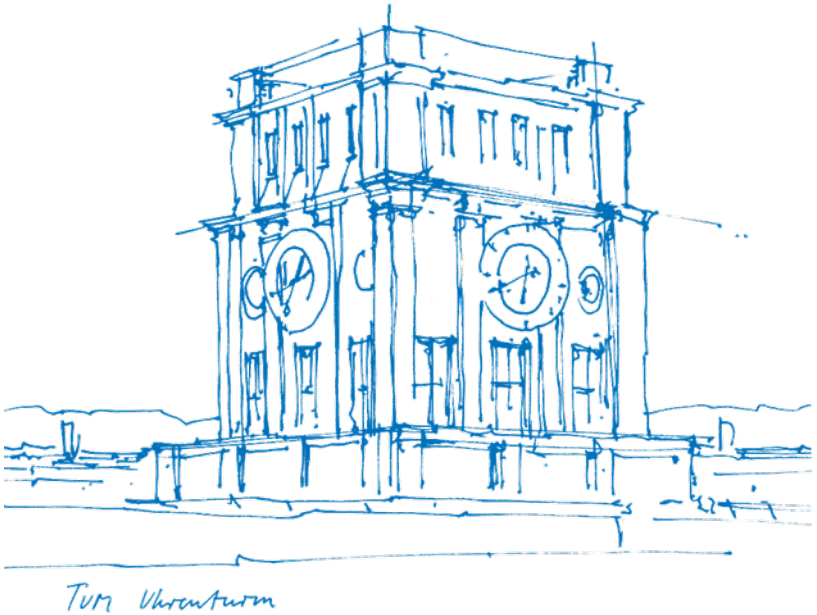
Waveform iteration in partitioned multiphysics with preCICE

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

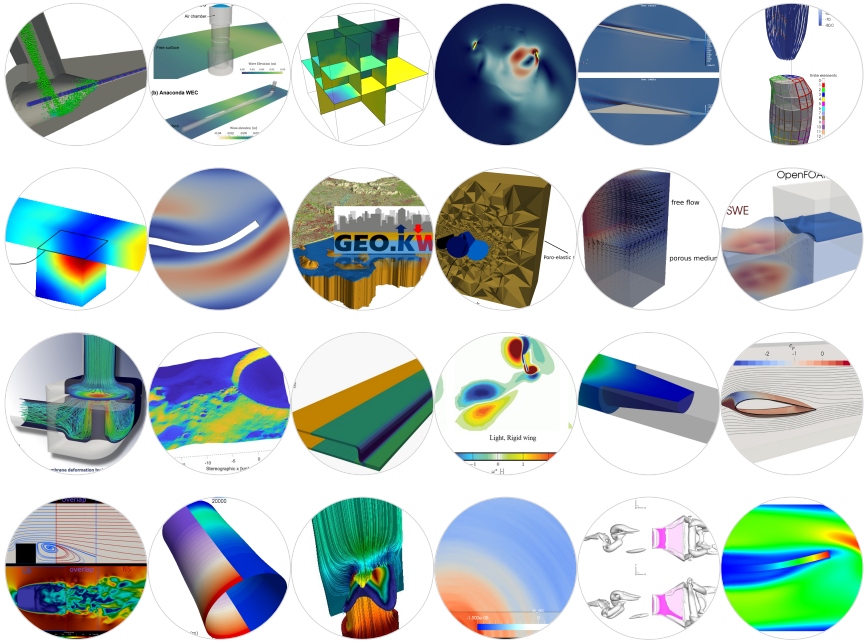
CREATOR Seminar, TU Darmstadt, March 11, 2024



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

Outline

precice.org/community-projects

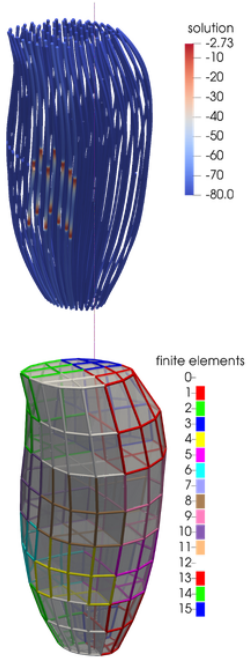


Community
Overview & news
preCICE Workshop 2024
iacm WCCM 2024
Support preCICE
Stories
Contributors
Contribute to preCICE
Community channels
Past events

Coupling mechanics and electrophysiology in skeletal muscles

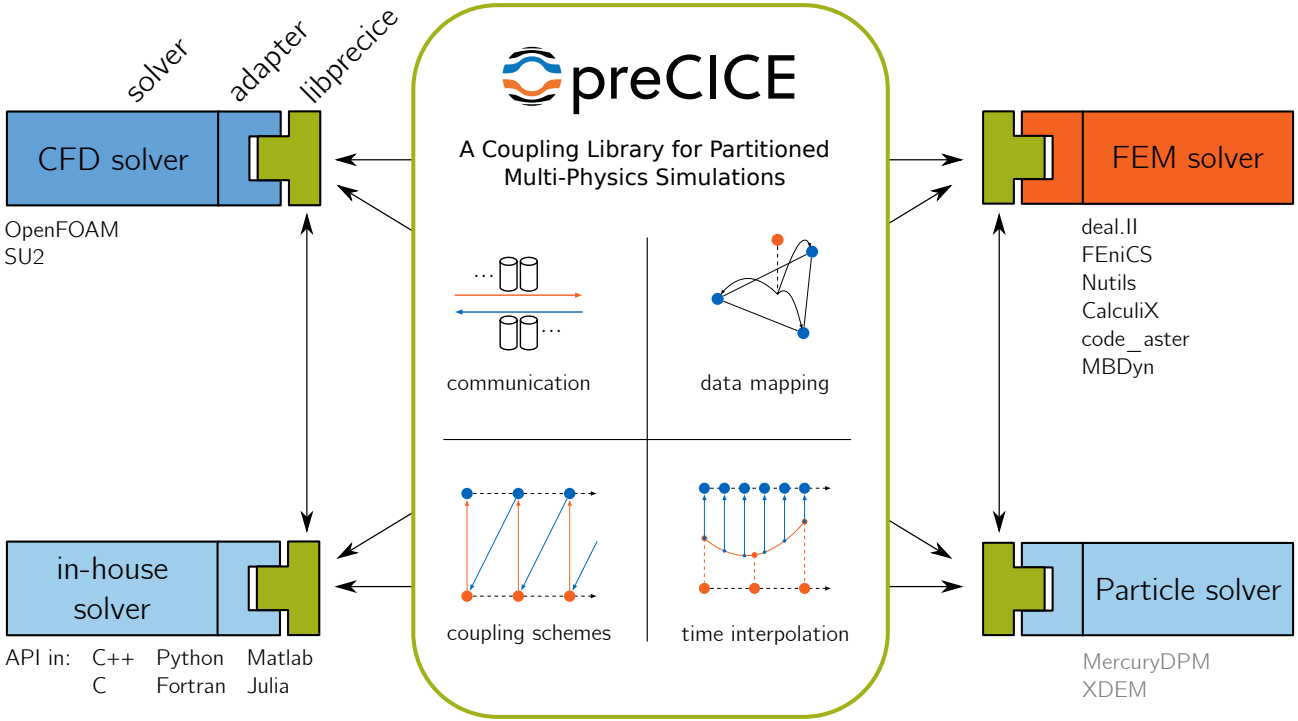
In our project "Towards a digital human", we improve the understanding of the neuromuscular system by simulating multi-scale models of skeletal muscles with realistic resolutions on HPC clusters. In particular, we implemented a complete biophysical multi-scale model of the neuromuscular system in our open-source software OpenDiHu. We simulate biochemical processes of subcellular force generation (0D), muscle fiber activation (1D) as well as electric conduction in the extracellular space (3D). In order to additionally take muscle contraction into account, we apply volume coupling between the finely resolved 3D electromyography model, which computes muscle activation, and a coarse 3D solid mechanics solver, which computes the muscle deformation. Using preCICE, we are able to simulate the respective highly resolved scenarios with 100 million degrees of freedom on more than 9,000 cores on the supercomputer Hawk at the High Performance Computing Center Stuttgart. Furthermore, preCICE helps us to numerically couple multiple mechanical models with different materials, e.g., the biceps brachii muscle and its tendons. Using preCICE, our code is envisioned to become a building block in combination with other external solvers within a comprehensive "digital human model". Learn more

—**Benjamin Maier and David Schneider** Simulation of Large Systems, University of Stuttgart, Germany

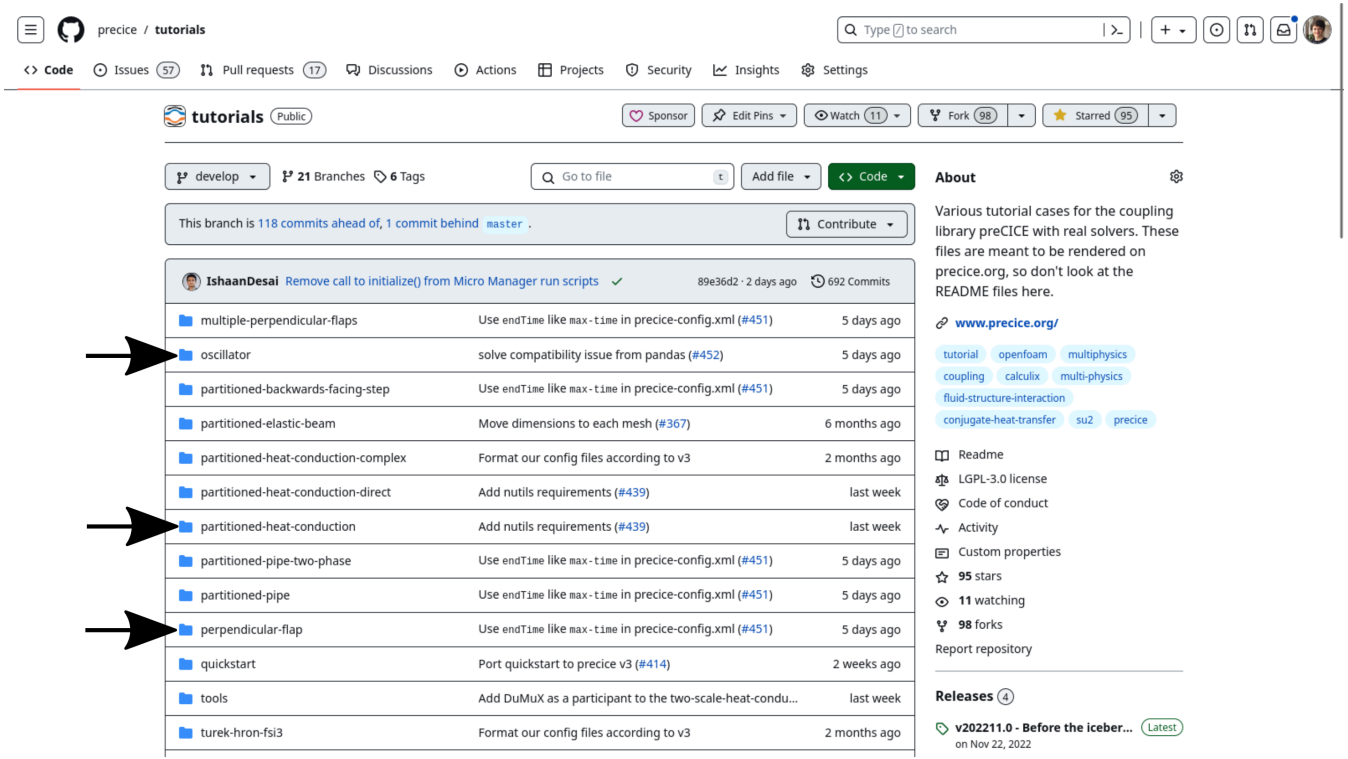


Coupling mechanics and electrophysiology in skeletal muscles

preCICE overview



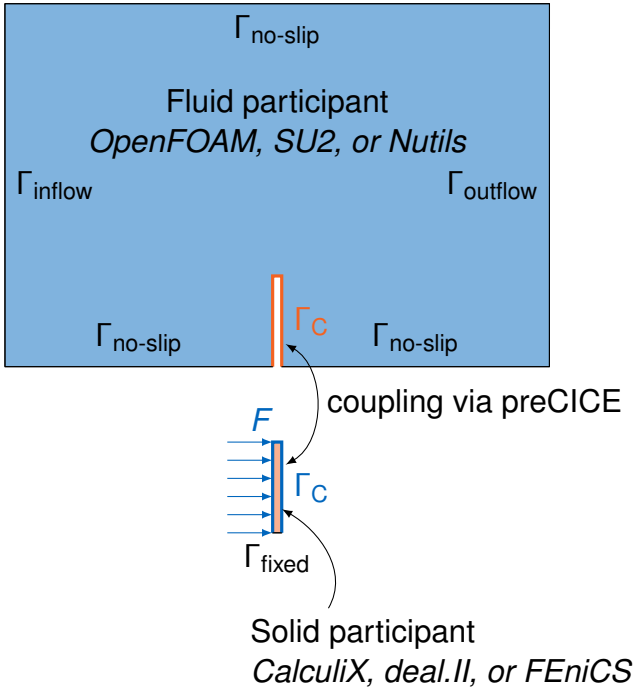
github.com/precice/tutorials



tutorials/perpendicular-flap

DEMO TIME

tutorials/perpendicular-flap



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

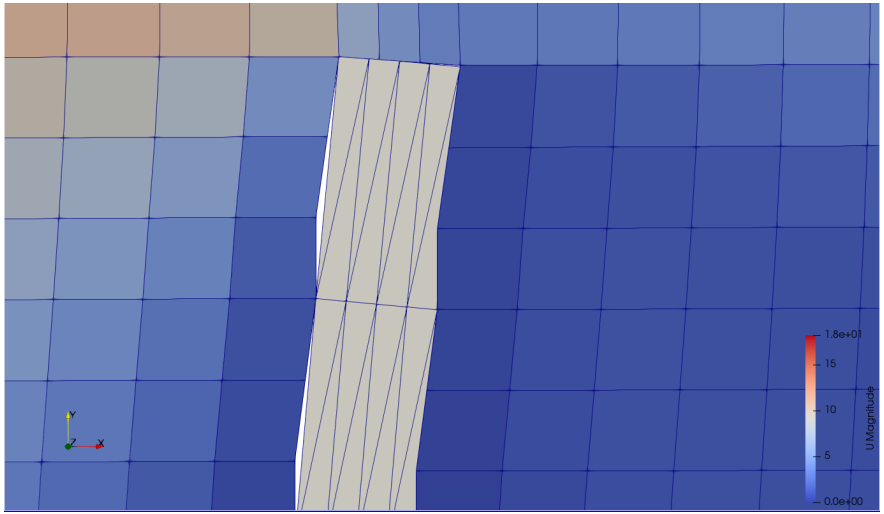
- $\mathscr{F}(\mathscr{S}(f^k)) = \tilde{f}^k$
- $\tilde{f}^k \xrightarrow{\mathscr{A}} f^{k+1}$

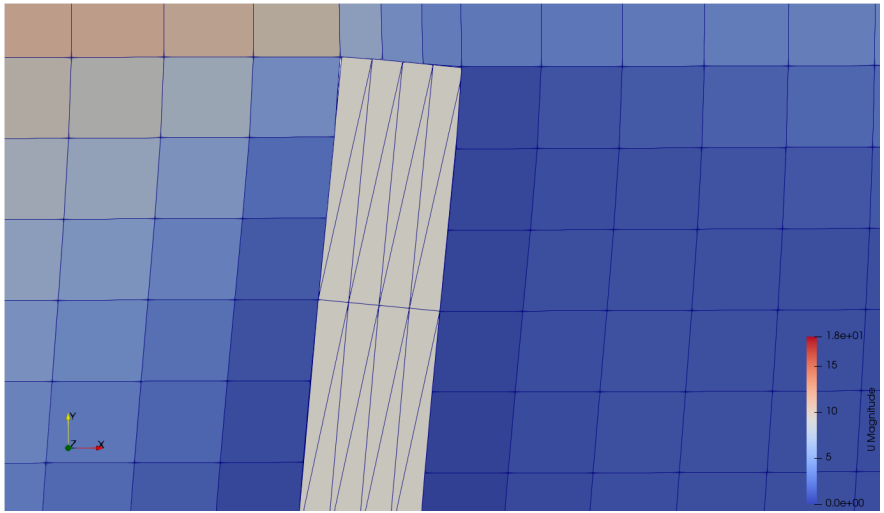
Picard iteration + acceleration

tutorials/perpendicular-flap

DEMO TIME

tutorials/perpendicular-flap

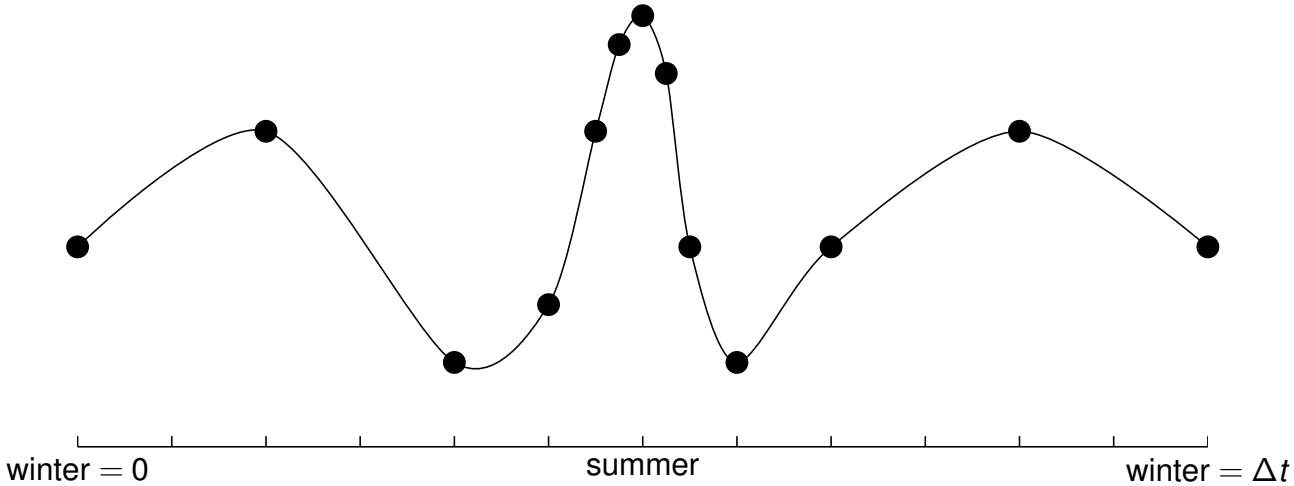




preCICE v3: Time interpolation

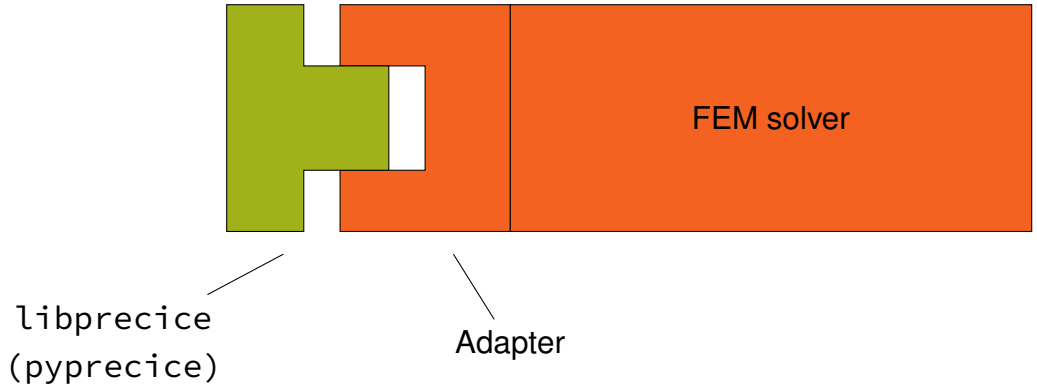
preCICE Workshop 2023: Two "ice sheet talks"[1]

- 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]

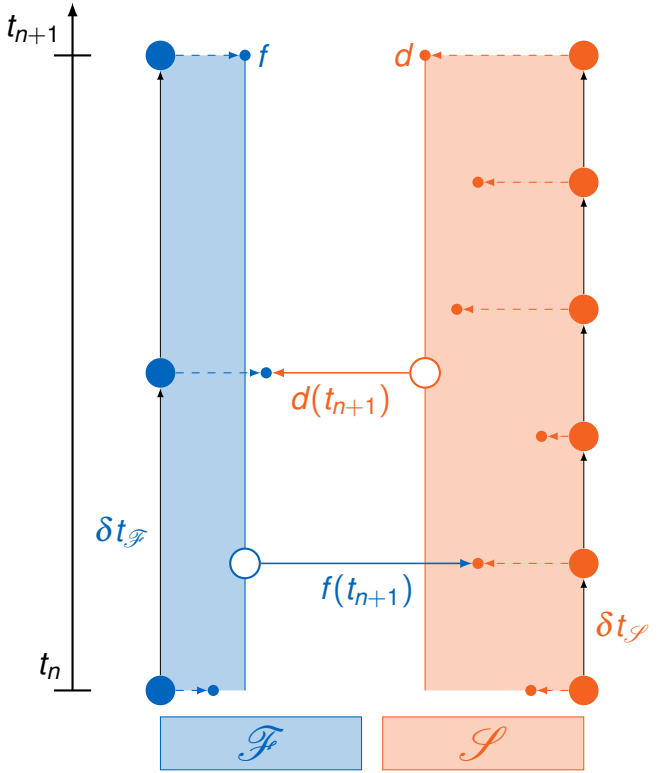
[1] https://github.com/precice/precice/releases/tag/v3.0.0

API for implicit Euler

```python
# example: FSI coupling, perspective of fluid solver 𝓕:
participant = precice.Participant("Fluid", "precice-config.xml")

# leaving out coupling mesh and data initialization

participant.initialize()

while participant.is_coupling_ongoing():
  # store checkpoint, if needed
  dt = participant.get_max_time_step_size()
  displacement = participant.read_data(dt)  # Read displacement at t_n + Δt: d_{n+1}
  forces = forces + dt * dfdt(displacements)  # Implicit Euler
  participant.write_data(forces)
  participant.advance(dt)
  # read checkpoint, if needed

participant.finalize()
```

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 v2: single-value coupling

Coupled evolution equations

$$\mathscr{F}(d) = f \quad \rightarrow \quad \mathscr{F}_1(u_2) = u_1 \qquad \text{solves: } \frac{\partial u_1}{\partial t} = f_1(u_1, u_2)$$

$$\mathscr{S}(f) = d \quad \rightarrow \quad \mathscr{F}_2(u_1) = u_2 \qquad \text{solves: } \frac{\partial u_2}{\partial t} = f_2(u_1, u_2)$$

$$\mathscr{F}(\mathscr{S}(f^k)) = \tilde{f}^k \quad \rightarrow \quad \mathscr{F}_2\left(\mathscr{F}_1\left(u_2^k\right)\right) = u_2^k \qquad \text{(let's forget about } \mathscr{A}(\cdot))$$

preCICE v2: single-value coupling

Coupled evolution equations

$$\mathscr{F}(d) = f \quad \rightarrow \quad \mathscr{F}_1(u_2) = u_1 \qquad \text{solves: } \frac{\partial u_1}{\partial t} = f_1(u_1, u_2)$$

$$\mathscr{S}(f) = d \quad \rightarrow \quad \mathscr{F}_2(u_1) = u_2 \qquad \text{solves: } \frac{\partial u_2}{\partial t} = f_2(u_1, u_2)$$

$$\mathscr{F}(\mathscr{S}(f^k)) = \tilde{f}^k \quad \rightarrow \quad \mathscr{F}_2\left(\mathscr{F}_1\left(u_2^k\right)\right) = u_2^k \qquad \text{(let's forget about } \mathscr{A}(\cdot)\text{)}$$

Discretization with trapezoidal rule & coupling

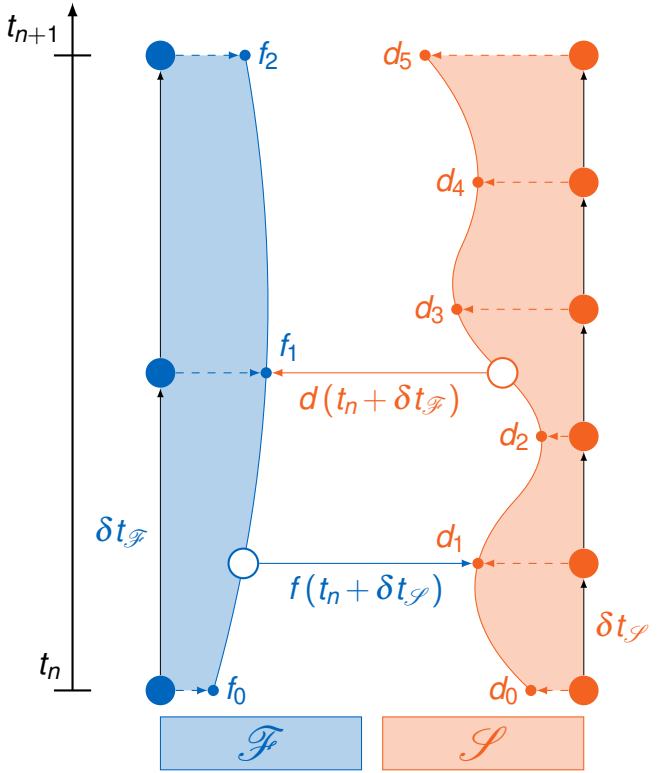
	update scheme							stability	order
fully explicit	$u_1^{n+1} =$	$u_1^n + \frac{\delta t}{2}[$	$f_1(u_1^n, u_2^n)$	$+$	$f_1(u_1^{n+1}, u_2^n)$	$]$		depends on δt	$\mathscr{O}(\delta t)$
$u_1 \rightarrow u_2$	$u_2^{n+1} =$	$u_2^n + \frac{\delta t}{2}[$	$f_2(u_1^{n+1}, u_2^n)$	$+$	$f_2(u_1^{n+1}, u_2^{n+1})$	$]$			
fully implicit	$u_1^{n+1} =$	$u_1^n + \frac{\delta t}{2}[$	$f_1(u_1^n, u_2^{n+1})$	$+$	$f_1(u_1^{n+1}, u_2^{n+1})$	$]$		depends on δt	$\mathscr{O}(\delta t)$
$u_1 \leftrightarrow u_2$	$u_2^{n+1} =$	$u_2^n + \frac{\delta t}{2}[$	$f_2(u_1^{n+1}, u_2^n)$	$+$	$f_2(u_1^{n+1}, u_2^{n+1})$	$]$			
waveform iteration	$u_1^{n+1} =$	$u_1^n + \frac{\delta t}{2}[$	$f_1(u_1^n, u_2^n)$	$+$	$f_1(u_1^{n+1}, u_2^{n+1})$	$]$		stable	$\mathscr{O}(\delta t^2)$
$u_1(t) \leftrightarrow u_2(t)$	$u_2^{n+1} =$	$u_2^n + \frac{\delta t}{2}[$	$f_2(u_1^n, u_2^n)$	$+$	$f_2(u_1^{n+1}, u_2^{n+1})$	$]$			

preCICE v3: waveform iteration

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

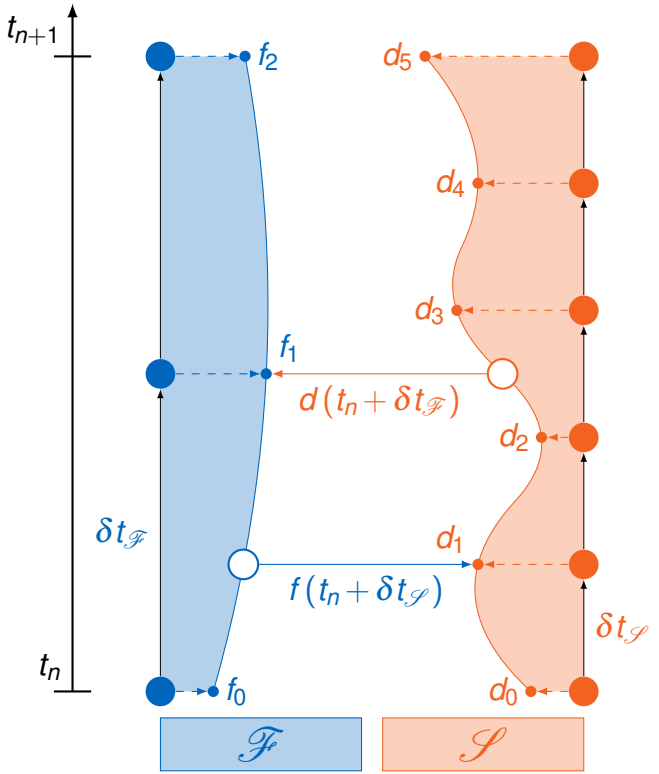
upgrade to preCICE v3

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



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>
```



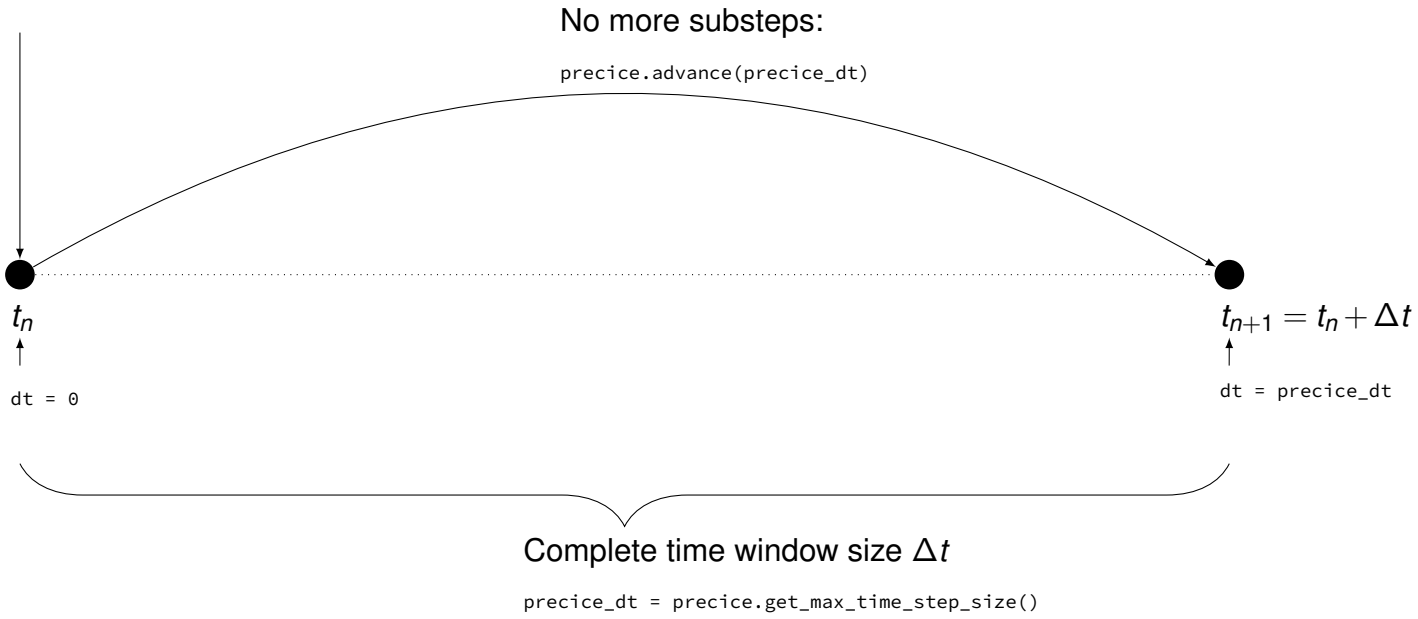
RK4 in code (simplified)

```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 for multirate

Read interpolated data from current time t_n:

```
displacement = precice.read_data(dt)
```



API for multirate

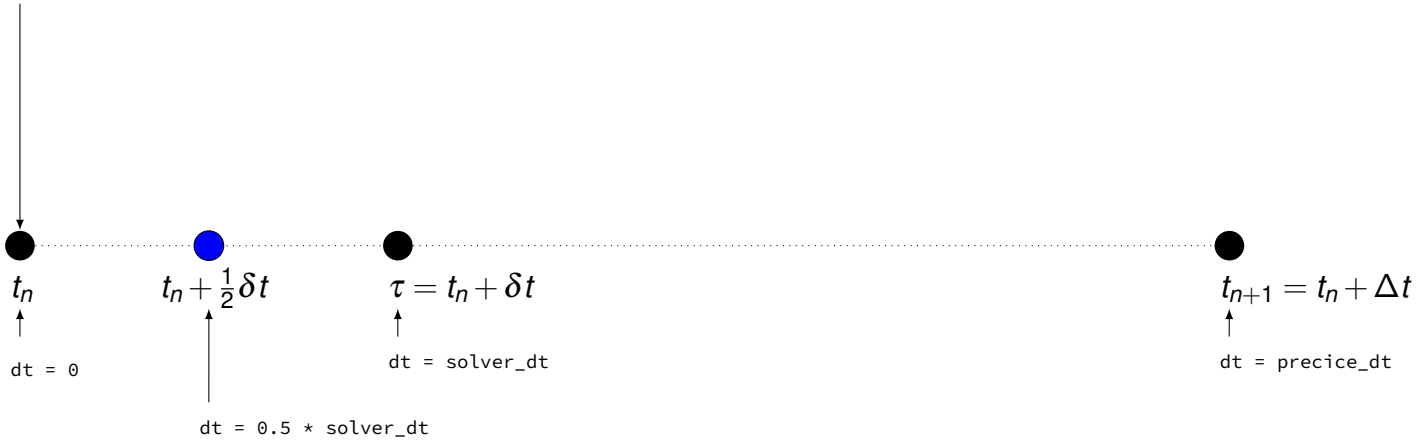
Read interpolated data from current time t_n:

```
displacement = precice.read_data(dt)
```

API for multirate

Read interpolated data from current time t_n:

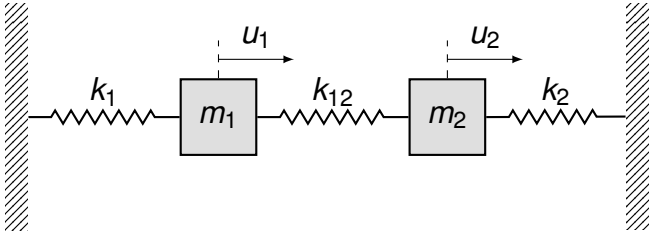
```
displacement = precice.read_data(dt)
```



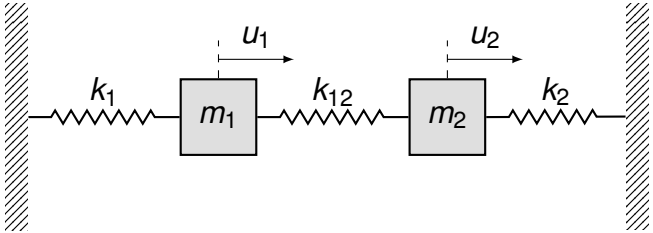
API with RK4 and multirate

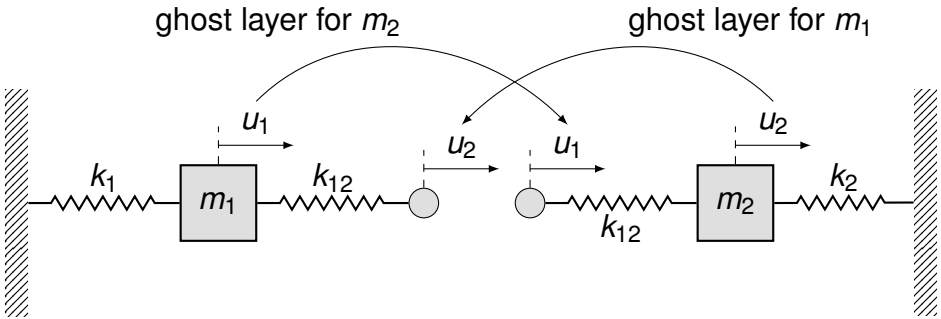
```python
# example: FSI coupling, perspective of fluid solver 𝓕:
# ...
participant.initialize()

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

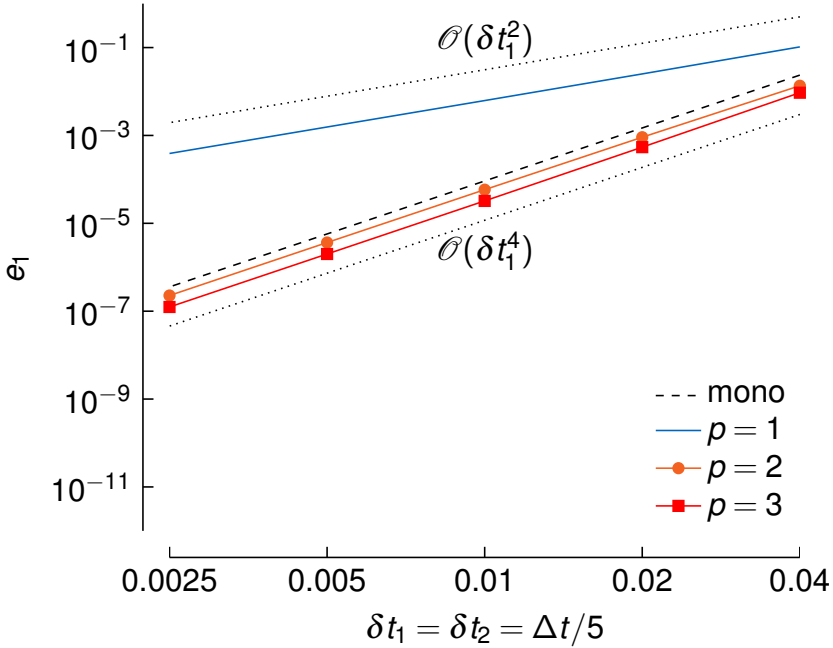


tutorials/oscillator

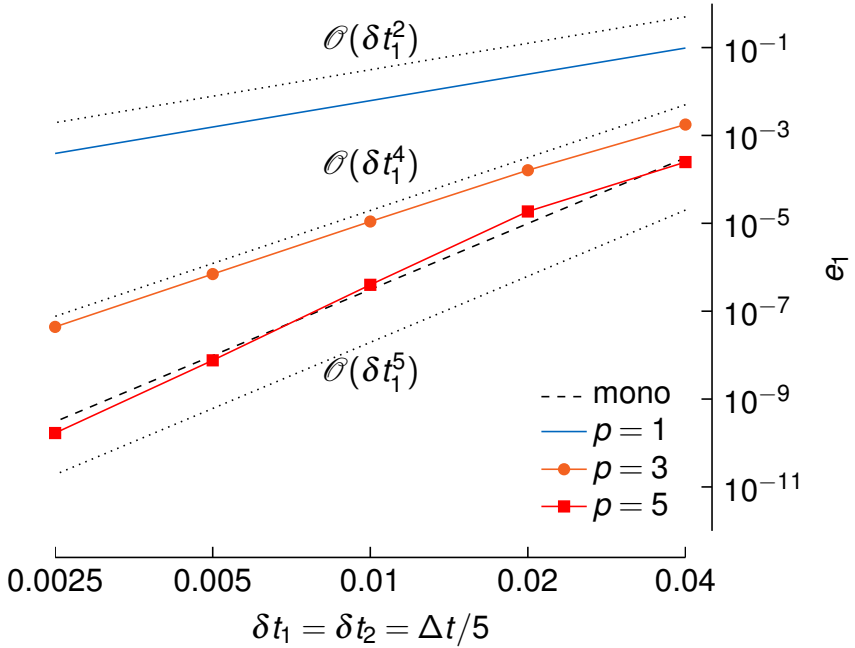




tutorials/oscillator

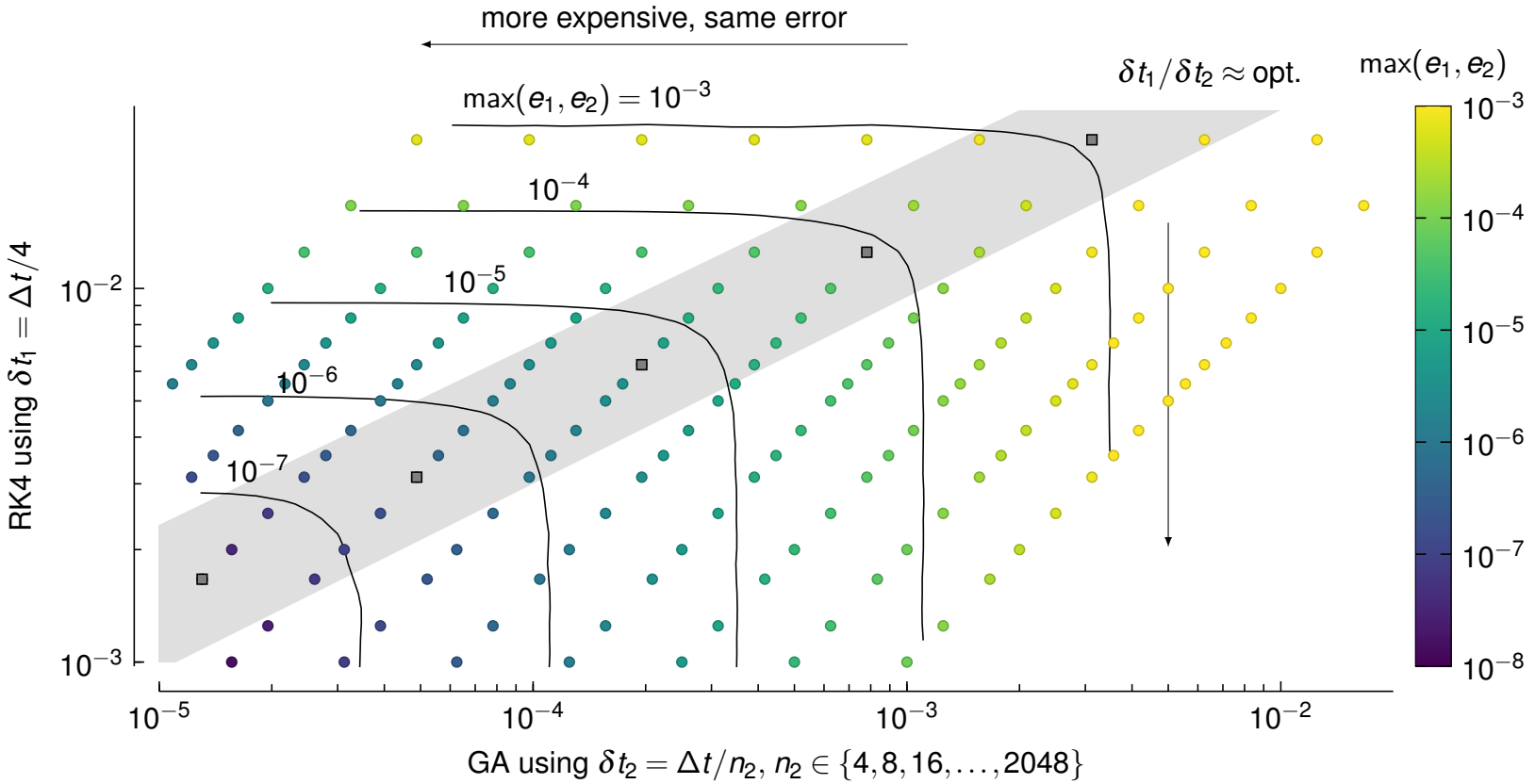


classic Runge-Kutta method

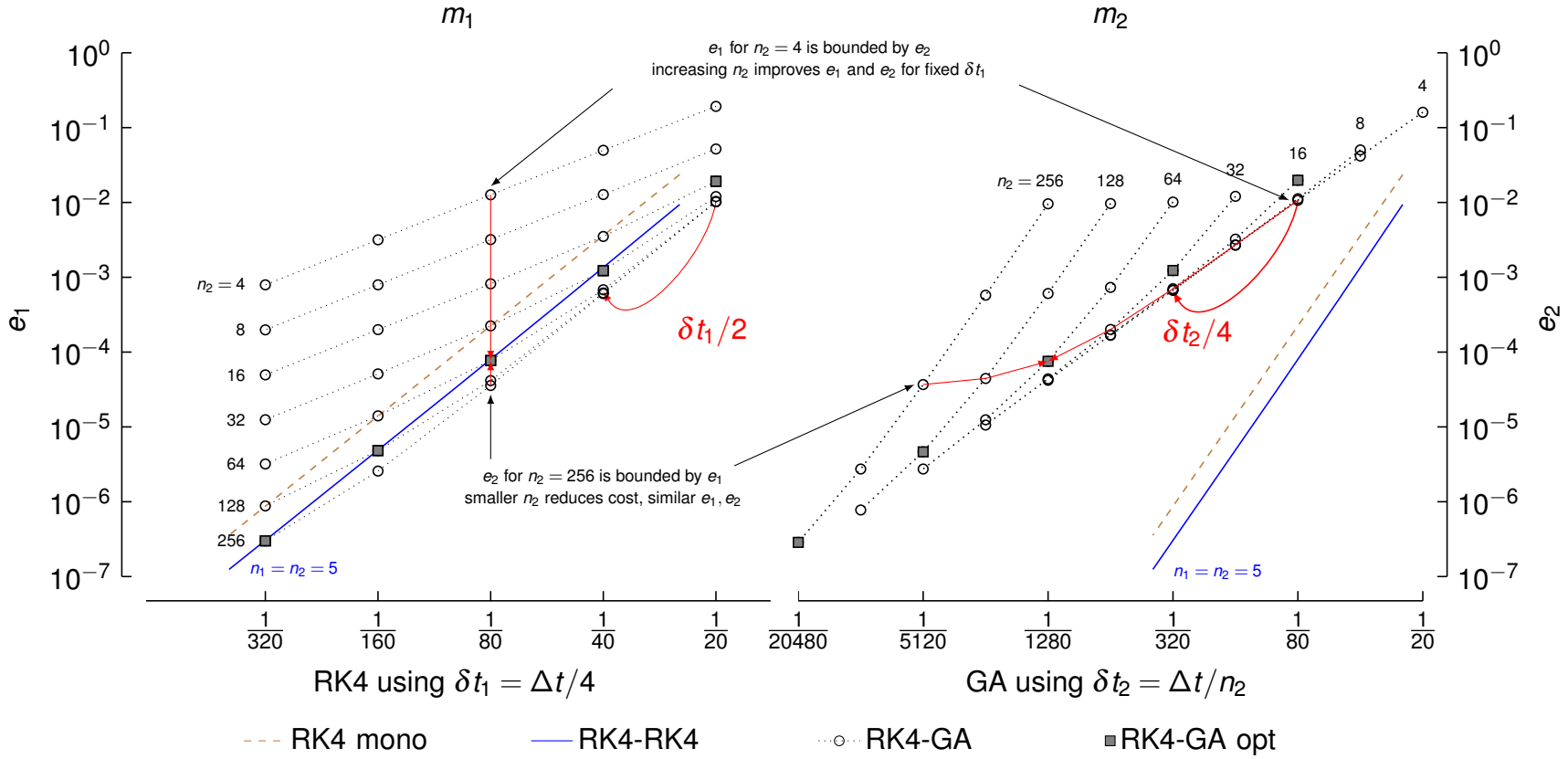


Radau IIA

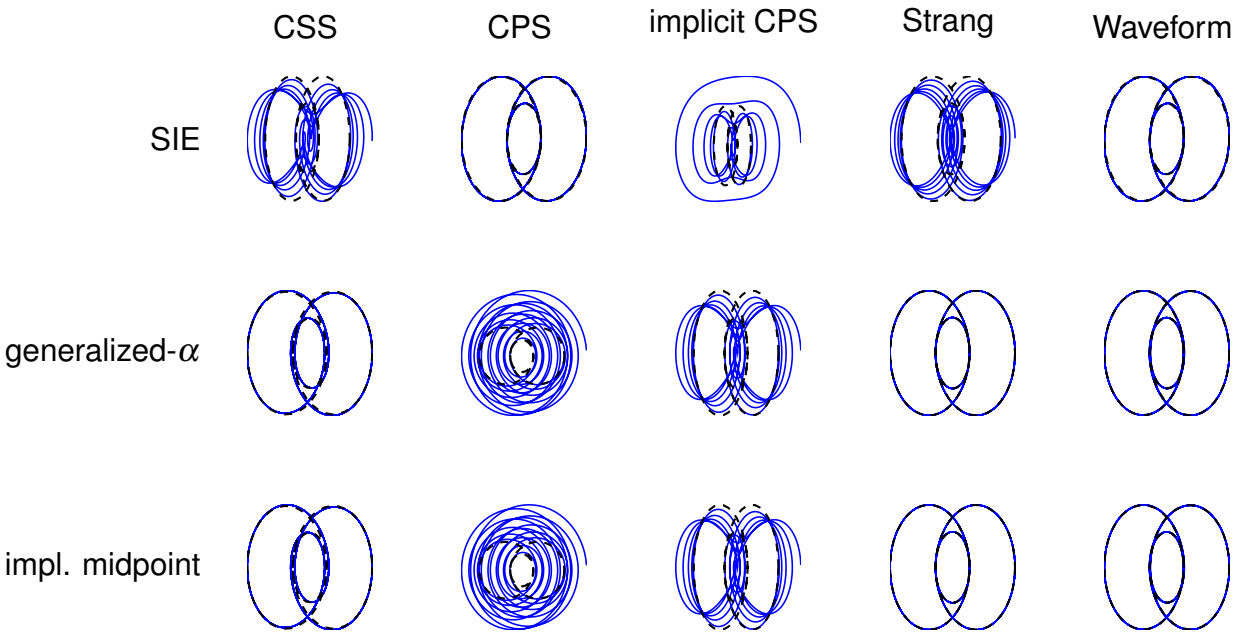
tutorials/oscillator



tutorials/oscillator

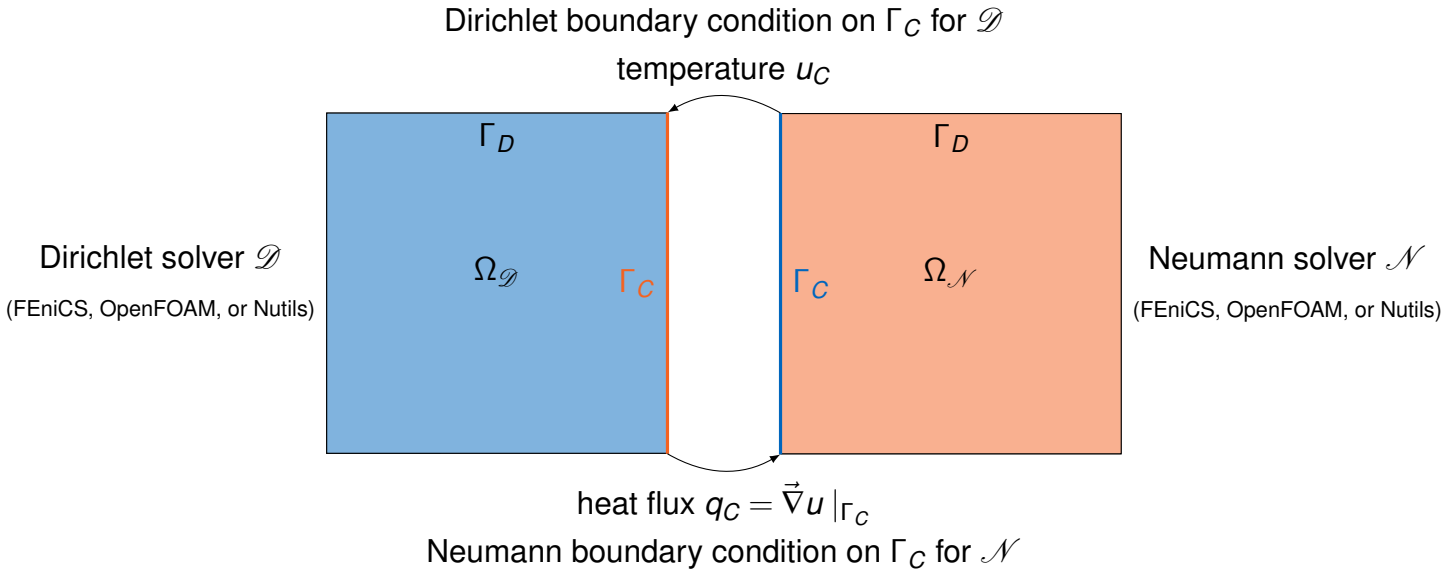


tutorials/oscillator

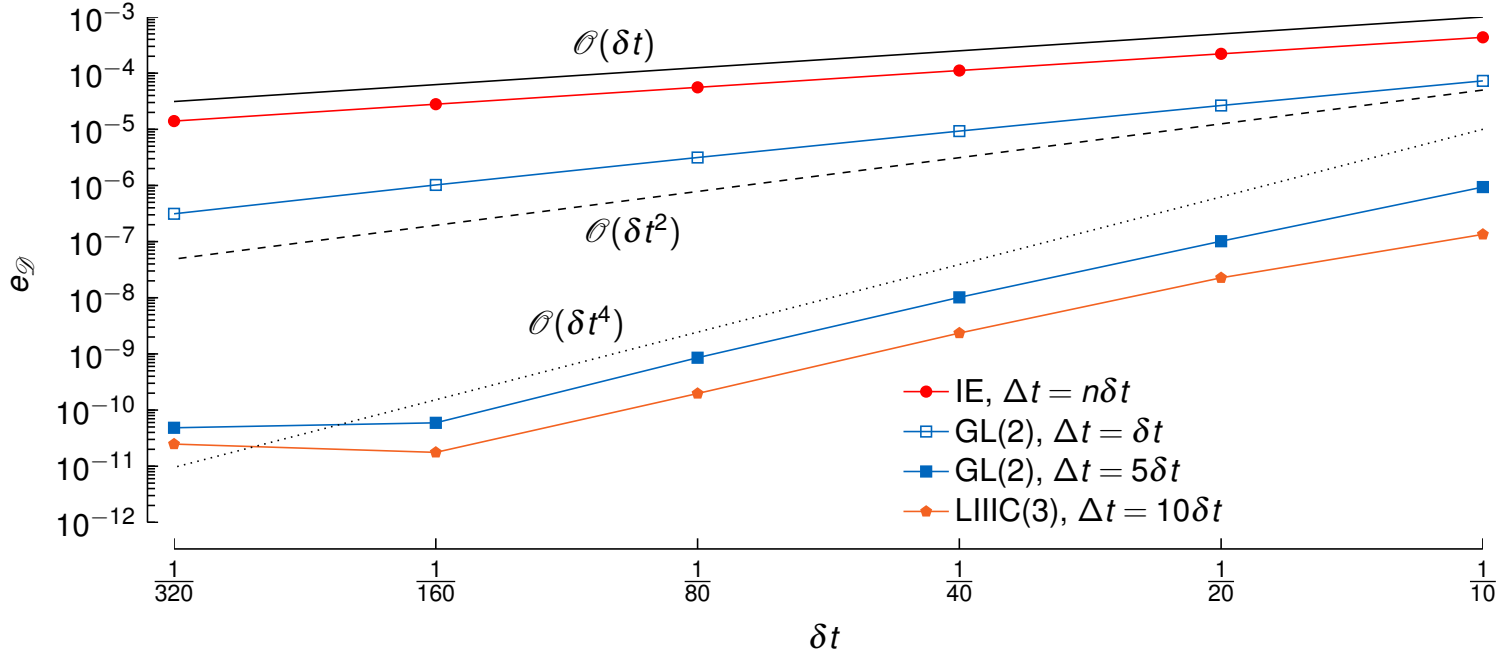


Schüller, V., Rodenberg, B., Uekermann, B., & Bungartz, H. J. (2022). *A Simple Test Case for Error Reduction of Black-Box Coupling Schemes*

tutorials/partitioned-heat-conduction

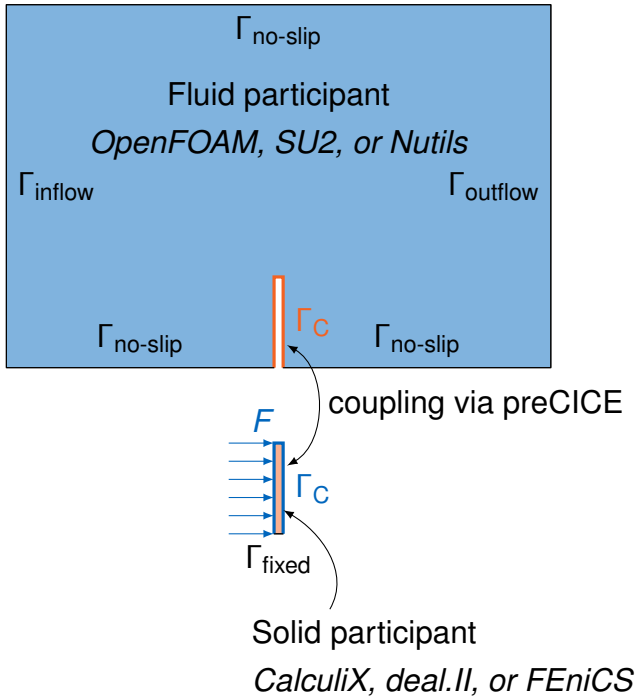


tutorials/partitioned-heat-conduction

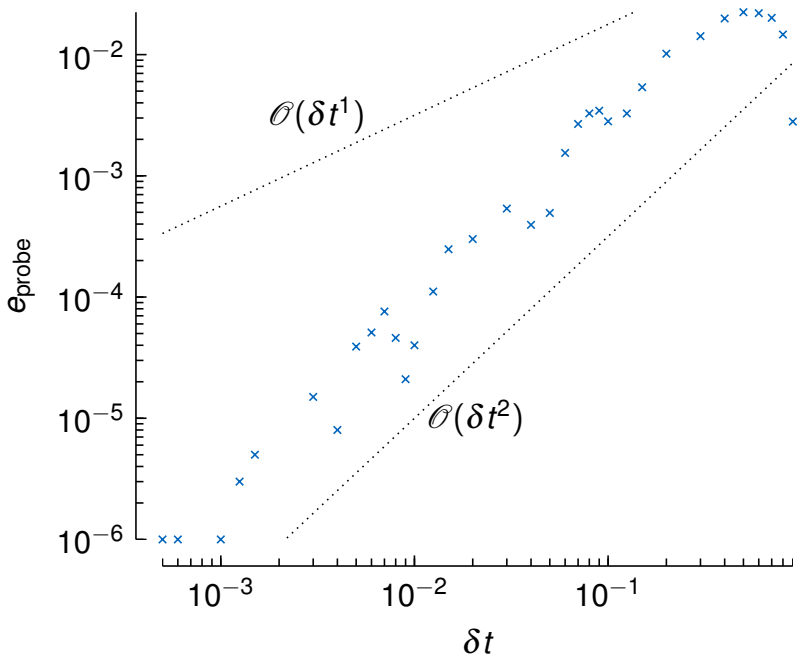


For more see Bachelor's thesis by Niklas Vinnitchenko *Evaluation of Higher-Order Coupling Schemes with FEniCS-preCICE*

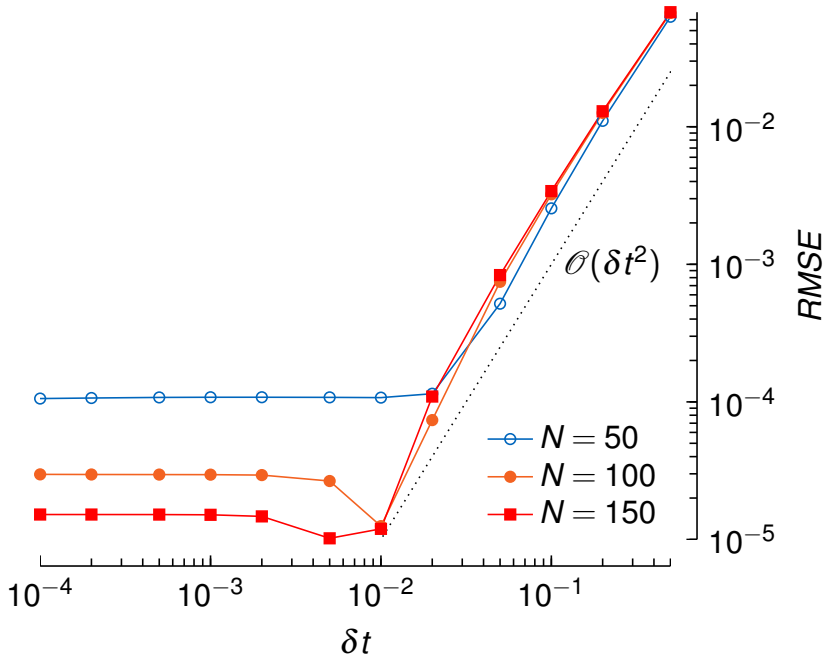
`tutorials/perpendicular-flap` (without coupling)



`tutorials/perpendicular-flap` (without coupling)



CalculiX beam (Probe at beam tip, compare to $\delta t = 10^{-5}$)



OpenFOAM Taylor-Green vortex

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

Interface quasi-Newton

Acceleration $\mathscr{A}(\cdot)$

- Coupled problem: $\mathscr{F}(\mathscr{S}(f^k)) = \tilde{f}^k$
- Acceleration: $\mathscr{A}(\tilde{f}^k) = f^{k+1}$
- Short: $\mathscr{H}(x^k) = \tilde{x}^k$ and $x^{k+1} = \mathscr{A}\left(\mathscr{H}(x^k)\right)$
- Underrelaxation: $\mathscr{A}(\tilde{x}^k) = (1-\omega)x^k + \omega\tilde{x}^k$

IQN = Interface quasi-Newton

- solve $\tilde{R}(\tilde{x}) = \tilde{x} - \mathscr{H}^{-1}(\tilde{x}) \stackrel{!}{=} 0$
- Newton step: $\mathscr{A}(\tilde{x}^k) = \tilde{x}^k - \tilde{R}^k / J_{\tilde{R}}^k$
- Jacobian: $V_k / J_{\tilde{R}}^k \approx W_k$ (multisecant equation)

IQN variants in preCICE

- IQN-ILS
  = IQN-inverse least squares
  = Anderson Acceleration
- IQN-IMVJ
  = IQN-inverse multi-vector Jacobian
  = generalized Broyden

Algorithm 1 Anderson acceleration

initial value x^0
$\tilde{x}^0 = \mathscr{H}(x^0)$ and $R^0 = \tilde{x}^0 - x^0$
$x^1 = x^0 + \omega R^0$
for $k = 1 \ldots$ **do**
  $\tilde{x}^k = \mathscr{H}(x^k)$ and $R^k = \tilde{x}^k - x^k$
  $V_k = \left[\Delta R^0, \ldots, \Delta R^{k-1}\right], \Delta R^i := R^{i+1} - R^i$
  $W_k = \left[\Delta \tilde{x}^0, \ldots, \Delta \tilde{x}^{k-1}\right], \Delta \tilde{x}^i := \tilde{x}^{i+1} - \tilde{x}^i$
  decompose $V^k = QU$
  solve $U\alpha = -Q^T R^k$
  $\tilde{s}^k = W_k \alpha$
  $x^{k+1} = \tilde{x}^k + \tilde{s}^k$
end for

from Uekermann, B. W. (2016). *Partitioned fluid-structure interaction on massively parallel systems*

Interface quasi-Newton

Acceleration $\mathscr{A}(\cdot)$

- Coupled problem: $\mathscr{F}(\mathscr{S}(f^k)) = \tilde{f}^k$
- Acceleration: $\mathscr{A}(\tilde{f}^k) = f^{k+1}$
- Short: $\mathscr{H}(x^k) = \tilde{x}^k$ and $x^{k+1} = \mathscr{A}\left(\mathscr{H}(x^k)\right)$
- Underrelaxation: $\mathscr{A}(\tilde{x}^k) = (1-\omega)x^k + \omega\tilde{x}^k$

IQN = Interface quasi-Newton

- solve $\tilde{R}(\tilde{x}) = \tilde{x} - \mathscr{H}^{-1}(\tilde{x}) \stackrel{!}{=} 0$
- Newton step: $\mathscr{A}(\tilde{x}^k) = \tilde{x}^k - \tilde{R}^k / J_{\tilde{R}}^k$
- Jacobian: $V_k / J_{\tilde{R}}^k \approx W_k$ (multisecant equation)

IQN variants in preCICE

- IQN-ILS
  = IQN-inverse least squares
  = Anderson Acceleration
- IQN-IMVJ
  = IQN-inverse multi-vector Jacobian
  = generalized Broyden

Algorithm 2 generalized Broyden

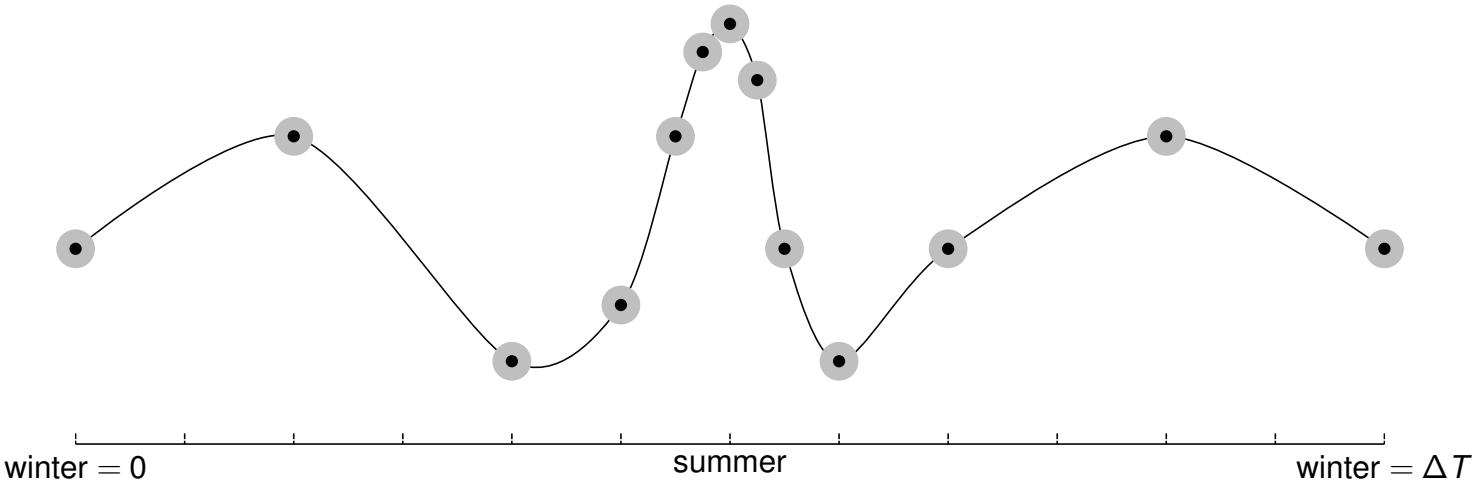
initial value x^0, $J_{\tilde{R}}^{-1,(N)}$ from previous time window
$\tilde{x}^0 = \mathscr{H}(x^0)$ and $R^0 = \tilde{x}^0 - x^0$
$x^1 = x^0 + \omega R^0$
for $k = 1 \ldots$ **do**
  $\tilde{x}^k = \mathscr{H}(x^k)$ and $R^k = \tilde{x}^k - x^k$
  $V_k = \left[\Delta R^0, \ldots, \Delta R^{k-1}\right], \Delta R^i := R^{i+1} - R^i$
  $W_k = \left[\Delta \tilde{x}^0, \ldots, \Delta \tilde{x}^{k-1}\right], \Delta \tilde{x}^i := \tilde{x}^{i+1} - \tilde{x}^i$
  decompose $V^k = QU$
  solve $UZ = Q^T$ for $Z \in \mathbb{R}^{k \times n}$
  $\left[J_{\tilde{R}}^{-1}\right]^k = J_{\tilde{R}}^{-1,(N)} + (W_k - J_{\tilde{R}}^{-1,(N)} V_k)Z$
  $\tilde{s}^k = -\left[J_{\tilde{R}}^{-1}\right]^k W_k \alpha$
  $x^{k+1} = \tilde{x}^k + \tilde{s}^k$
end for

from Uekermann, B. W. (2016). *Partitioned fluid-structure interaction on massively parallel systems*

Interface quasi-Newton

Waveform IQN (preCICE: WIP with Lund University P. Birken & N. Kotarsky)

- x^k holds control points of B-spline $u(t) \rightarrow$ how to build V_k and W_k? (QN-WI and rQN-WI)[1]
- size of x^k and x^{k+1} may differ $\rightarrow$ resampling of x^k onto grid t^{k+1} (already underrelaxation)

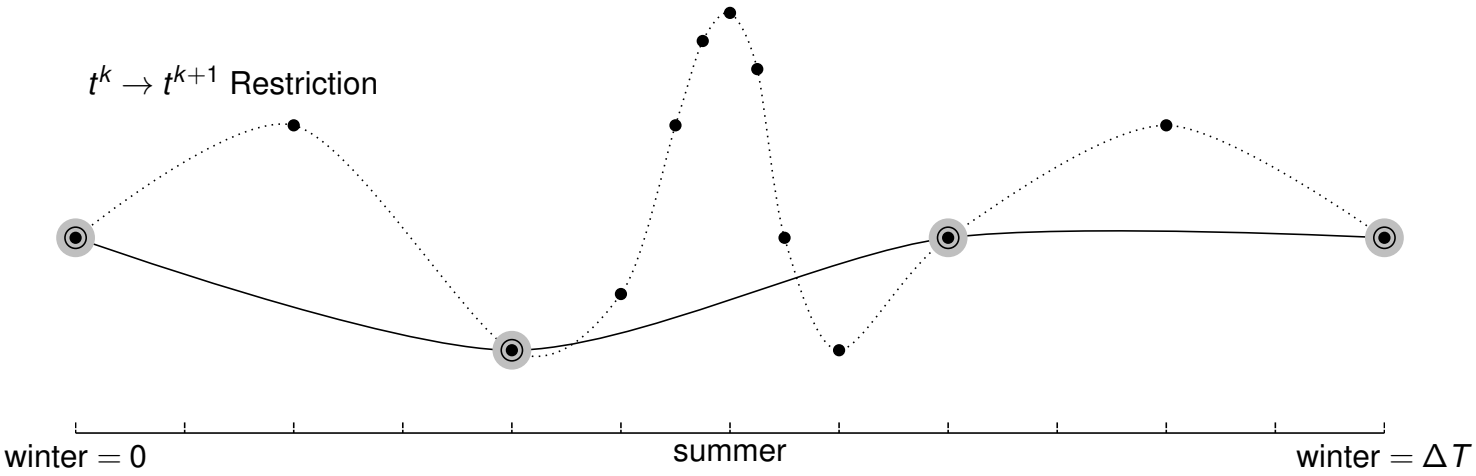


[1] Rüth, B., Uekermann, B., Mehl, M., Birken, P., Monge, A., & Bungartz, H. J. (2021). *Quasi-Newton waveform iteration for partitioned surface-coupled multiphysics applications*

Interface quasi-Newton

Waveform IQN (preCICE: WIP with Lund University P. Birken & N. Kotarsky)

- x^k holds control points of B-spline $u(t) \rightarrow$ how to build V_k and W_k? (QN-WI and rQN-WI)[1]
- size of x^k and x^{k+1} may differ $\rightarrow$ resampling of x^k onto grid t^{k+1} (already underrelaxation)

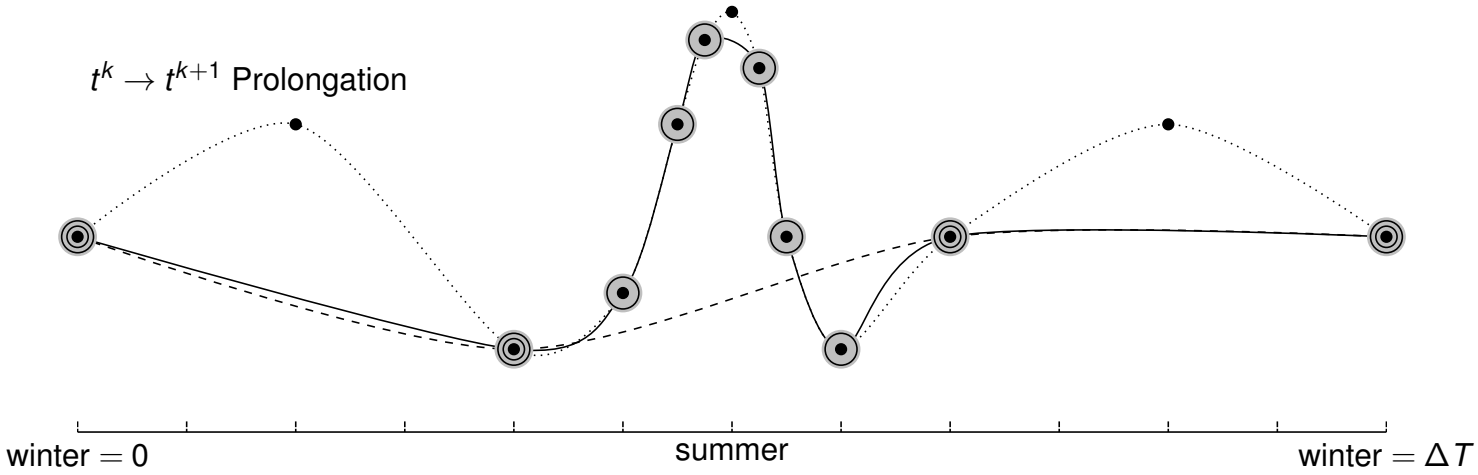


[1] Rüth, B., Uekermann, B., Mehl, M., Birken, P., Monge, A., & Bungartz, H. J. (2021). *Quasi-Newton waveform iteration for partitioned surface-coupled multiphysics applications*

Interface quasi-Newton

Waveform IQN (preCICE: WIP with Lund University P. Birken & N. Kotarsky)

- x^k holds control points of B-spline $u(t)$ $\rightarrow$ how to build V_k and W_k? (QN-WI and rQN-WI)[1]
- size of x^k and x^{k+1} may differ $\rightarrow$ resampling of x^k onto grid t^{k+1} (already underrelaxation)



[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*

Conclusion

State of development

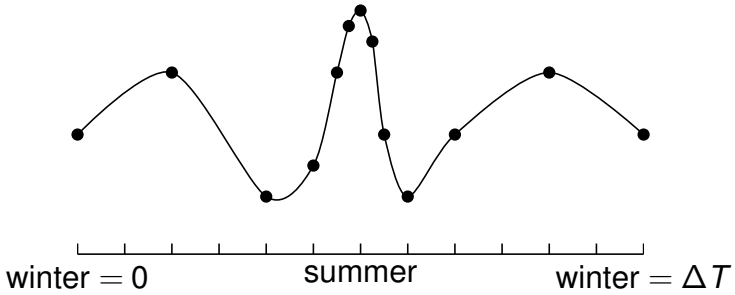
- Multirate time-stepping + black-box + usable API
- Time interpolation new in preCICE v3
- First tutorials use time interpolation

Where to be careful:

- Requirement: High order interpolation needs subcycling
- Restriction: Only simple underrelaxation with substeps, `https://github.com/precice/precice/pull/1834`

Many questions:

- Synchronization, subcycling, and performance?
- Fluid-structure interaction case?
- Adaptivity (inside window / across windows)
- Electric machines?



Community

preCICE Workshop 2023@Munich

Stay in touch?

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

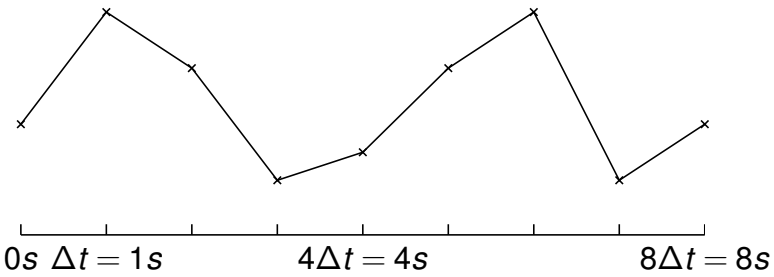
Conferences

- WCCM + preCICE course
  July 2024@Vancouver
- preCICE Workshop
  Sept 2024@Stuttgart
- COUPLED
  2025@Sicily

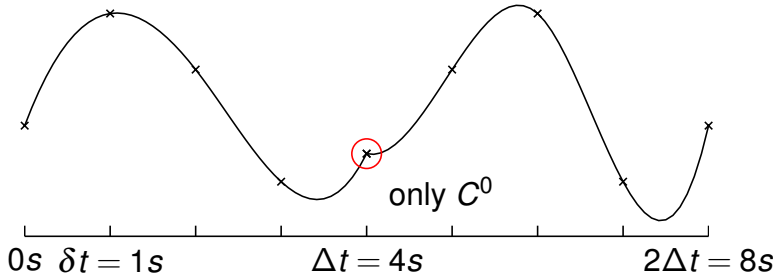
Subcycling

- 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



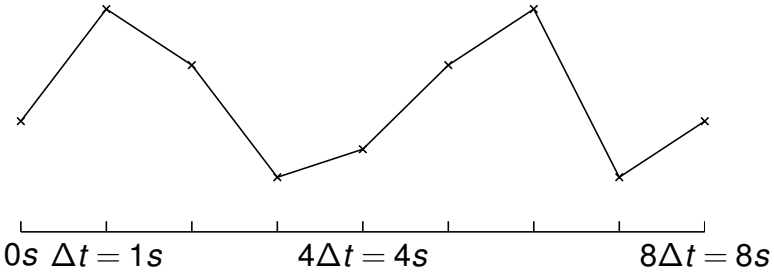
With subcycling (third order BSpline)



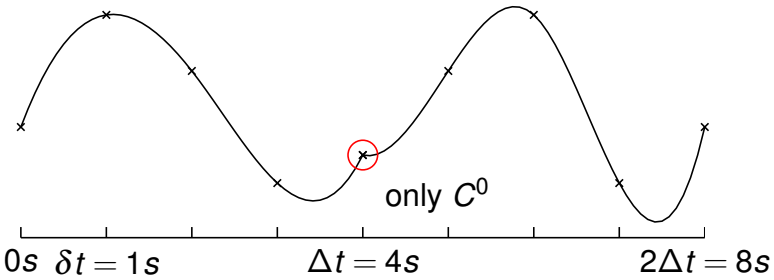
[1] Rüth, B, Uekermann, B, Mehl, M, Birken, P, Monge, A, Bungartz, H-J. Quasi-Newton waveform iteration for partitioned surface-coupled multiphysics applications. Int J Numer Methods Eng. 2021; 122: 5236– 5257. https://doi.org/10.1002/nme.6443

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

Implementation: partitioned heat conduction

Tooling: Higher-order time stepping

- FEniCS for discretization + UFL
- `github.com/firedrakeproject/Irksome`[1]

```
inner(Dt(u), v)*dx + inner(grad(u), grad(v))*dx - inner(rhs, v)*dx
```

- Applied general approach to FEniCS:
  - B.Sc. thesis Nikola Wullenweber monolithic higher-order
  - B.Sc. thesis Niklas Vinnitchenko partitioned higher-order
  - Need derivatives of B-splines!

[1] Farrell, Patrick E., Robert C. Kirby, and Jorge Marchena-Menendez. *Irksome: Automating Runge–Kutta time-stepping for finite element methods*

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
```

$\Delta t \ggg \delta t$