}^h$: discrete



- Perform an order of accuracy test

$$\|\tilde{\mathbf{u}} - \tilde{\mathbf{u}}^h\| \leq C\, h^p$$



Assets and limitations of the standard MMS

- Elegant and easy to apply procedure
- Allows to detect different kind of errors in the code
  - Coding mistakes (e.g. "-" instead of "+")
  - Implemented models (e.g. one component of the flux is wrong)
  - Algorithmic mistakes (e.g. non-linear solver steps not correctly followed)
  - Mathematical well-posedness (e.g. unstable discretization)
  - …

$$B_i(\tilde{\mathbf{u}}) = \mathbf{u} - \mathbf{u}(\mathbf{x}_{\mathrm{BC}}, t) \quad \forall B_i$$

- But auxiliary conditions (IC,BC) not considered during the manufacturing step → their implemention into the code cannot be verified
  - Specialized BC can potentially be a critical element of the code!

Development of a general methodology inspired by the case-dependent existing strategies

- Two main existing strategies
  - Multiply the MS by a function representing the boundary geometry [3, 4]

  → possibility to make such that the MS takes the original BC value on the boundary
  - Make the MS dependent on free parameters to be fixed based on some chosen constraints [5, 6]

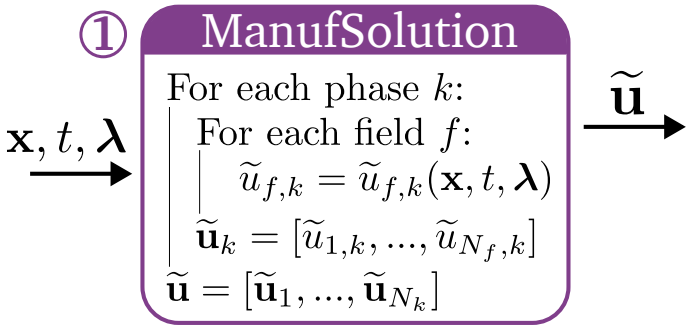
- Objective of this work
  - Establish a generic numerical framework to build MS satisfying prescribed constraints (based on the idea of MS depending on free parameters)

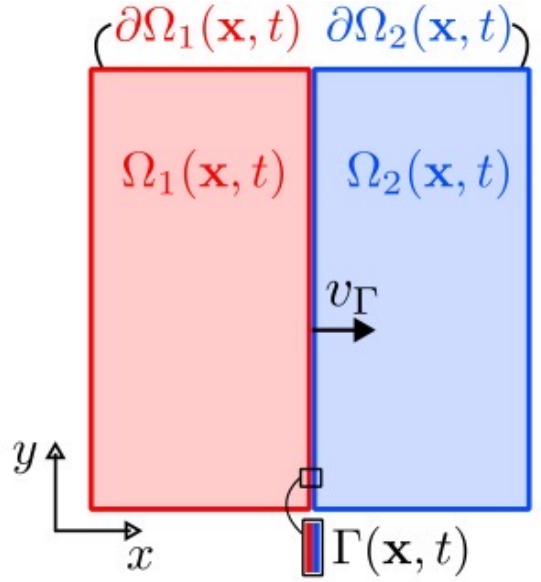
$$B_i(\tilde{\mathbf{u}}) = \mathbf{u} - \mathbf{u}(\mathbf{x}_{\mathrm{BC}}, t) \quad \forall B_i \longrightarrow \begin{array}{lll} B_i(\tilde{\mathbf{u}}) = B_i(\mathbf{u}) & \text{if} & B_i \in \mathbf{C} \\ B_i(\tilde{\mathbf{u}}) = \mathbf{u} - \mathbf{u}(\mathbf{x}_{\mathrm{BC}}, t) & \text{if} & B_i \notin \mathbf{C} \end{array}$$

[3] Bond et al. (2007)
[4] Choudhary et al. (2016)
[5] Brady et al. (2012)
[6] Freno et al. (2022)

1. The Method of Manufactured Solution (MMS)
2. Building MS satisfying prescribed conditions with PyManufSol
3. Applications to two-phase Navier-Stokes equations
4. Conclusions and perspectives

Step 1 – choosing the shape of the MS





$$\widetilde{\mathbf{u}} = \widetilde{\mathbf{u}}(\mathbf{x}, t, \boldsymbol{\lambda})$$

$$\widetilde{u}_1 = a_1 + b_1 \sin(\pi x/4)$$

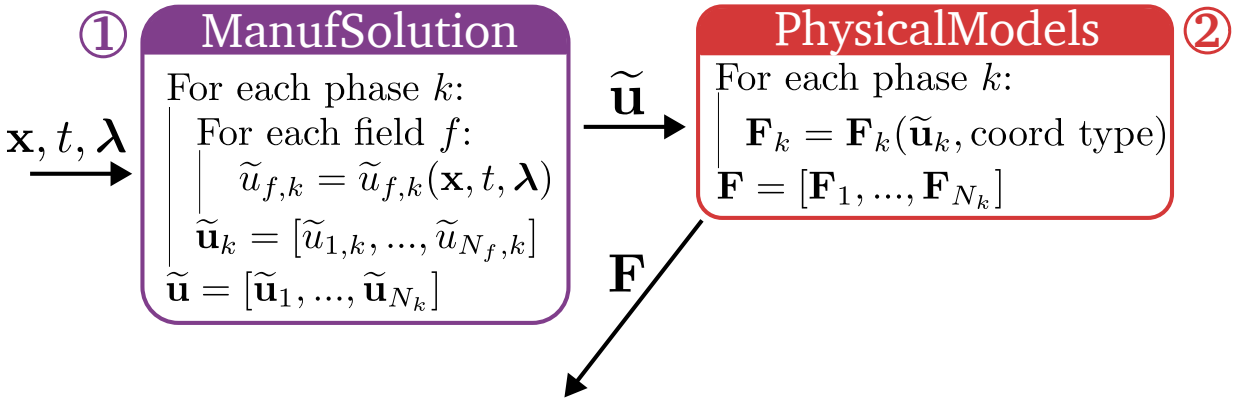
$$\widetilde{u}_2 = a_2 + b_2 \sin(\pi x/4)$$

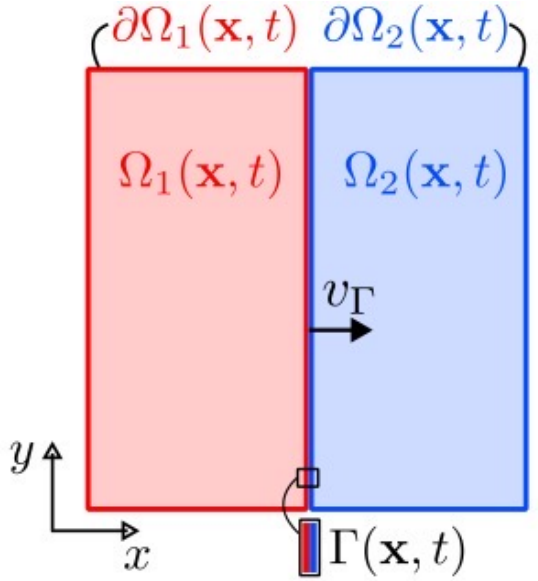
$$\boldsymbol{\lambda} = \{a_1, b_1, a_2, b_2\}$$

PyManufSol modules

[7] Henneaux (2023)

Step 2 – defining physical models





$$\mathbf{F}(\mathbf{u}(\mathbf{x}, t)) = 0$$

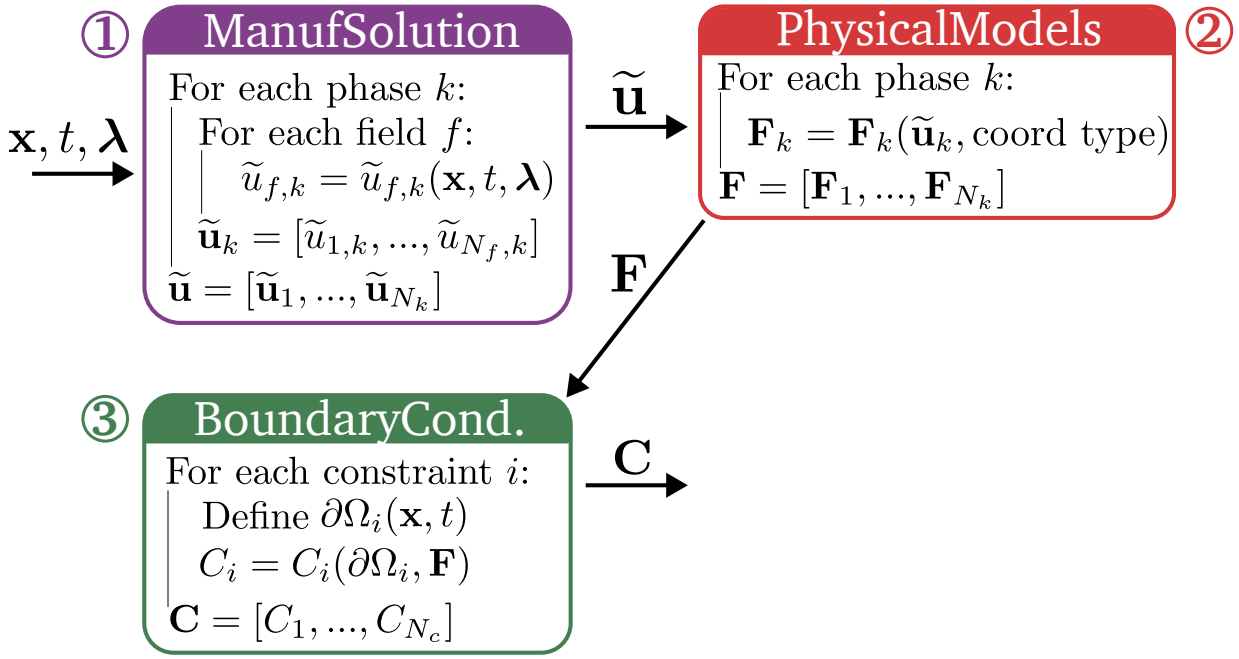
$$\mathbf{F}_1(u_1) = \frac{\partial u_1}{\partial t} + \nabla \cdot [c_{1,x} u_1, c_{1,y} u_1] - \nabla \cdot \left[k_{1,x} \frac{\partial u_1}{\partial x}, k_{1,y} \frac{\partial u_1}{\partial y} \right]$$

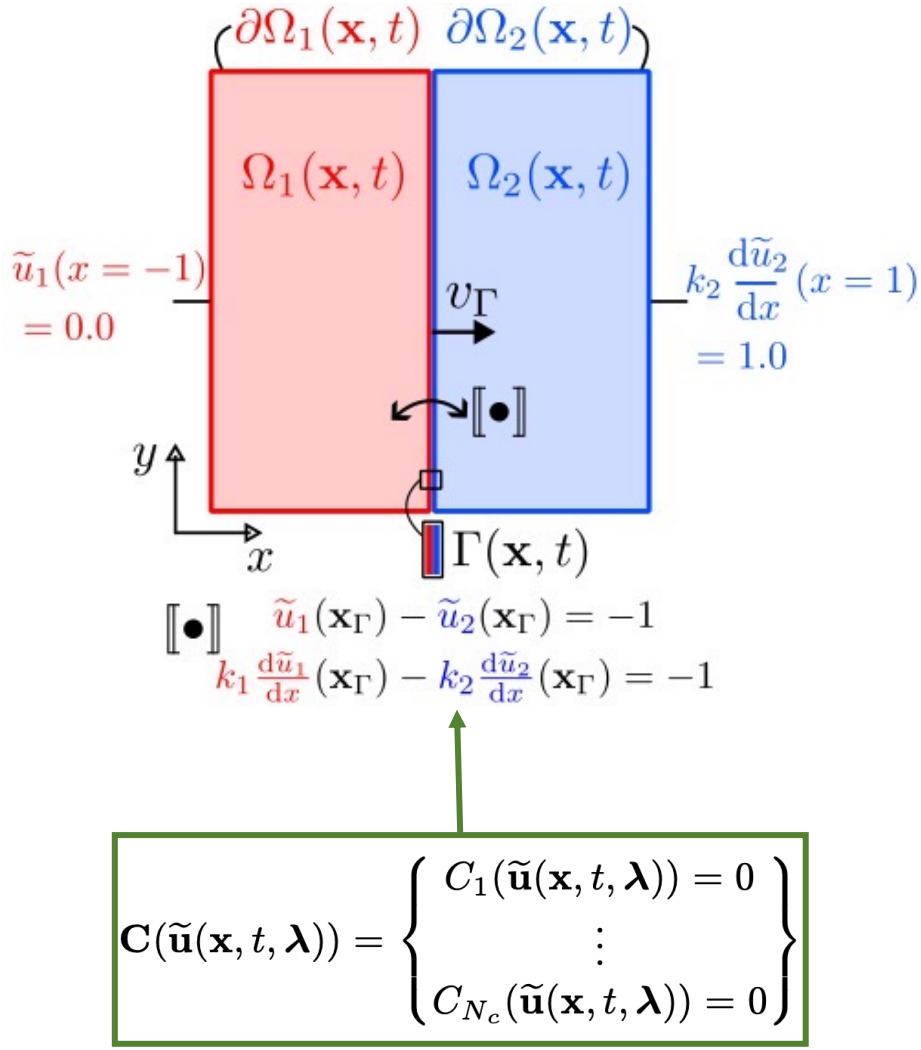
$$\mathbf{F}_2(u_2) = \frac{\partial u_2}{\partial t} + \nabla \cdot [c_{2,x} u_2, c_{2,y} u_2] - \nabla \cdot \left[k_{2,x} \frac{\partial u_2}{\partial x}, k_{2,y} \frac{\partial u_2}{\partial y} \right]$$

PyManufSol modules

[7] Henneaux (2023)

Step 3 – defining the set of constraints on MS

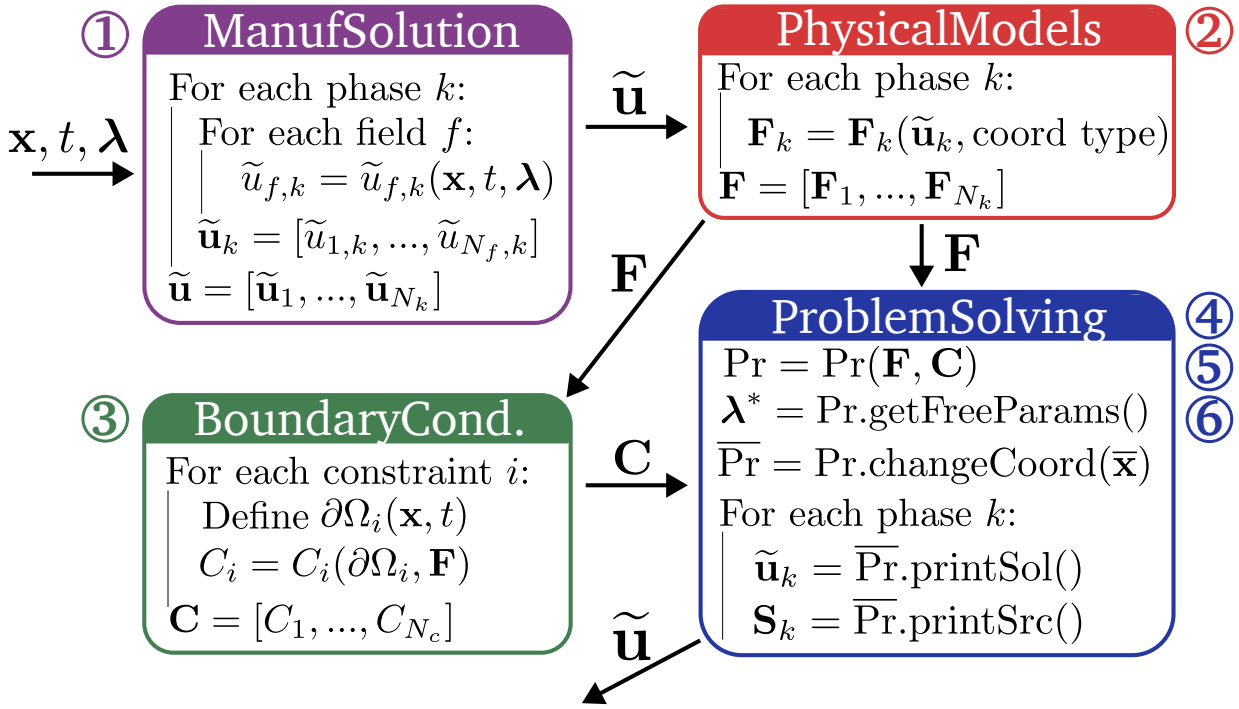




PyManufSol modules

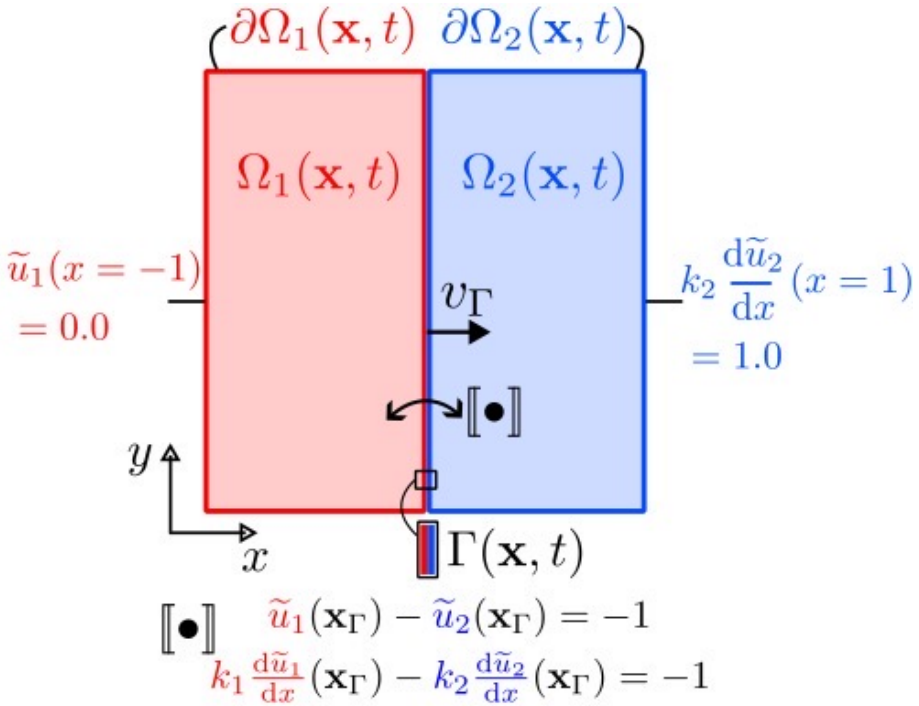
[7] Henneaux (2023)

Step 4 – computing the free parameters



PyManufSol modules

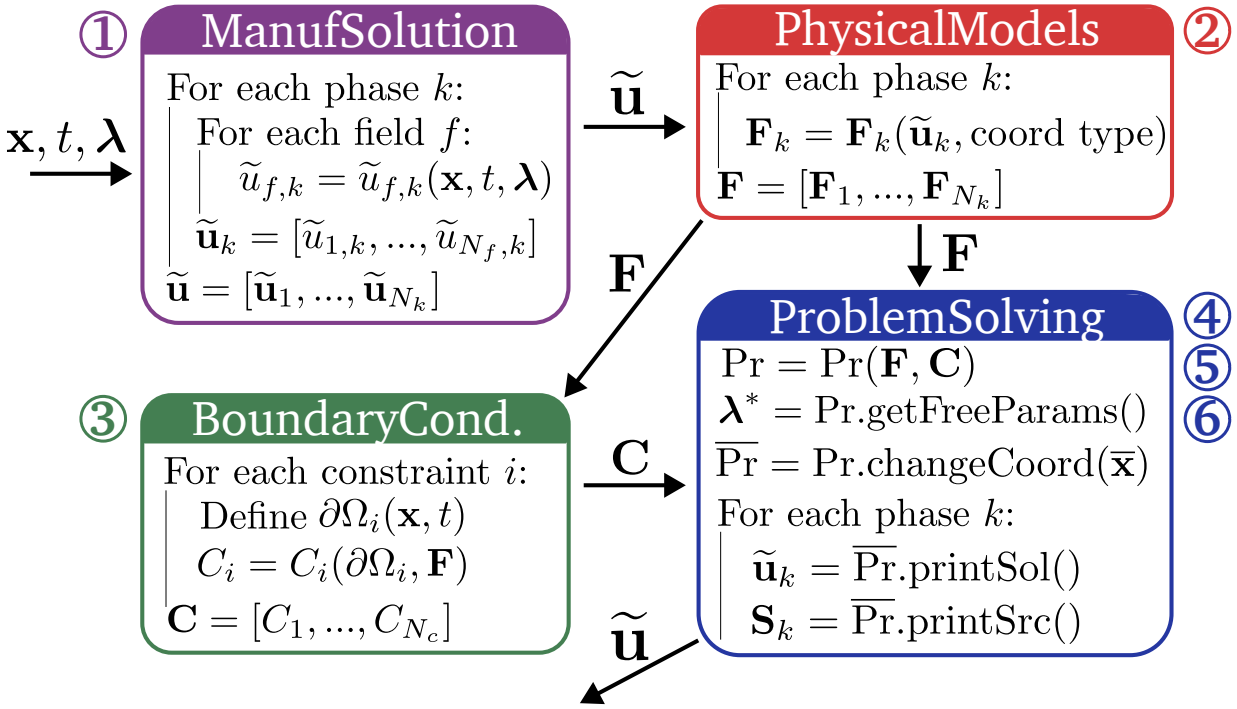
[7] Henneaux (2023)

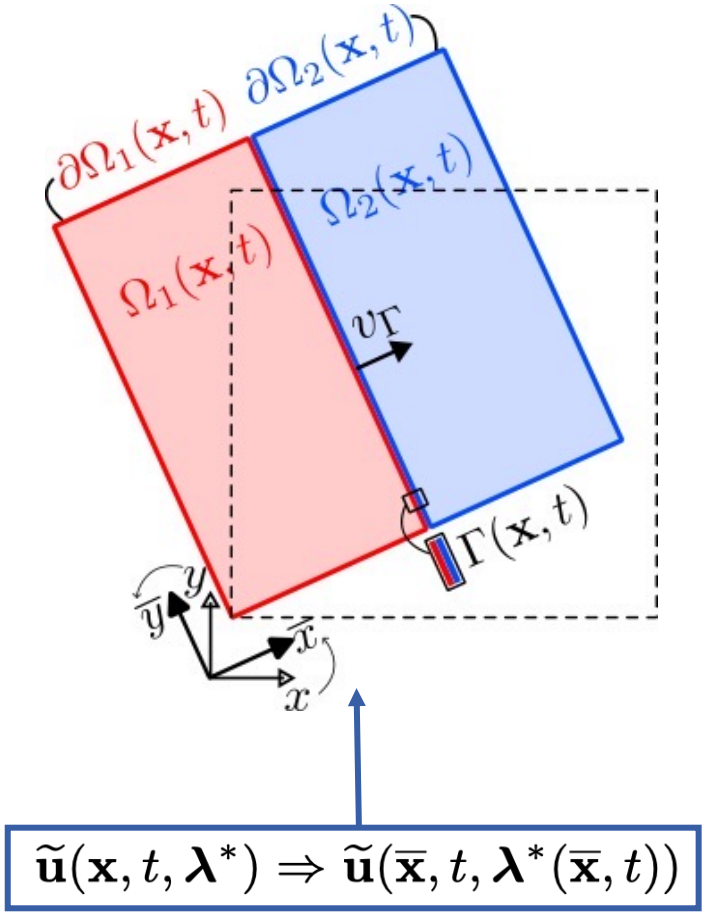


$$\boldsymbol{\lambda}^*(\mathbf{x}, t) \text{ s.t. } \mathbf{C}(\widetilde{\mathbf{u}}(\mathbf{x}, t, \boldsymbol{\lambda}^*)) = 0$$

$$\begin{cases} \widetilde{u}_1(x = -1, \boldsymbol{\lambda}^*) = 0 \\ k_2 \dfrac{\partial \widetilde{u}_2}{\partial x}(x = 1, \boldsymbol{\lambda}^*) = 1 \\ \widetilde{u}_1(x_\Gamma, \boldsymbol{\lambda}^*) - \widetilde{u}_2(x_\Gamma, \boldsymbol{\lambda}^*) = -1 \\ k_1 \dfrac{\partial \widetilde{u}_1}{\partial x}(x_\Gamma, \boldsymbol{\lambda}^*) - k_2 \dfrac{\partial \widetilde{u}_2}{\partial x}(x_\Gamma, \boldsymbol{\lambda}^*) = -1 \end{cases}$$

Step 5 – change of spatial coordinates system

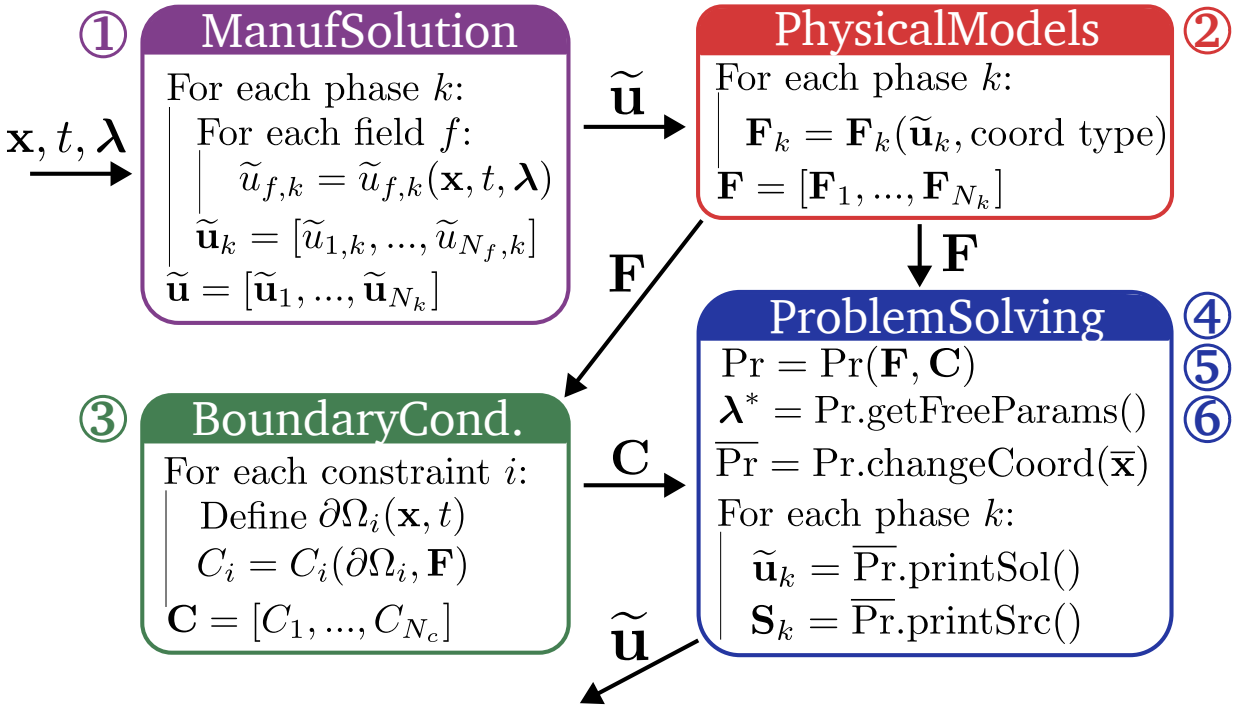




PyManufSol modules

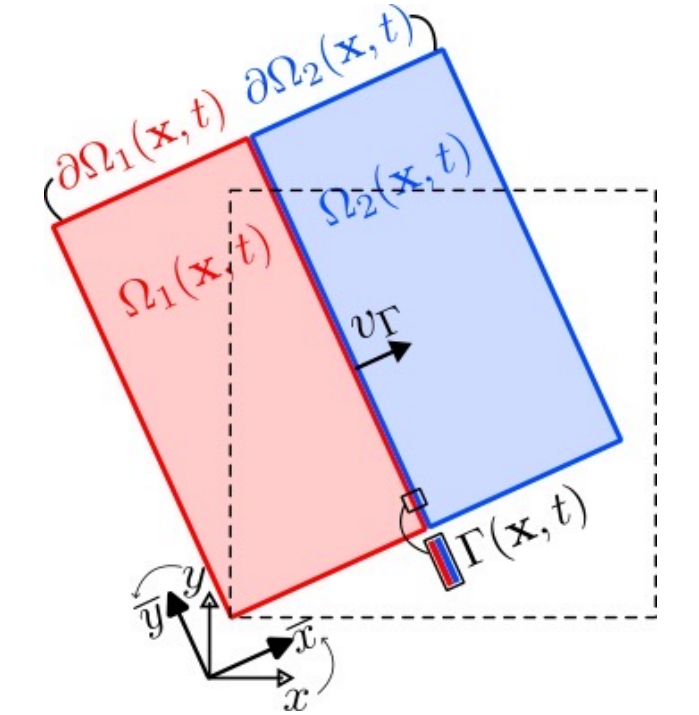
[7] Henneaux (2023)

Step 6 – source term computation



PyManufSol modules

[7] Henneaux (2023)

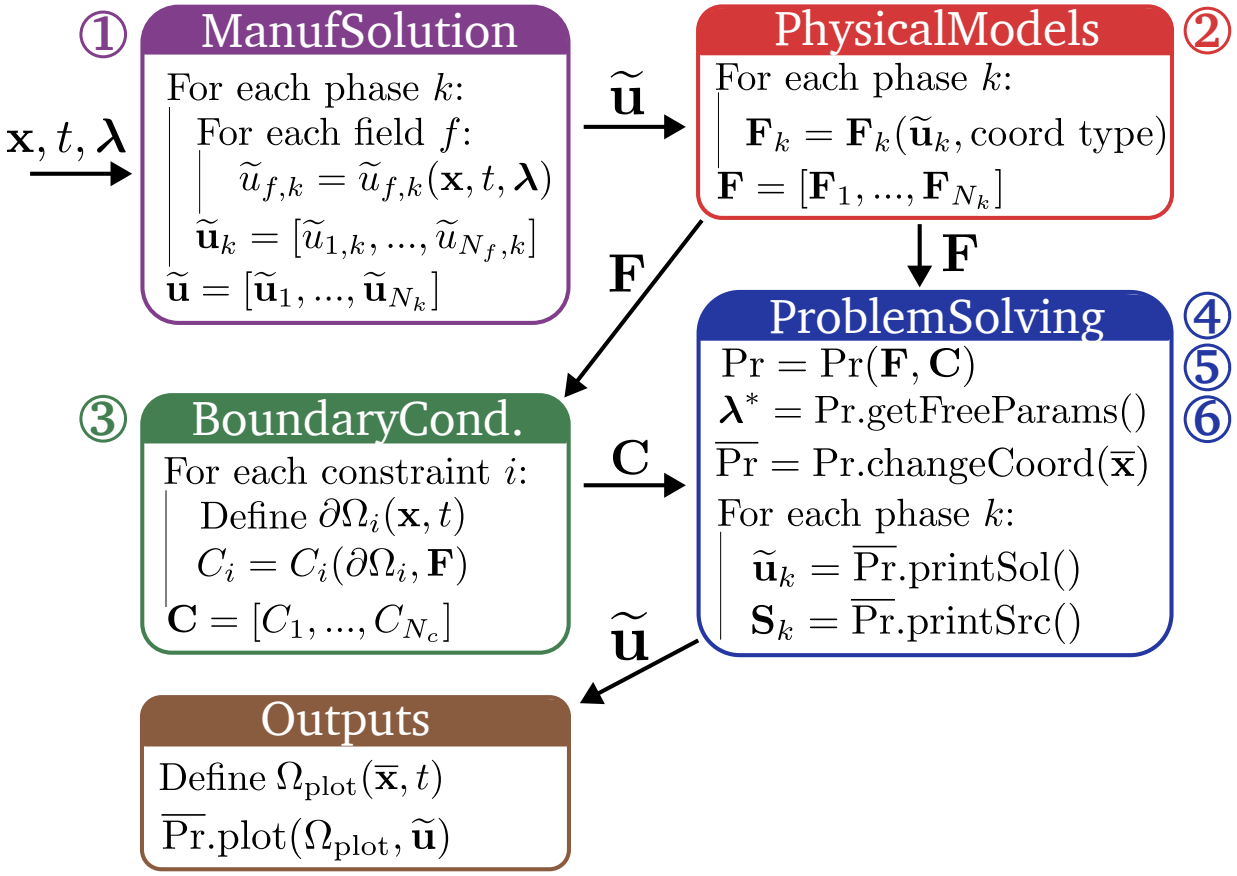


$$\mathbf{S}(\overline{\mathbf{x}}, t) = \mathbf{F}(\widetilde{\mathbf{u}}(\overline{\mathbf{x}}, t, \boldsymbol{\lambda}^*))$$

$$S_1(\overline{\mathbf{x}}, t) = F_1(\widetilde{u}_1(\overline{\mathbf{x}}, t, \boldsymbol{\lambda}^*))$$

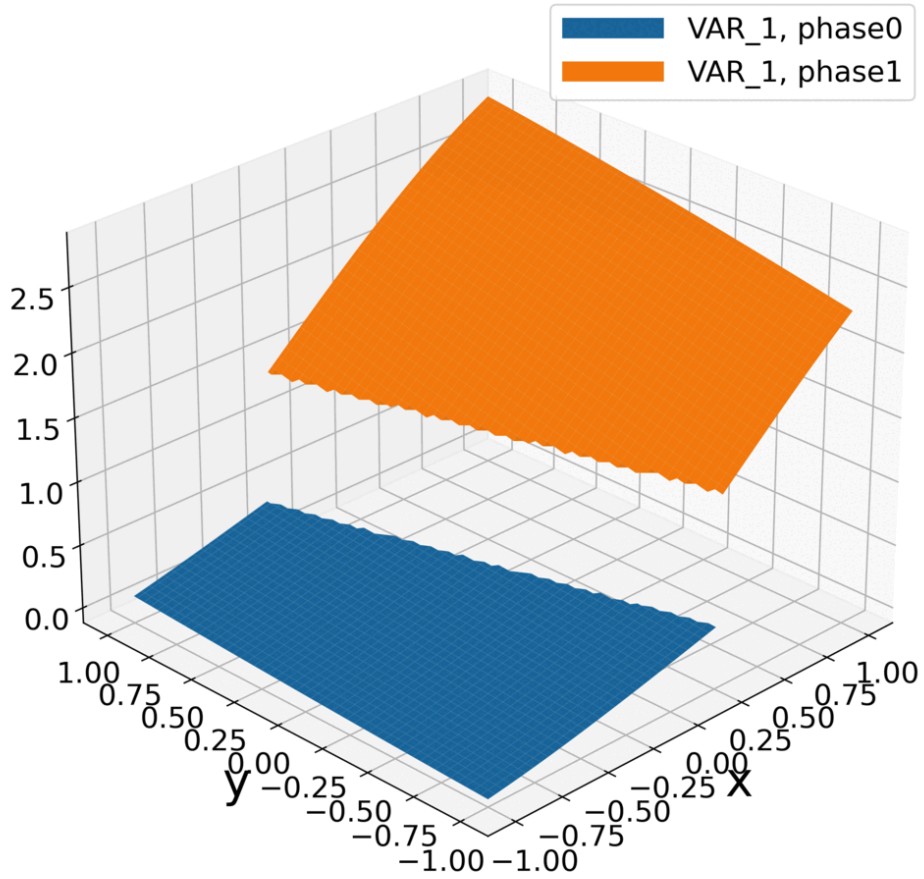
$$S_2(\overline{\mathbf{x}}, t) = F_2(\widetilde{u}_2(\overline{\mathbf{x}}, t, \boldsymbol{\lambda}^*))$$

Visualization of the MS (checking of constraints)

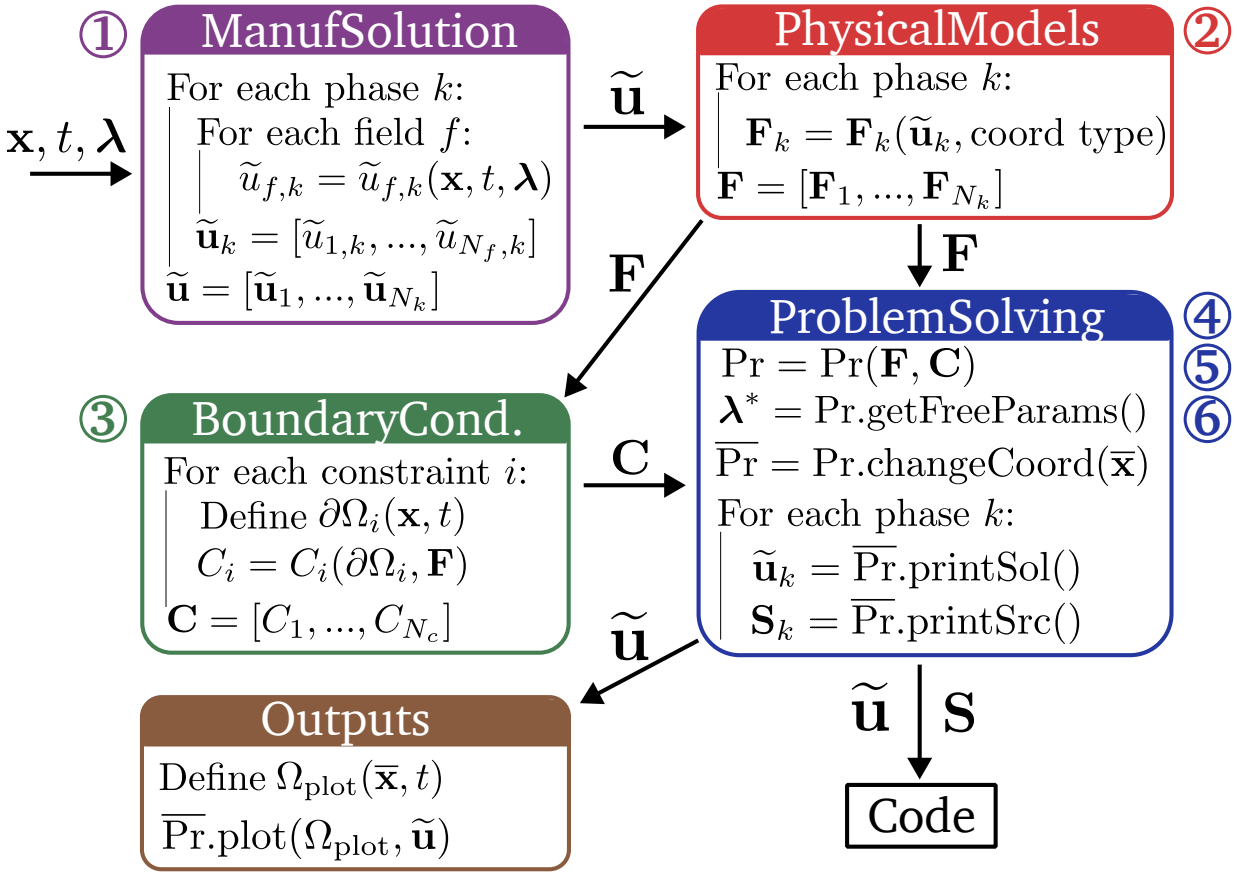


PyManufSol modules

[7] Henneaux (2023)



Reading by the code of the printed MS and source



PyManufSol modules

[7] Henneaux (2023)

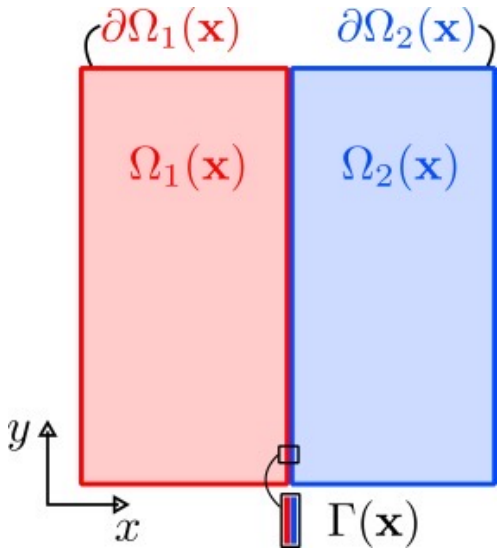
1. The Method of Manufactured Solution (MMS)
2. Building MS satisfying prescribed conditions with PyManufSol
3. Applications to two-phase Navier-Stokes equations
4. Conclusions and perspectives

Viscous compressible two-phase flow system

- Equations in the bulk:

$$\nabla \cdot \left(\underline{\boldsymbol{F}}_k^c(\mathbf{u}) - \underline{\boldsymbol{F}}_k^d(\mathbf{u}, \nabla \mathbf{u}_k) \right) = 0 \text{ in } \Omega_k$$

$$\mathbf{u}_k = \begin{bmatrix} \rho_k \\ \rho_k \mathbf{v}_k \\ \rho_k E_k \end{bmatrix} \quad \underline{\boldsymbol{F}}_k^c(\mathbf{u}_k) = \begin{bmatrix} \rho_k \mathbf{v}_k \\ \rho_k \mathbf{v}_k \otimes \mathbf{v}_k + p_k \underline{\boldsymbol{I}} \\ \rho_k \mathbf{v}_k H_k \end{bmatrix} \quad \underline{\boldsymbol{F}}_k^d(\mathbf{u}_k, \nabla \mathbf{u}_k) = \begin{bmatrix} 0 \\ \boldsymbol{\tau}_k \\ \underline{\boldsymbol{\tau}}_k \cdot \mathbf{v}_k - \mathbf{q}_k \end{bmatrix}$$



- Jumps conditions at the interface:

$$\text{Non-linear Robin jumps} \begin{cases} \dot{m}_\Gamma \, [\![v_n]\!] + [\![p]\!] - \Delta p = [\![\tau_{nn}]\!] \\ \dot{m}_\Gamma \, [\![v_t]\!] - \Delta \tau_{nt} = [\![\tau_{nt}]\!] \\ \dot{m}_\Gamma \, [\![E]\!] + [\![p \, v_n]\!] - \Delta p \, v_{n,\Gamma} - \Delta \tau_{nt} \, v_{t,\Gamma} - \Delta q = [\![\tau_{nn} \, v_n]\!] + [\![\tau_{nt} \, v_t]\!] - [\![q_n]\!] \end{cases}$$

$$[\![\dot{m}_\Gamma]\!] = 0$$

$$\text{Dirichlet jumps} \begin{cases} [\![v_t]\!] = \Delta v_t \\ [\![T]\!] = \Delta T \end{cases}$$

$$\dot{m}_\Gamma := \rho_k (v_{n,k} - v_{n,\Gamma}) = \dot{m}_\Gamma^{\text{model}}$$

Discretized by high-order sharp interface extended discontinuous Galerkin method [8]

[8] Henneaux et al. (2020)

Inclined interface test case – solution manufacturing

$$\widetilde{p}_1(y) = -0.11\,\frac{\cos(\pi\,y/2)}{\cos(\pi/6)} + 1.21$$

$$\boldsymbol{\lambda} = \{\lambda_{p,1}, \lambda_{v_x,1}, \lambda_{v_x,2}, \lambda_{v_y,1}, \lambda_{v_y,2}, \lambda_{T,1}, \lambda_{T,2}\}$$

$$\widetilde{v}_{x,1}(y) = 1.0\cos(0.75\,\pi\,y) + 2.45$$

$$\widetilde{v}_{y,1}(y) = 1.0\cos(0.75\,\pi\,y) + 2.45$$

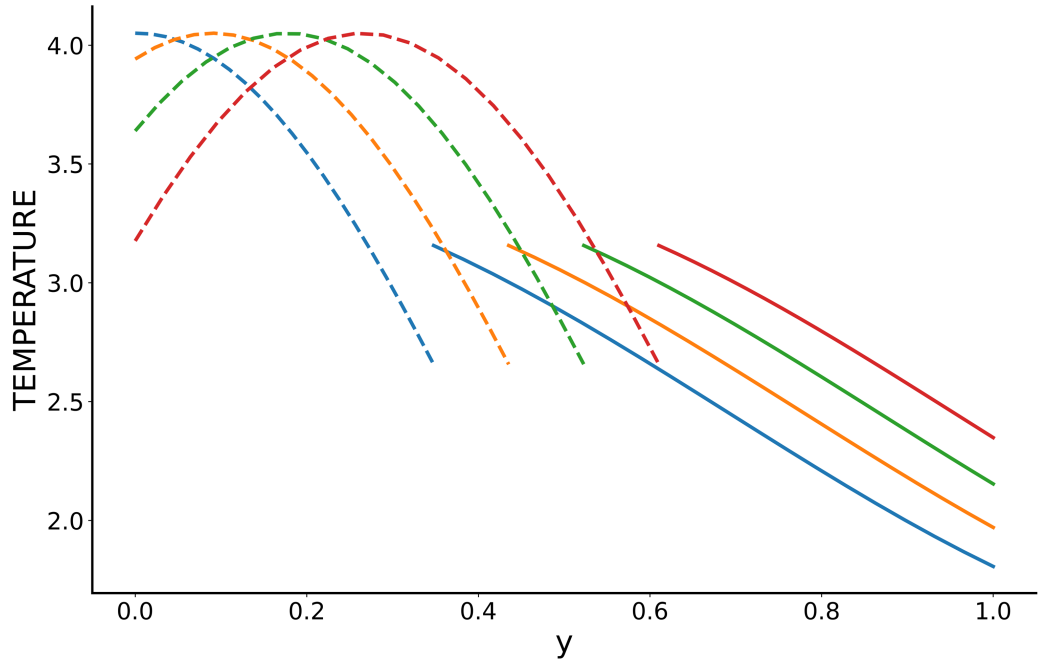
$$\widetilde{T}_1(y) = 1.0\cos(0.75\,\pi\,y) + 2.45$$

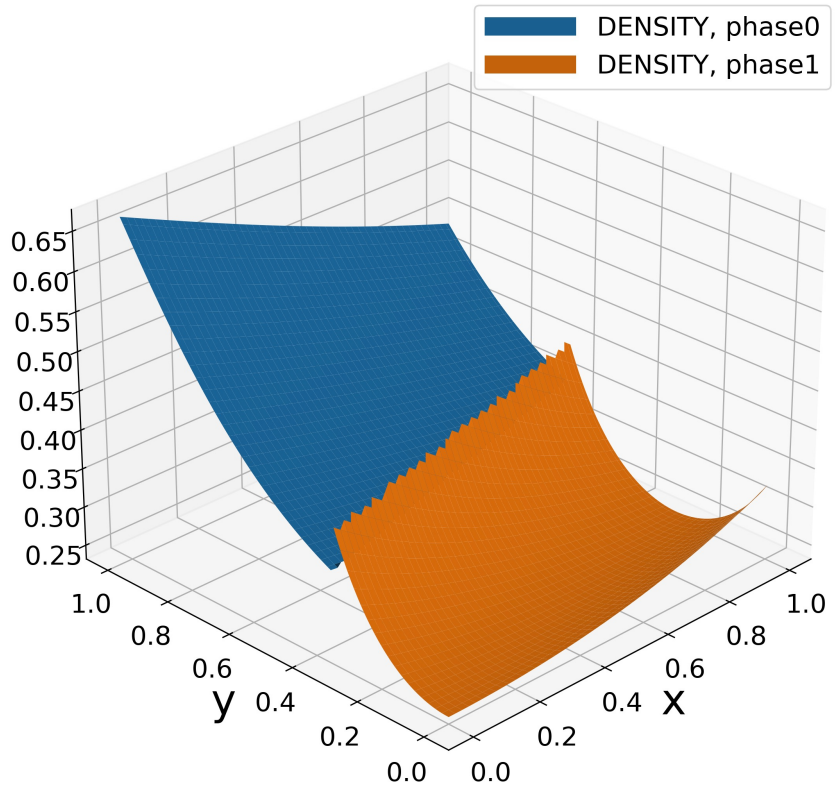
$$\widetilde{p}_2(y) = \lambda_{p,1} - 2.36178756984203\,\cos(y) + 3.53178756984203$$

$$\widetilde{v}_{x,2}(y) = \lambda_{v_x,1}\cos(5\,\pi\,y/4) + 2.25 + \lambda_{v_x,2}$$

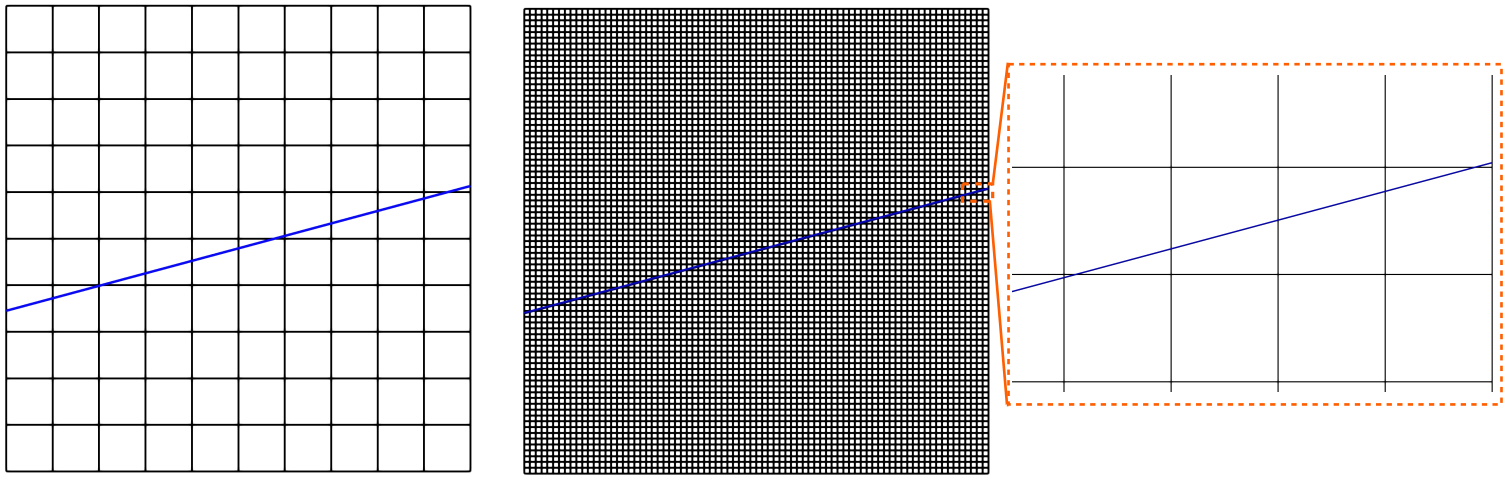
$$\widetilde{v}_{y,2}(y) = \lambda_{v_y,1}\cos(5\,\pi\,y/4) + 2.25 + \lambda_{v_y,2}$$

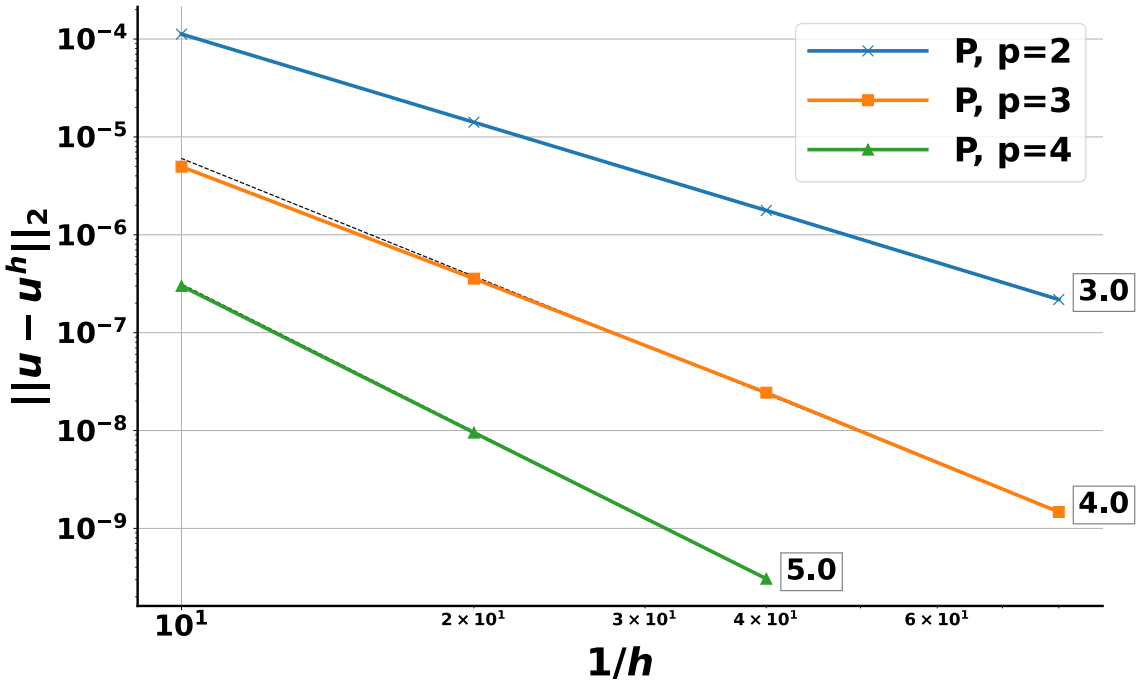
$$\widetilde{T}_2(y) = \lambda_{T,1}\cos(5\,\pi\,y/4) + 2.25 + \lambda_{T,2}$$





Inclined interface test case - order of accuracy study

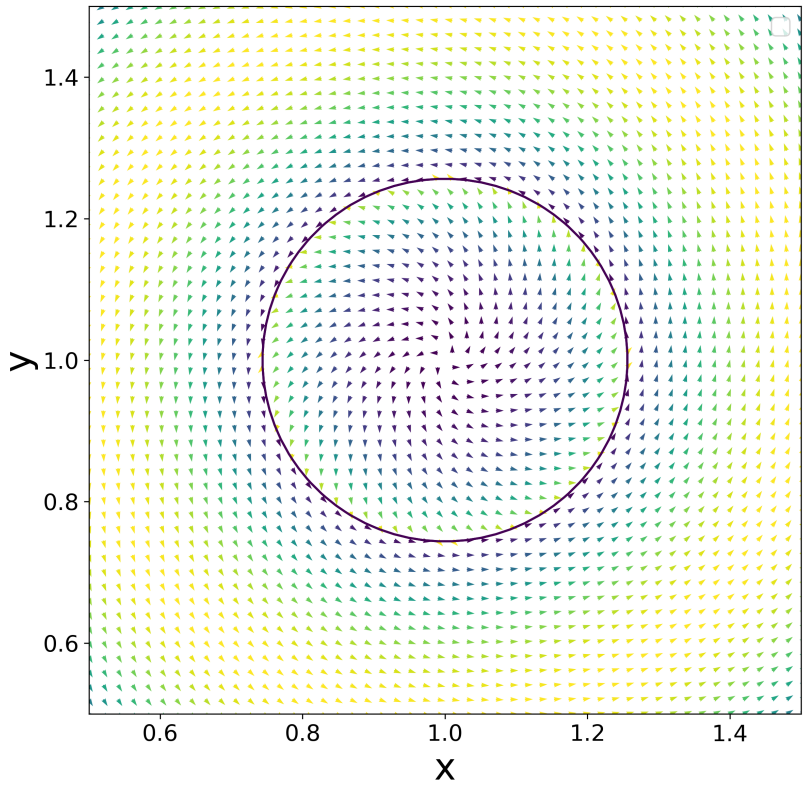


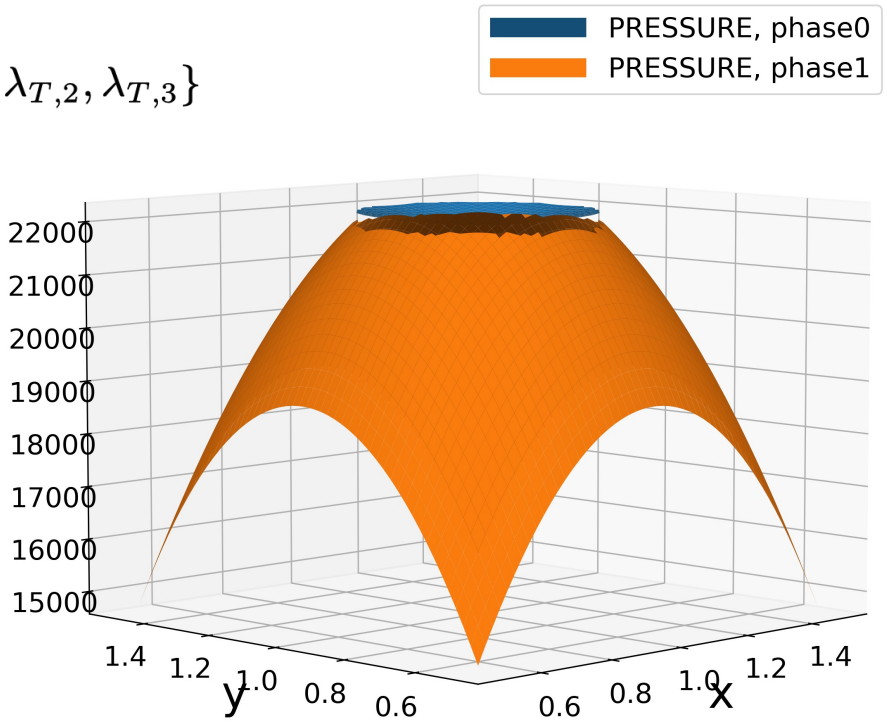


Circular interface test case – solution manufacturing

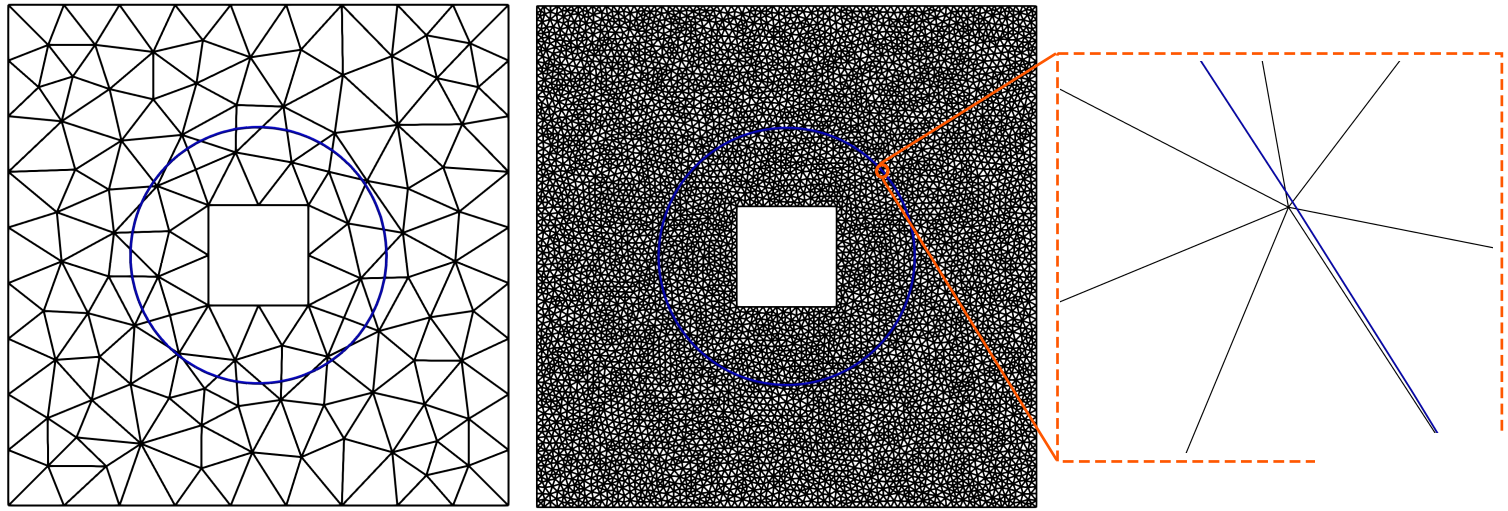
$$\widetilde{p}_1(r) = 3380.84473527662\ r^2 + 21978.0\ ,$$
$$\widetilde{v}_{r,1}(r) = 59.4303622988639\ r^3\ ,$$
$$\widetilde{v}_{\theta,1}(r) = 59.4303622988639\ r^3\ ,$$
$$\widetilde{T}_1(r) = 1942.94288270104\ - 115.466222827242\ r^3\ ,$$
$$\widetilde{p}_2(r) = \lambda_{p,1} - 16751.9333729923\ r^2 + 23300.0\ ,$$
$$\widetilde{v}_{r,2}(r) = \lambda_{v_r,1} + \lambda_{v_r,2}\, r + \lambda_{v_r,3}\, r^2\ ,$$
$$\widetilde{v}_{\theta,2}(r) = \lambda_{v_\theta,1} + \lambda_{v_\theta,2}\, r + \lambda_{v_\theta,3}\, r^2\ ,$$
$$\widetilde{T}_2(r) = \lambda_{T,1} + \lambda_{T,2}\, r + \lambda_{T,3}\, r^2\ ,$$

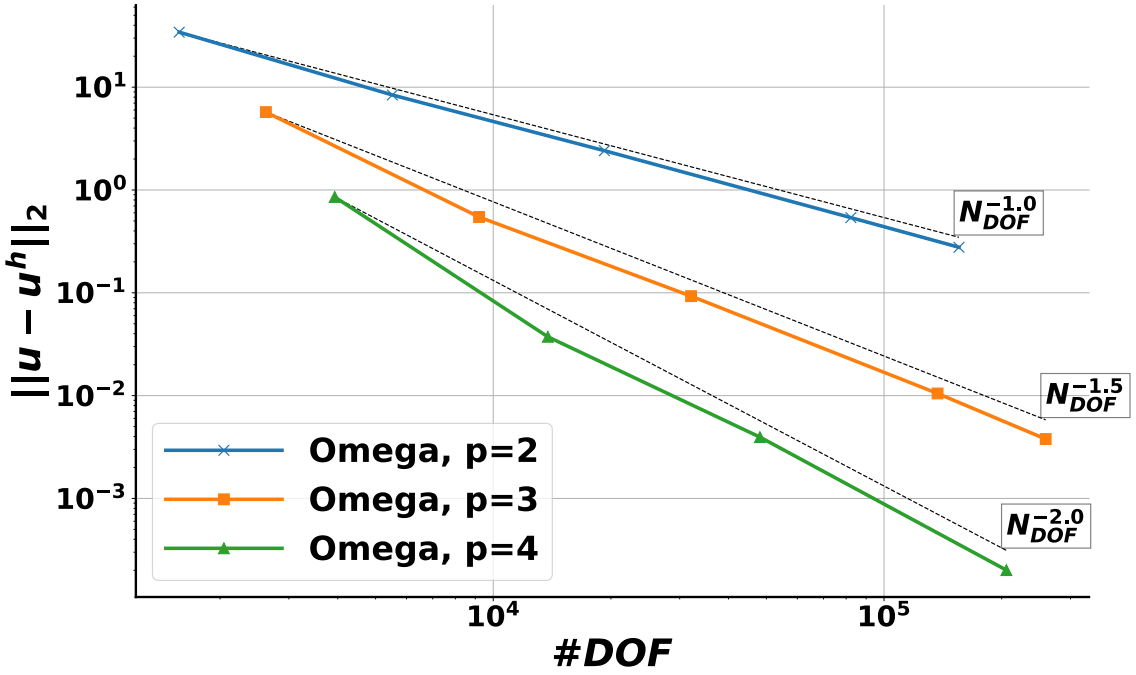
$$\boldsymbol{\lambda} = \{\lambda_{p,1}, \lambda_{v_r,1}, \lambda_{v_r,2}, \lambda_{v_r,3}, \lambda_{v_\theta,1}, \lambda_{v_\theta,2}, \lambda_{v_\theta,3}, \lambda_{T,1}, \lambda_{T,2}, \lambda_{T,3}\}$$





Circular interface test case - order of accuracy study





1. The Method of Manufactured Solution (MMS)
2. Building MS satisfying prescribed conditions with PyManufSol
3. Applications to two-phase Navier-Stokes equations
4. Conclusions and perspectives

Conclusions

- Development of a general framework using symbolic manipulations to generate manufactured solutions for code verification
- Flexibility to design a wide range of complicated solutions subjected to user-defined constraints in order to verify critical parts of the code
- Application on challenging viscous compressible two-phase systems

- From our experience, using this approach allowed us to detect and correct different kinds of errors in our CFD solver (bugs, model implementation) as well as to identify necessary discretization formulation improvements to reach optimal order of convergence

Perspectives

- Overcoming the limitations on the boundaries geometry
  - Limited to straight interfaces / circles
  - Could investigate regression approach [9] or the use of multiplicative functions encompassing the geometry [3]

- Overcoming the limitations on the treatment of Neumann-type constraints
  - Free parameters shouldn't depend on spatial coordinates, otherwize need to solve a DAE system
  - Could investigate the approaches in [10,6] where they manufacture some parameters contained in the bulk equations (instead of the solution) to satisfy some conditions

[3] Bond et al. (2007)
[6] Freno et al. (2022)
[9] Grier et al. (2015)
[10] Brglez et al. (2014)

Thank you for your attention and to

Some references if you're interested

[1] Navah, Farshad, and Siva Nadarajah. "A comprehensive high-order solver verification methodology for free fluid flows." *Aerospace Science and Technology* 80 (2018): 101-126.

[2] Steinberg, Stanly, and Patrick J. Roache. "Symbolic manipulation and computational fluid dynamics." *Journal of Computational Physics* 57.2 (1985): 251-284.

[3] Bond, Ryan B., et al. "Manufactured solution for computational fluid dynamics boundary condition verification." *AIAA journal* 45.9 (2007): 2224-2236.

[4] Choudhary, Aniruddha, et al. "Code verification of boundary conditions for compressible and incompressible computational fluid dynamics codes." *Computers & Fluids* 126 (2016): 153-169.

[5] Brady, P. T., Marcus Herrmann, and J. M. Lopez. "Code verification for finite volume multiphase scalar equations using the method of manufactured solutions." *Journal of Computational Physics* 231.7 (2012): 2924-2944.

[6] Freno, Brian A., et al. "Nonintrusive manufactured solutions for non-decomposing ablation in two dimensions." *Journal of Computational Physics* 463 (2022): 111237.

[7] Henneaux, D. (2023). PyManufSol (Version 1.0.0) [Computer software]. https://github.com/henneauxd/PyManufSol

[8] Henneaux, David, et al. "Extended discontinuous Galerkin method for solving gas-liquid compressible flows with phase transition." *AIAA AVIATION 2020 FORUM*. 2020.

[9] Grier, Benjamin, et al. "Discontinuous solutions using the method of manufactured solutions on finite volume solvers." *AIAA Journal* 53.8 (2015): 2369-2378.

[10] Brglez, Špela. "Code verification for governing equations with arbitrary functions using adjusted method of manufactured solutions." *Engineering with Computers* 30.4 (2014): 669-678.

The extended discontinuous Galerkin method