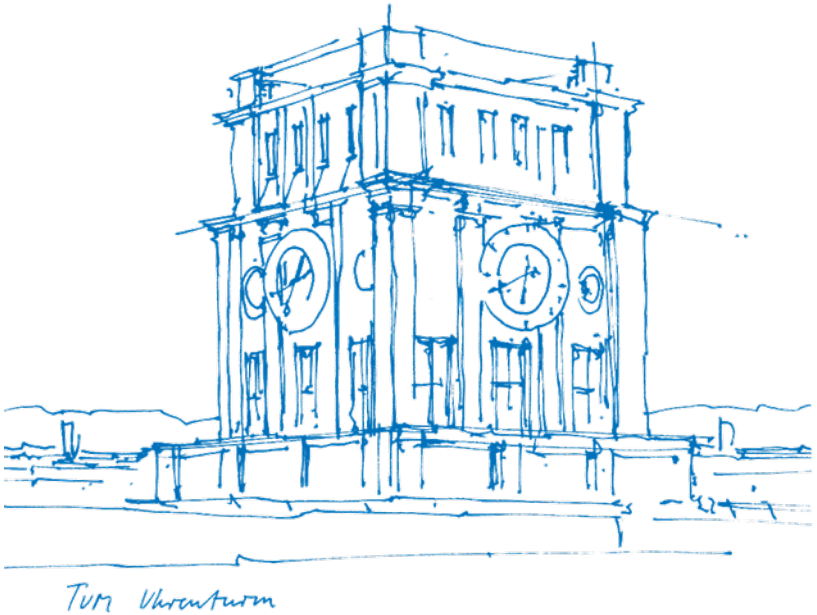
High-fidelity partitioned multiphysics with preCICE and open-source solvers

Benjamin Rodenberg, B. Uekermann, H.J. Bungartz

PDESoft 2024, Cambridge, UK, July 3, 2024



Technical University of Munich, School for Computation, Information and Technology

Outline

preCICE overview

`tutorials/perpendicular-flap` (demo + model)

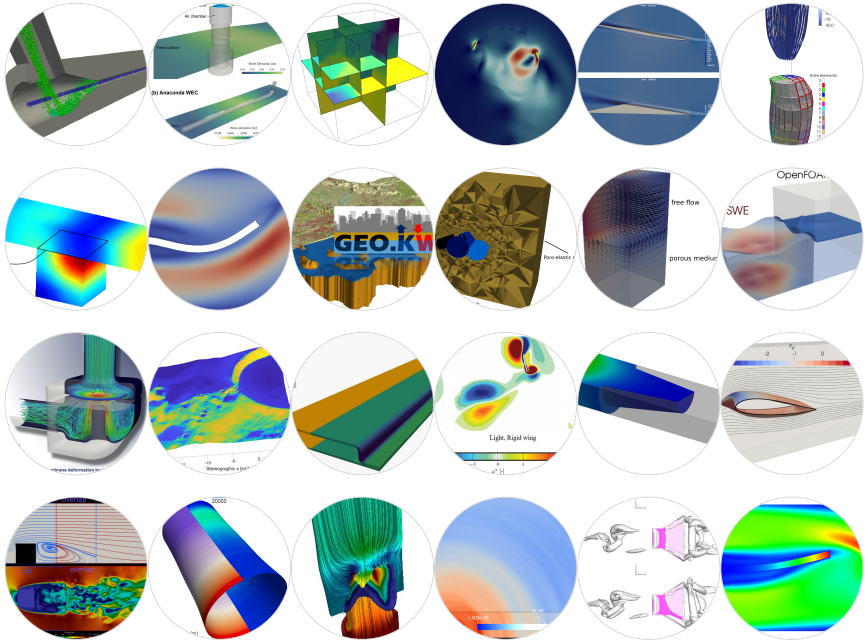
preCICE v3: Time interpolation

`tutorials/oscillator` (validation)

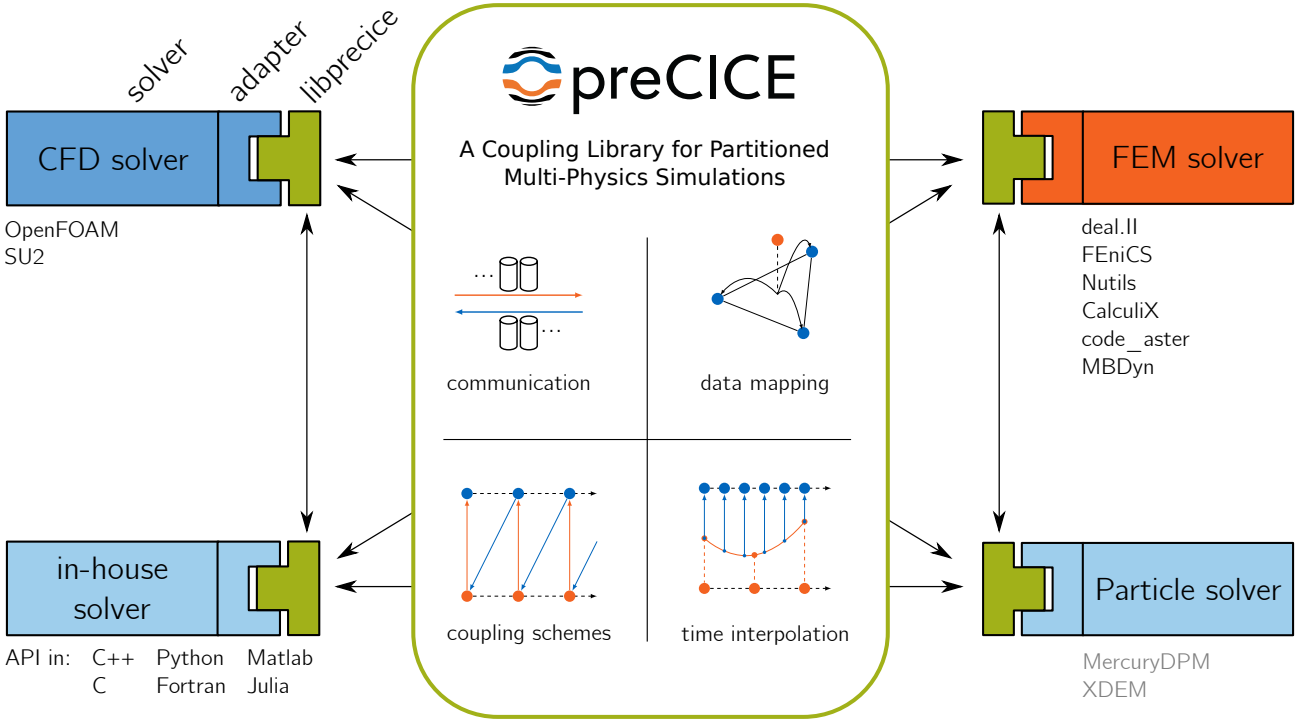
`tutorials/perpendicular-flap` (application?)

Conclusion

precice.org/community-projects



preCICE overview



preCICE

Tutorials

- Introduction
- Basic cases
- All tutorials
  - ASTE turbine
  - Breaking dam with flexible pillar 2D
  - Channel transport
  - Elastic tube 1D
  - Elastic tube 3D
  - Flow around controlled moving cylinder
  - Flow over a heated plate
  - Heat exchanger
  - Heat exchanger simplified
  - Multiple perpendicular flaps
  - Oscillator
  - Oscillator overlap
  - Partitioned elastic beam
  - Partitioned flow
  - Partitioned heat conduction
  - Perpendicular flap
  - Two-scale heat conduction
  - Turek-Hron FSI3
  - Volume-coupled diffusion
  - Volume-coupled flow

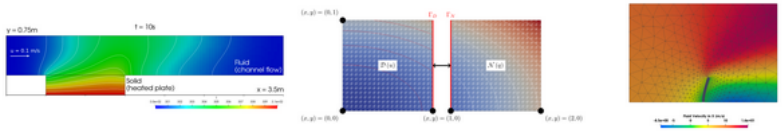
You can find all tutorial case files in the tutorials repository. Get the latest release, or clone the Git repository to easily update them in the future:

```
git clone --branch=master --depth 1 https://github.com/precice/tutorials.git
```

Basic cases

We recommend that you start from one of the following cases, which you can quickly run on your laptop:

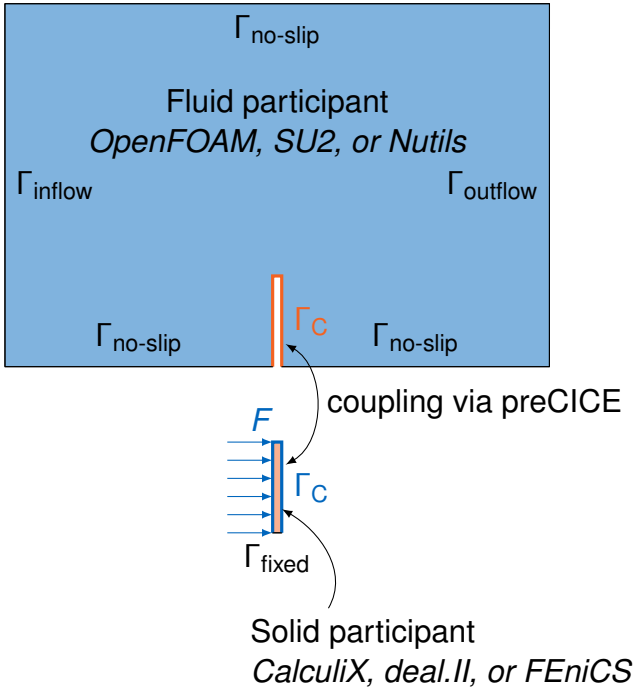
- Flow over a heated plate: A **conjugate heat transfer** scenario. Try OpenFOAM, FEniCS, or Nutils.
- Partitioned heat conduction: The mathematician's dream: split the **heat equation** in two and glue it again. Pick your Dirichlet and Neumann solvers among FEniCS, Nutils, and OpenFOAM.
- Flow in a channel with an elastic perpendicular flap: A **fluid-structure interaction** scenario. Feel free to combine different solvers, among OpenFOAM, SU2, deal.II, FEniCS, Nutils, CalculiX, and DUNE.



`tutorials/perpendicular-flap` (demo + model)

DEMO TIME

`tutorials/perpendicular-flap` (demo + model)



Divide

- OpenFOAM $\neq$ FEniCS
- Dirichlet-Neumann $=$ black box

Conquer

- Fluid: $\mathscr{F}(d) = f$
- Solid: $\mathscr{S}(f) = d$

Boundary response maps

(= Poincaré-Steklov operator)

Combine

- $\mathscr{F}(\mathscr{S}(f^k)) = \tilde{f}^k$
- $\tilde{f}^k \xrightarrow{\mathscr{A}} f^{k+1}$

Picard iteration + acceleration

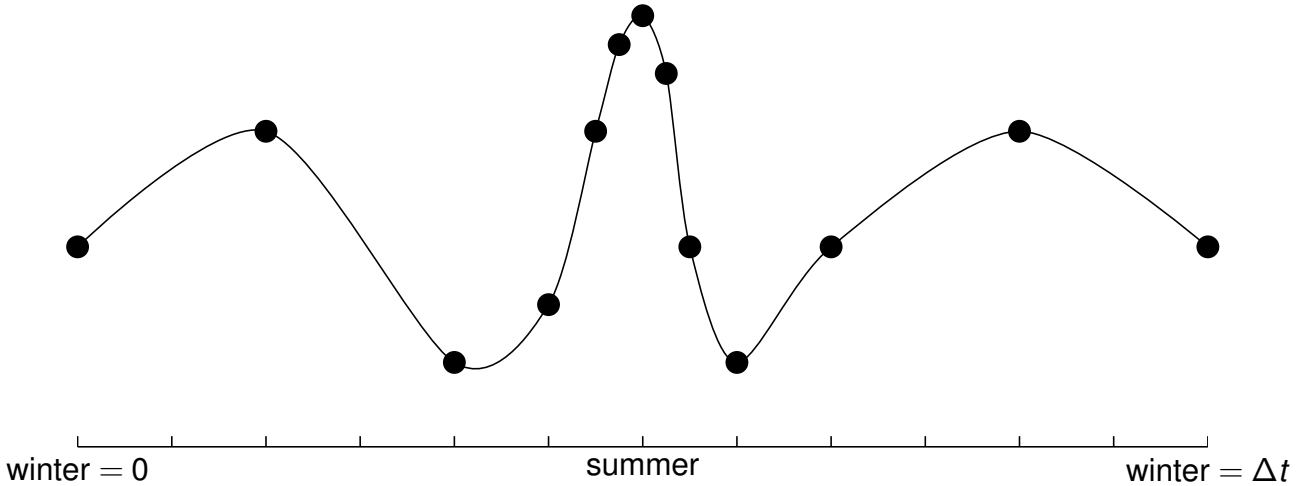
`tutorials/perpendicular-flap` (demo + model)

DEMO TIME

preCICE v3: Time interpolation

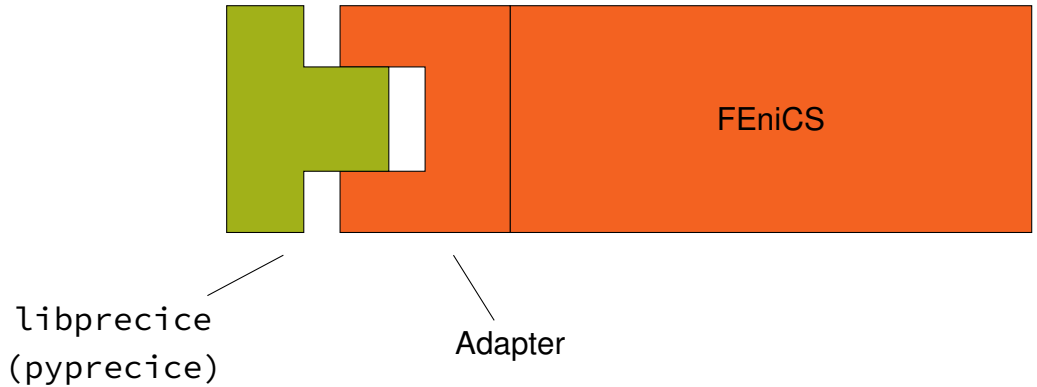
preCICE Workshop 2023: Two "ice sheet talks"[1]

- Yannic Fischler: *A preCICE-interface for the ice-sheet and sea-level system model*
- Daniel Abele: *Coupling an ice sheet model with satellite image based simulation of calving fronts*



[1] https://precice.org/precice-workshop-2023.html

Requirements for time interpolation in preCICE



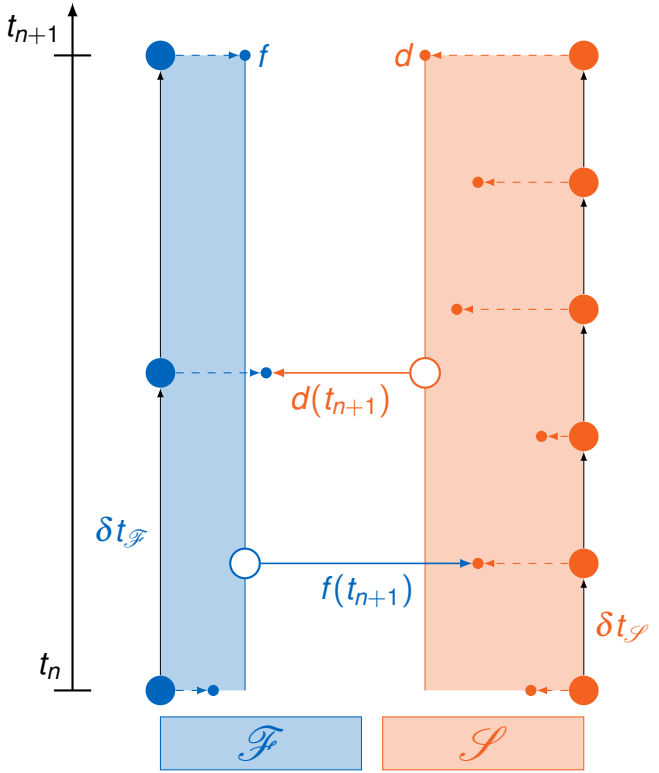
Goal: Black-box + higher-order + multirate

- Numerics: Accuracy, convergence order, energy conservation
- API Design: Simple user interface!
- Move as much multirate/higher-order/interpolation logic *inside* preCICE as possible
- preCICE v3 + new API was just released[1]

[1] https://github.com/precice/precice/releases/tag/v3.0.0

preCICE v2: single-value coupling

```
<data:vector name="Force" />
<data:vector name="Displ" />
...
<coupling-scheme:serial-implicit>
  <participants first="Fluid" second="Solid" />
  <exchange
    data="Force"
    from="Fluid"
    to="Solid" />
  <exchange
    data="Displ"
    from="Solid"
    to="Fluid" />
  ...
</coupling-scheme:serial-implicit>
```

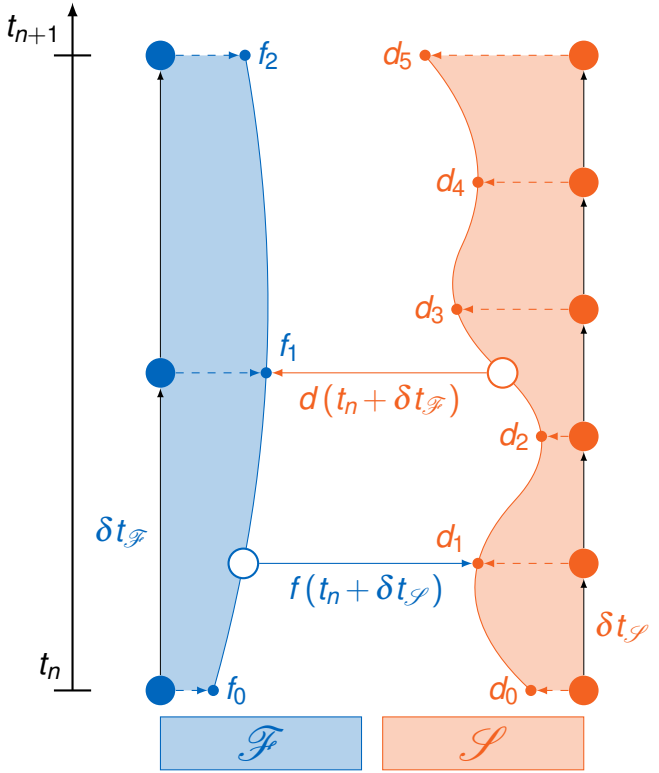


preCICE v3: waveform iteration

$$\mathscr{F}(\mathscr{S}(f^k)) = \tilde{f}^k$$

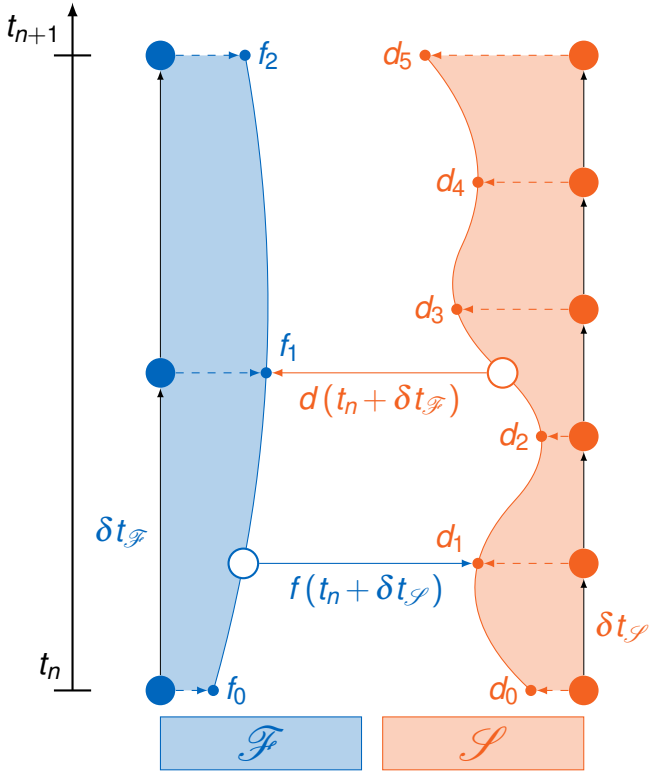
upgrade to preCICE v3

$$\mathscr{F}(\mathscr{S}(f(t)^k)) = \tilde{f}(t)^k$$



preCICE v3: waveform iteration

```
<data:vector name="Force" waveform-degree="2" />
<data:vector name="Displ" waveform-degree="3" />
...
<coupling-scheme:serial-implicit>
  <participants first="Fluid" second="Solid" />
  <exchange
    data="Force"
    from="Fluid"
    to="Solid"
    substeps="true" />
  <exchange
    data="Displ"
    from="Solid"
    to="Fluid"
    substeps="true" />
  ...
</coupling-scheme:serial-implicit>
```



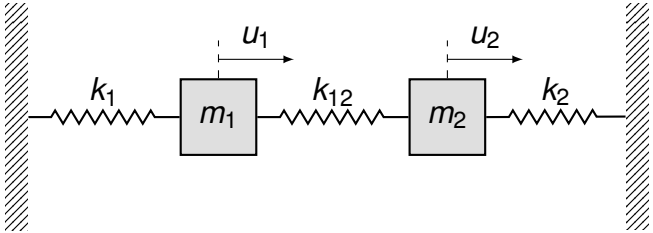
RK4 in code (simplified)

```python
# determine time step size δt for this time step
dt = participant.get_max_time_step_size()

# compute stages k_i and evaluate waveform at times c_i δt
k1 = A.dot(u)                    + participant.read_data("Displ", 0.0 * dt)
k2 = A.dot(u + k1 * 0.5 * dt) + participant.read_data("Displ", 0.5 * dt)
k3 = A.dot(u + k2 * 0.5 * dt) + participant.read_data("Displ", 0.5 * dt)
k4 = A.dot(u + k3 * 1.0 * dt) + participant.read_data("Displ", 1.0 * dt)
# assemble new solution
u_new = u + dt / 6 * (k1 + 2 * k2 + 2 * k3 + k4)

# do writing
force = extract_force(u_new)
participant.write_data("Force", force)
# end time step of size δt
participant.advance(dt)
```

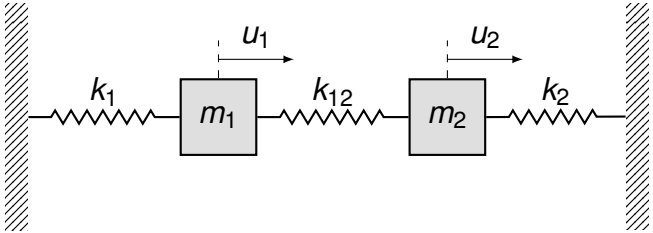
API with RK4 and multirate

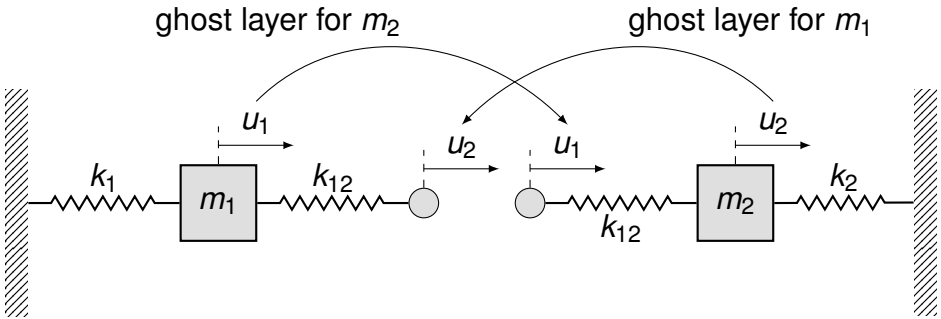
```python
# example: FSI coupling, perspective of fluid solver 𝓕:
# ...
participant.initialize()

while participant.is_coupling_ongoing():
  # store checkpoint, if needed
  precice_dt = participant.get_max_time_step_size()  # until end of window
  solver_dt = time_stepper.get_max_dt()  # stability, adaptivity
  dt = np.min([precice_dt, solver_dt])  # actual time step size δt
  # time_stepper represents s-stage RK scheme
  ts = time_stepper.rhs_eval_points(dt)  # t_n + c_1, t_n + c_2 ..., t_n + c_s
  displacements = [participant.read_data(t) for t in ts]  # d(t_n + c_1), d(t_n + c_2) ..., d(t_n + c_s)
  forces = time_stepper.do_step(displacements, dt)  # f_{n+1} = f_n + δt Σ_{i=1}^{s} b_i k_i
  participant.write_data(forces)
  participant.advance(dt)
  # read checkpoint, if needed

participant.finalize()
```

`tutorials/oscillator` (validation)

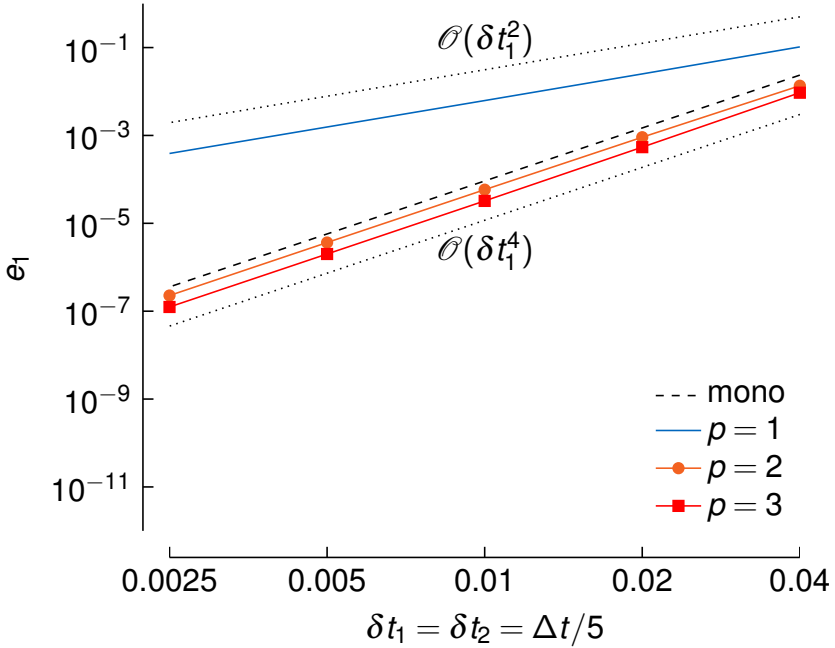


`tutorials/oscillator` (validation)

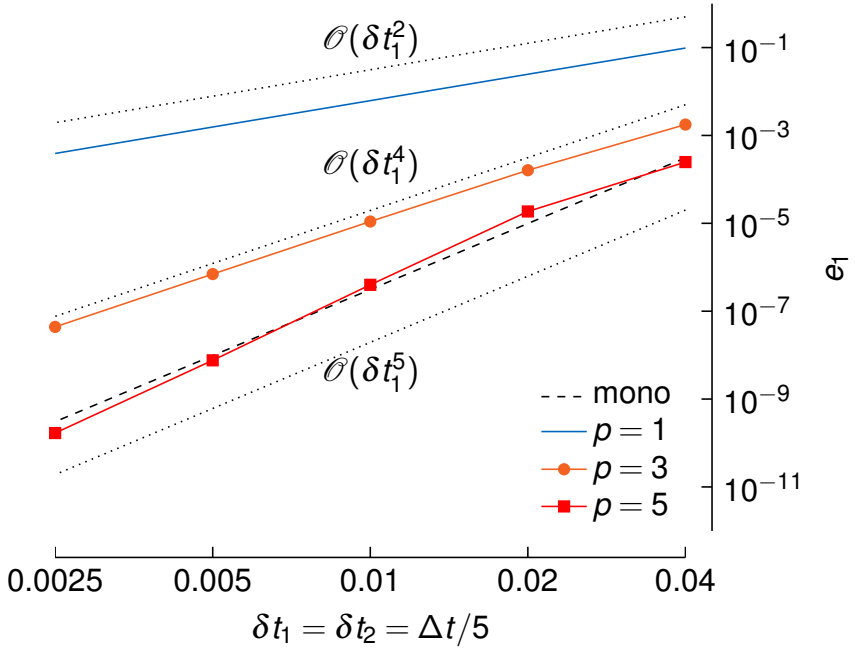




`tutorials/oscillator` (validation)

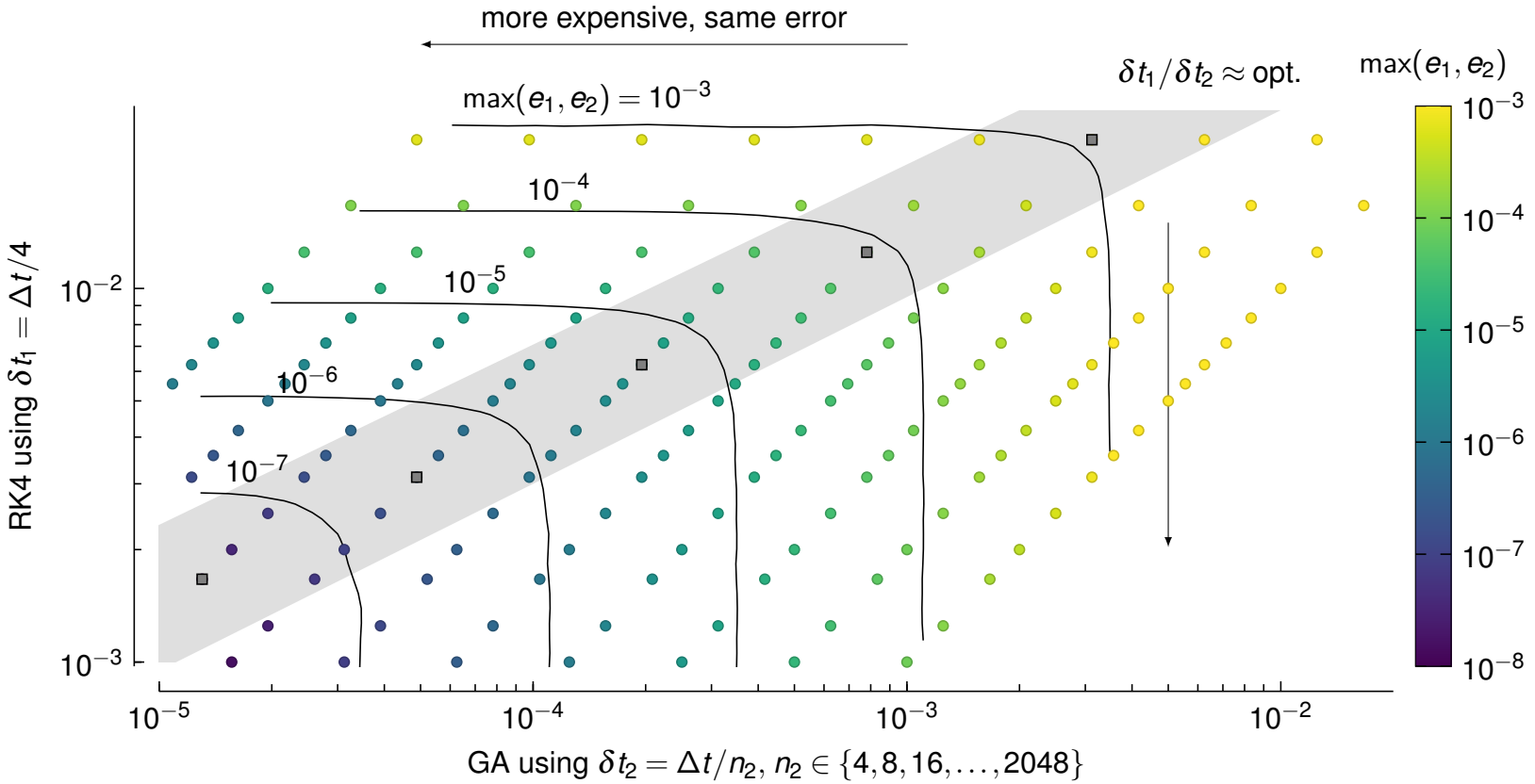


classic Runge-Kutta method

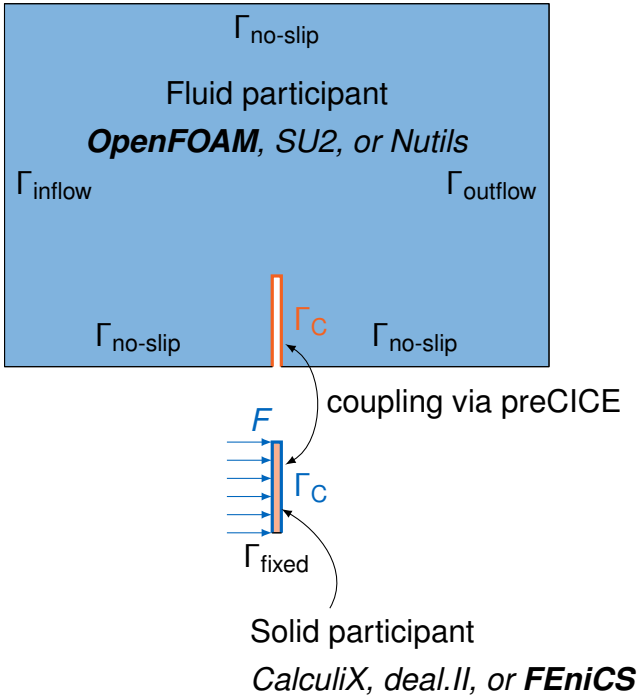


Radau IIA

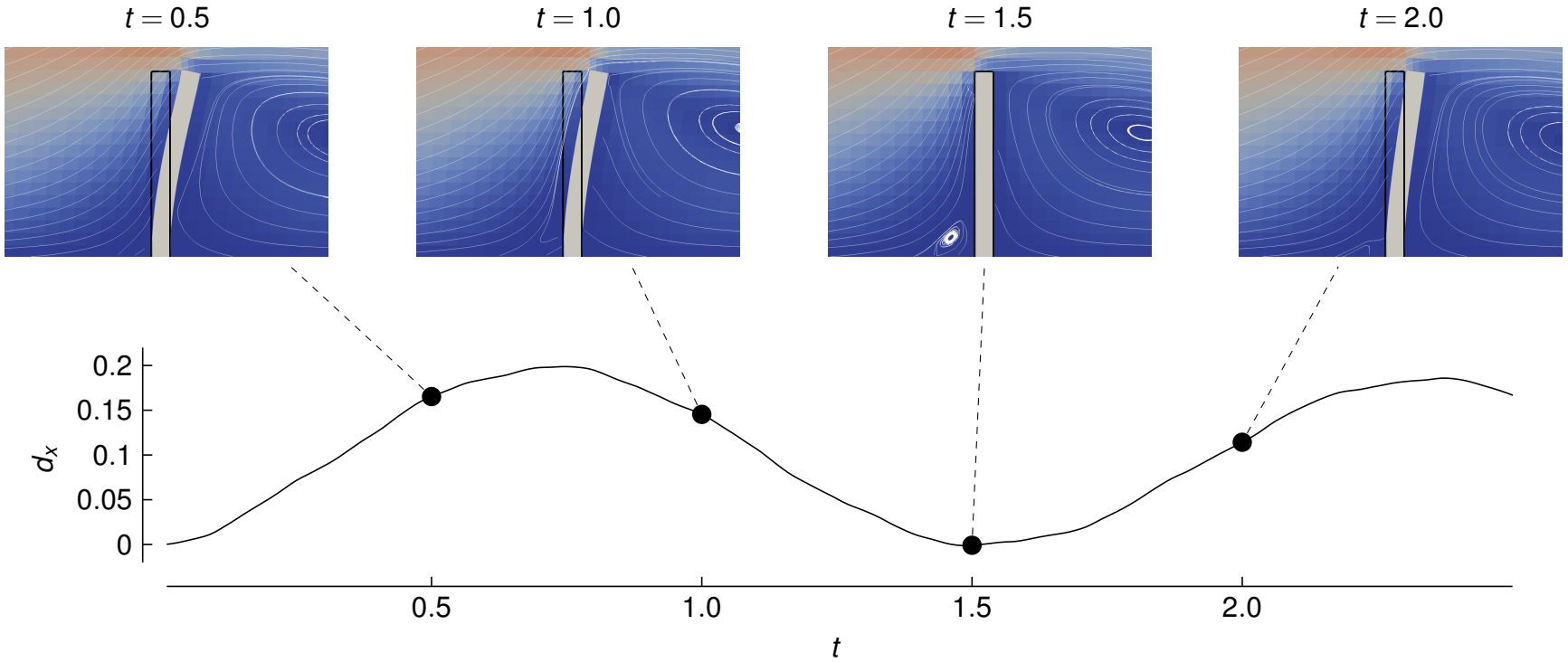
`tutorials/oscillator` (validation)



`tutorials/perpendicular-flap` (application?)

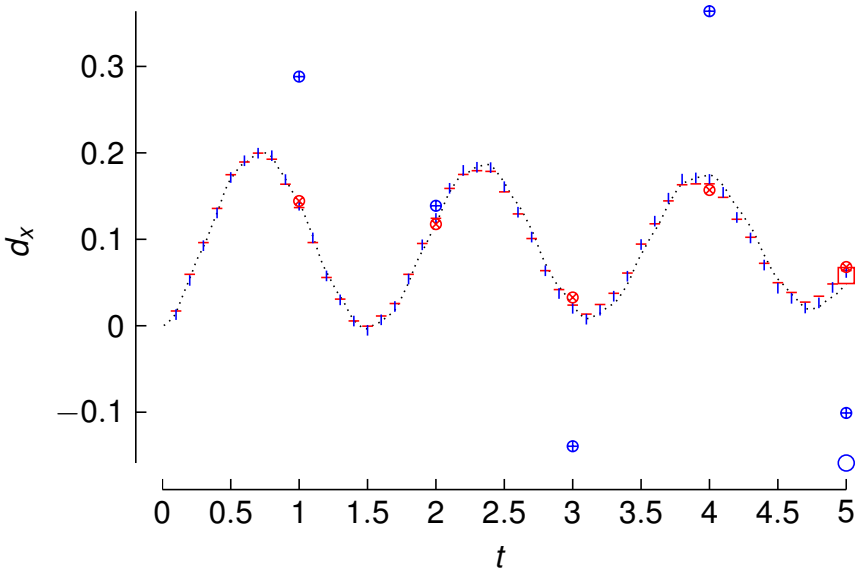


`tutorials/perpendicular-flap` (application?)



`tutorials/perpendicular-flap` (application?)

Large windows and IQN

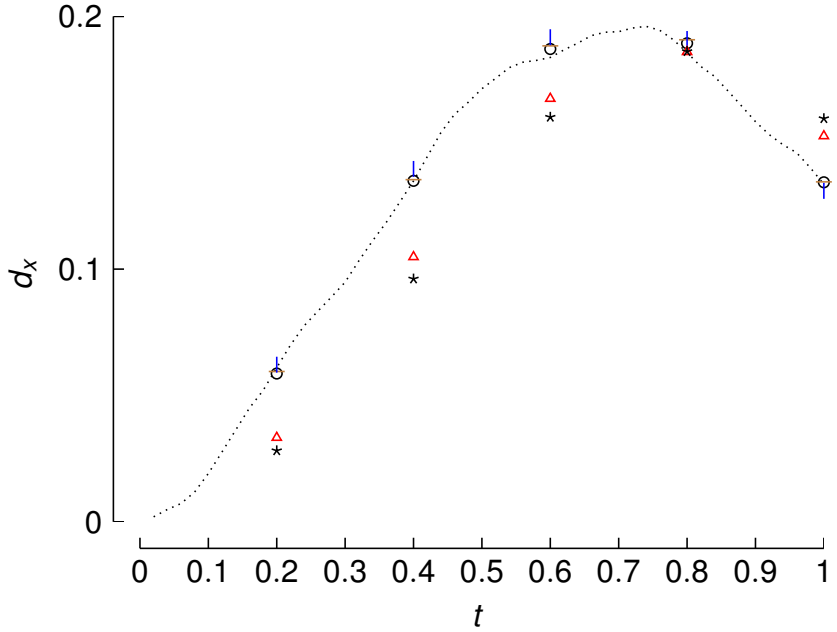


	Δt	$\sigma_F = \sigma_S$	$\delta t_F = \delta t_S$	substeps	#QN	cost
·····	0.01	1	0.01	irrelevant	1500	1500
|	0.1	10	0.01	"false"	151	1510
–	0.1	10	0.01	"true"	151	1510
⊕	1	100	0.01	"false"	24	2400
⊗	1	100	0.01	"true"	21	2100
○	5	500	0.01	"false"	5	2500
□	5	500	0.01	"true"	8	4000

Using "bleeding edge" WI-IQN by Niklas Kotarsky from Lund University `https://github.com/precice/precice/pull/2005`

`tutorials/perpendicular-flap` (application?)

Multirate and IQN



	Δt	σ_F	σ_S	δt_F	δt_S	μ_{SF}	#QN	#UR
·····	0.02	20	20	0.001	0.001	1	222	269
○	0.2	20	40	0.01	0.005	0.5	16	36
—	0.2	20	20	0.01	0.01	1	16	36
|	0.2	20	10	0.01	0.02	2	16	37
△	0.2	20	5	0.01	0.04	4	16	38
⋆	0.2	20	2	0.01	0.1	10	16	37

Using "bleeding edge" WI-IQN by Niklas Kotarsky from Lund University `https://github.com/precice/precice/pull/2005`

Conclusion

State of development

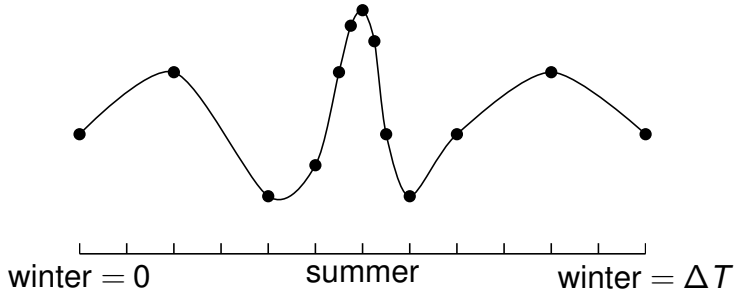
- Multirate time-stepping + black-box + usable API
- Time interpolation new in preCICE v3
- First tutorials use time interpolation

Where to be careful:

- Requirement: High order interpolation needs subcycling
- Officially in preCICE v3.1: Only underrelaxation if substeps
- "Bleeding edge" implementation of WI-IQN[1] (thanks to Niklas Kotarsky, Lund)

Open questions:

- Synchronization, subcycling, and performance?
- High-fidelity?



[1] Rüth, B., Uekermann, B., Mehl, M., Birken, P., Monge, A., & Bungartz, H. J. (2021). *Quasi-Newton waveform iteration for partitioned surface-coupled multiphysics applications*. International Journal for Numerical Methods in Engineering, 122(19), 5236-5257.

Community

preCICE Workshop 2023@Munich

Stay in touch?

- `precice.org/community`
- `precice.discourse.group`

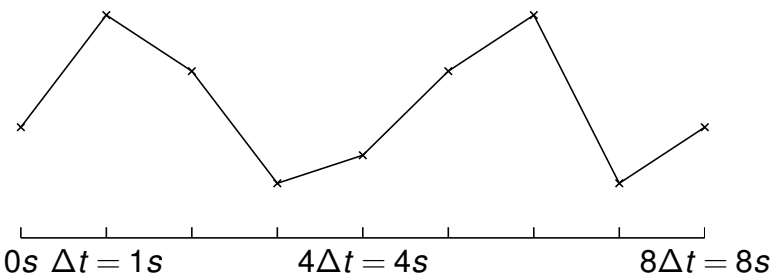
Conferences

- WCCM + preCICE course
  July 21-26 2024@Vancouver
- preCICE Workshop
  Sept 2024@Stuttgart
  (deadline for contributions July 7)
- COUPLED
  2025@Sicily

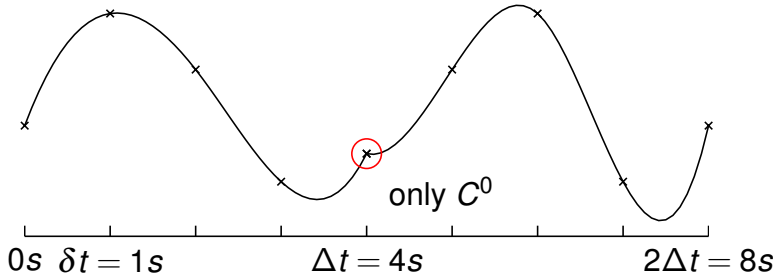
Subcycling

- Time window size $\Delta t \geq$ time step size δt_1 and δt_2.
- Do n time steps in window: $\Delta t = n_1 \delta t_1 = n_2 \delta t_2$
- Allows to create BSpline of degree $n - 1$. (Goal reached: Something better than linear interpolation)
- Restriction: Only use data of current window!
- Larger window + subcycling has impact on number of QN iterations[1]

Without subcycling



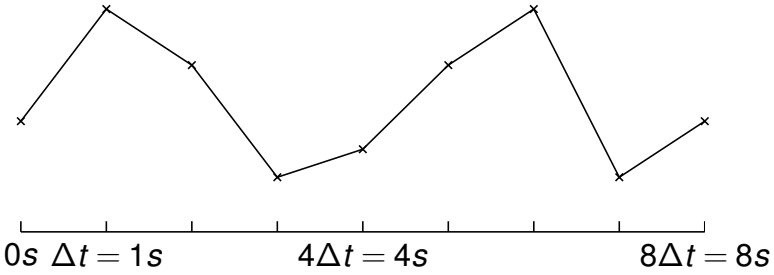
With subcycling (third order BSpline)



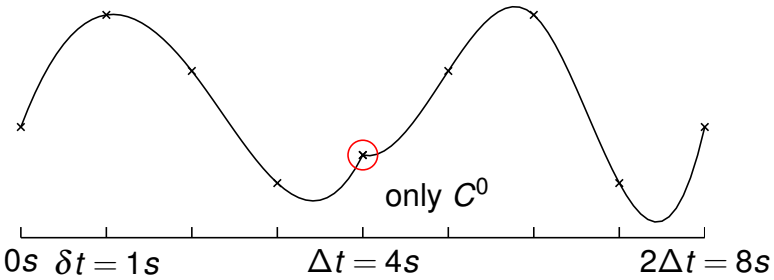
[1] Rüth, B, Uekermann, B, Mehl, M, Birken, P, Monge, A, Bungartz, H-J. Quasi-Newton waveform iteration for partitioned surface-coupled multiphysics applications. Int J Numer Methods Eng. 2021; 122: 5236– 5257. https://doi.org/10.1002/nme.6443

QN iterations

Without subcycling



With subcycling (third order BSpline)

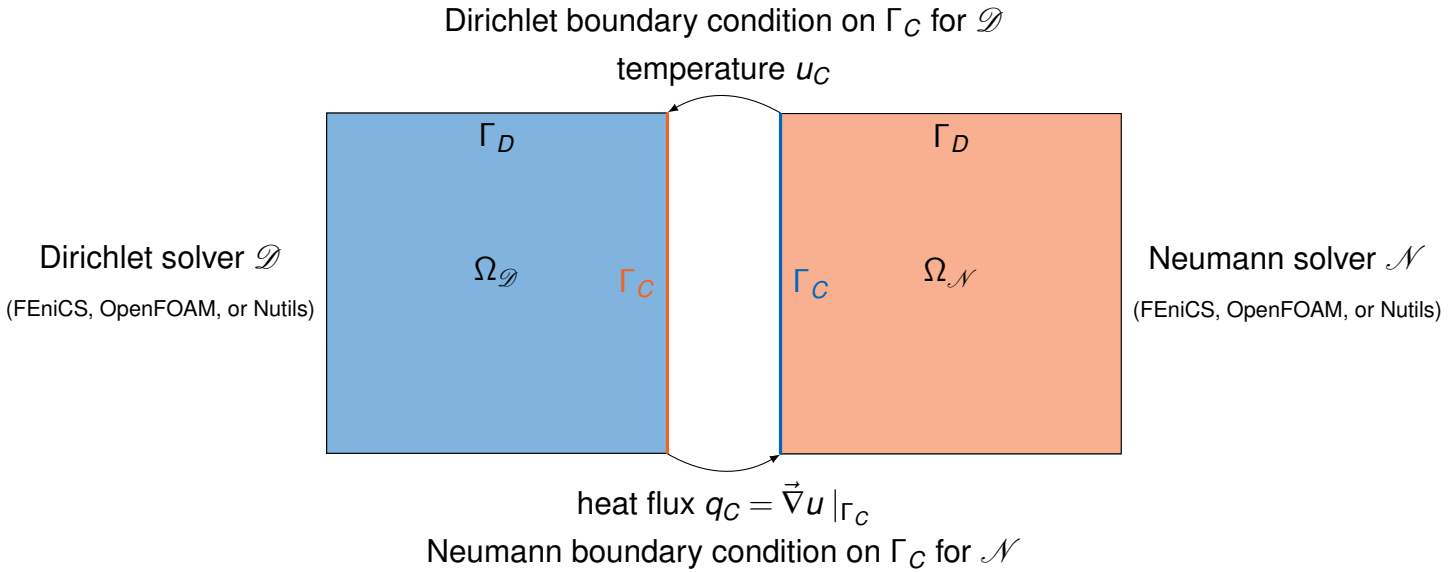


rQN-WI | Δt	0.5	0.1
WI$(1,1;1)$	7.85	5.45
WI$(5,5;1)$	10.95	7.48

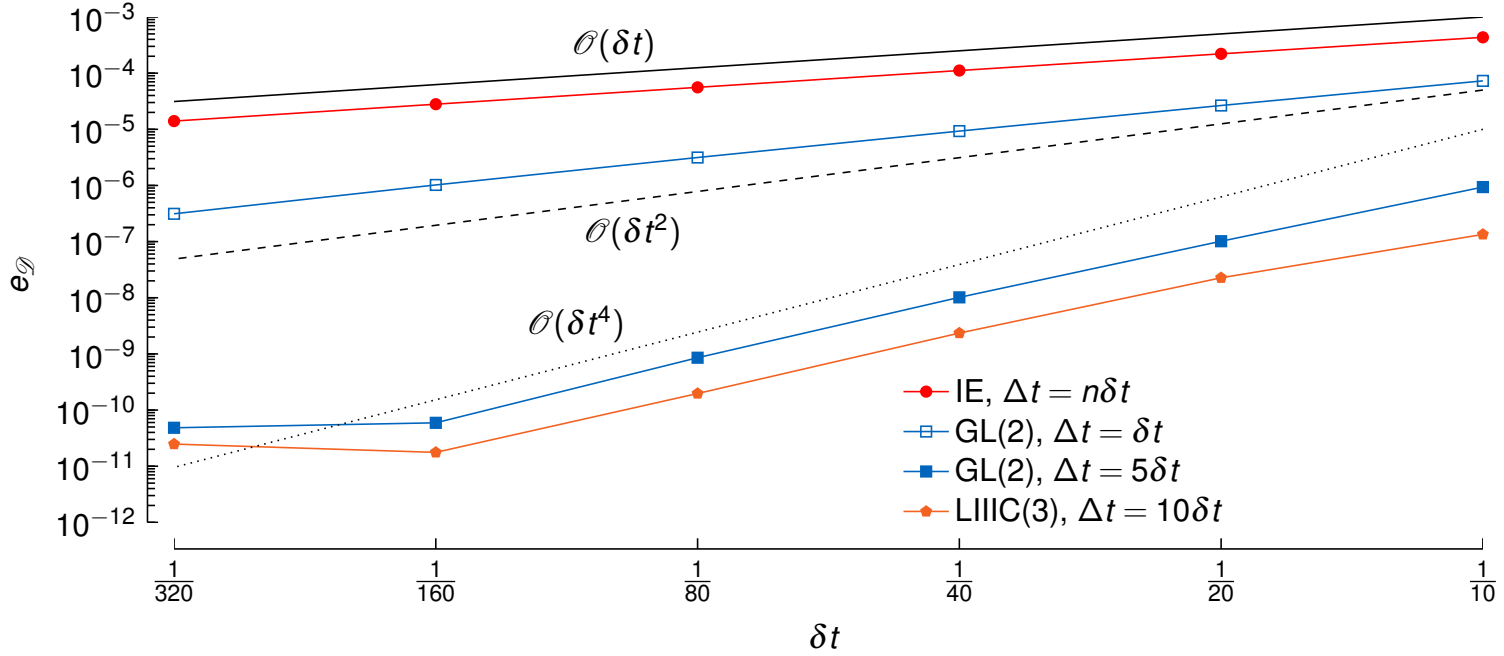
rQN-WI means we only use the data at the end of the window for Quasi-Newton. Different example case, but similar implementation. More possibilities shown in[1].

[1] Rüth, B, Uekermann, B, Mehl, M, Birken, P, Monge, A, Bungartz, H-J. Quasi-Newton waveform iteration for partitioned surface-coupled multiphysics applications. Int J Numer Methods Eng. 2021; 122: 5236– 5257. https://doi.org/10.1002/nme.6443

tutorials/partitioned-heat-conduction



tutorials/partitioned-heat-conduction

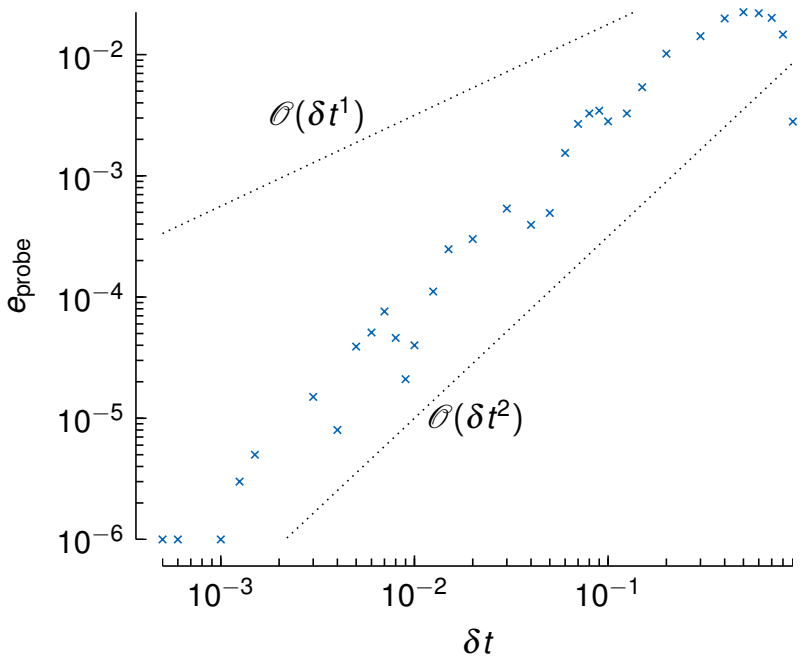


For more see Bachelor's thesis by Niklas Vinnitchenko *Evaluation of Higher-Order Coupling Schemes with FEniCS-preCICE*

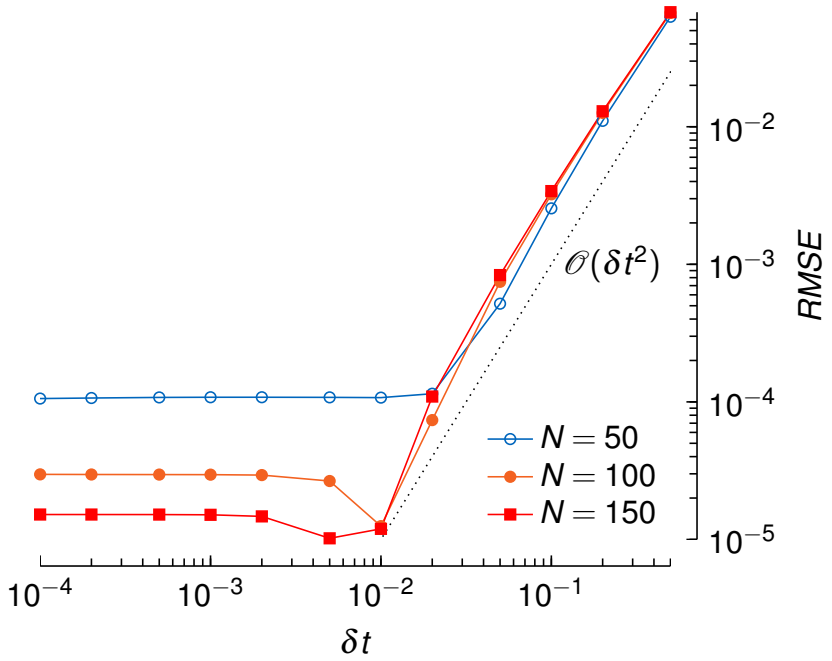
RK4 with expert knowledge

```python
# determine time step size δt for this time step
precice_dt = participant.get_max_time_step_size()
dt = min(precice_dt, solver_dt)

s = 4  # number of stages s = 4
rhs, k = s * [], s * []
for i in range(s):
    rhs[i] = participant.read_data("Force", c[i] * dt)  # evaluate at c_i δt
    k[i] = A.dot(x + ...) + rhs[i]  # compute stages k_i of scheme

x_new = dense_output(x_0, dt, k)  # x(θ) = x_0 + δt Σ_{i=1}^{s=4} k_i b_i(θ)

for i in range(s-1):  # write and advance s−1 times to get a p = s−1 polynomial (x_0 known)
    displ = extract_displacement(x_new((i+1) * dt / (s-1)))
    participant.write_data("Displacement", displ)
    participant.advance(dt / (s-1))  # after s−1 advance calles δt is complete
```

`tutorials/perpendicular-flap` (convergence)



CalculiX beam (FEniCS beam similar)



OpenFOAM Taylor-Green vortex (channel only 1st order!)

Guided research project by Marc Amorós Trepat *Review of higher-order time stepping schemes in open-source solvers*