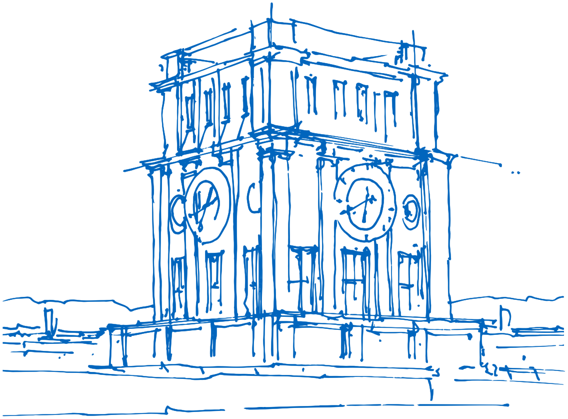
Chair of Scientific Computing
TUM School of Computation, Information and Technology
Technical University of Munich

How to use time interpolation in the preCICE tutorials

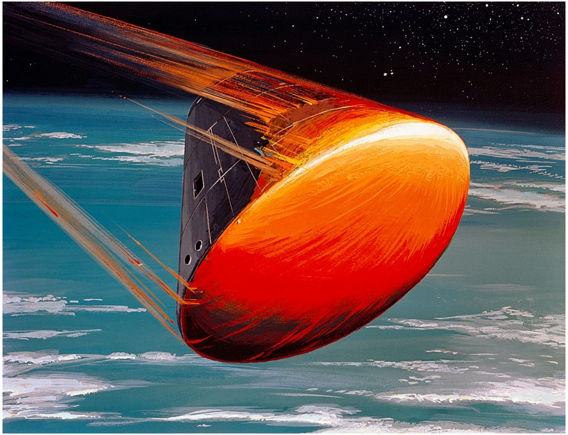
preCICE Workshop 2024, Stuttgart

Benjamin Rodenberg

September 25, 2024

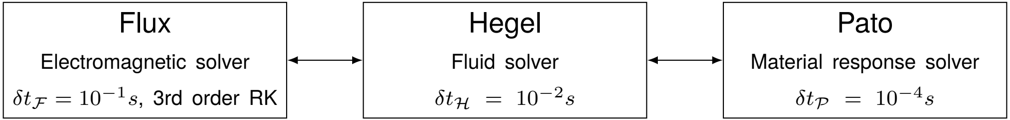


Partitioned solver for inductively coupled plasma wind tunnels



By North American Rockwell, Public Domain, `https://commons.wikimedia.org/w/index.php?curid=2466251`

Partitioned solver for inductively coupled plasma wind tunnels



Flux

Electromagnetic solver

$\delta t_{\mathcal{F}} = 10^{-1}s$, 3rd order RK

Hegel

Fluid solver

$\delta t_{\mathcal{H}} = 10^{-2}s$

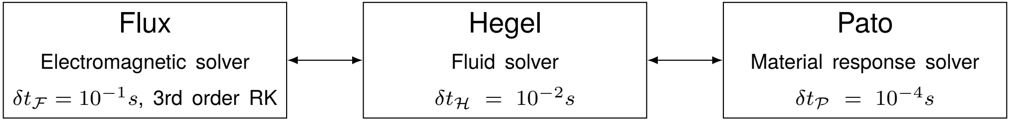
Pato

Material response solver

$\delta t_{\mathcal{P}} = 10^{-4}s$

Setup from *Alessandro Munafò, et al. A Multi-Physics Modeling Framework for Inductively Coupled Plasma Wind Tunnels. 2022.* `https://doi.org/10.2514/6.2022-1011` (Skipping Plato solver)

Partitioned solver for inductively coupled plasma wind tunnels



Flux

Electromagnetic solver

$\delta t_{\mathcal{F}} = 10^{-1} s$, 3rd order RK

Hegel

Fluid solver

$\delta t_{\mathcal{H}} = 10^{-2} s$

Pato

Material response solver

$\delta t_{\mathcal{P}} = 10^{-4} s$

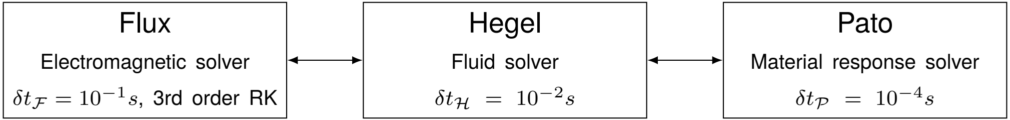
Setup from *Alessandro Munafò, et al. A Multi-Physics Modeling Framework for Inductively Coupled Plasma Wind Tunnels. 2022.* *`https://doi.org/10.2514/6.2022-1011`* (Skipping Plato solver)

Subcycling & multirate

Time **window** size: $\Delta t = 10^{-2} s$

Time **step** size: $\delta t_{\mathcal{F}} \neq \delta t_{\mathcal{H}} \neq \delta t_{\mathcal{P}}$

Partitioned solver for inductively coupled plasma wind tunnels



Flux

Electromagnetic solver

$\delta t_{\mathcal{F}} = 10^{-1} s$, 3rd order RK

Hegel

Fluid solver

$\delta t_{\mathcal{H}} = 10^{-2} s$

Pato

Material response solver

$\delta t_{\mathcal{P}} = 10^{-4} s$

Setup from *Alessandro Munafò, et al. A Multi-Physics Modeling Framework for Inductively Coupled Plasma Wind Tunnels. 2022.* `https://doi.org/10.2514/6.2022-1011` (Skipping Plato solver)

Subcycling & multirate

Time **window** size: $\Delta t = 10^{-2} s$, we compute N time windows

Time **step** size: $\delta t_{\mathcal{F}} \neq \delta t_{\mathcal{H}} \neq \delta t_{\mathcal{P}}$, we do S time steps s.th. $S_i \delta t_i = \Delta t$

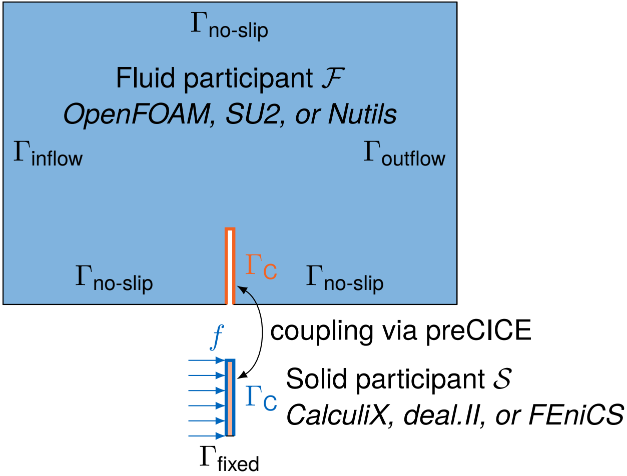
1. Prototype: Perpendicular flap

2. Black-box subcycling

3. Time interpolation feature

4. Tutorials overview
   - `tutorials/oscillator`
   - `tutorials/partitioned-heat-conduction`
   - `tutorials/resonant-circuit`
   - `tutorials/perpendicular-flap`

Prototype: Perpendicular flap



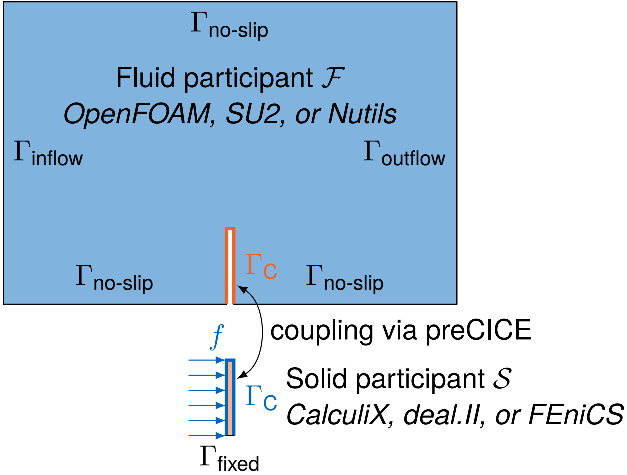
$\Gamma_{\text{no-slip}}$
Fluid participant $\mathcal{F}$
OpenFOAM, SU2, or Nutils
Γ_{inflow}
Γ_{outflow}
$\Gamma_{\text{no-slip}}$
Γ_{C}
$\Gamma_{\text{no-slip}}$
f
coupling via preCICE
Solid participant $\mathcal{S}$
CalculiX, deal.II, or FEniCS
Γ_{C}
Γ_{fixed}

Divide

Conquer

Combine

Prototype: Perpendicular flap



$\Gamma_{\text{no-slip}}$

Fluid participant $\mathcal{F}$

OpenFOAM, SU2, or Nutils

Γ_{inflow} Γ_{outflow}

$\Gamma_{\text{no-slip}}$ Γ_C $\Gamma_{\text{no-slip}}$

f

coupling via preCICE

Solid participant $\mathcal{S}$

CalculiX, deal.II, or FEniCS

Γ_C

Γ_{fixed}

Divide

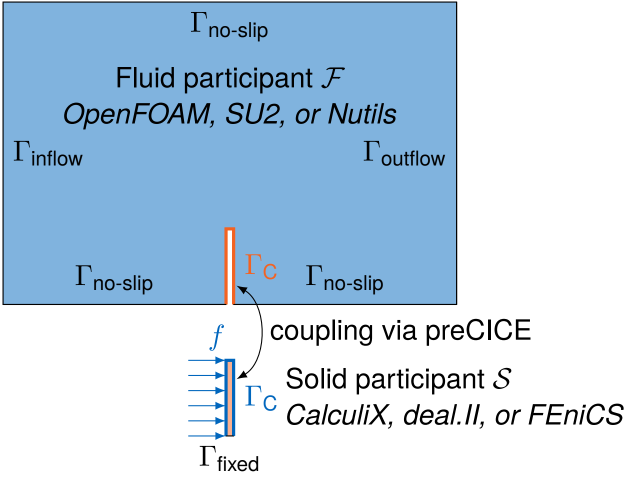
- OpenFOAM $\neq$ FEniCS
- Dirichlet-Neumann

  ($=$ black box)

Conquer

Combine

Prototype: Perpendicular flap


$\Gamma_{\text{no-slip}}$
Γ_{inflow}
Γ_{outflow}
$\Gamma_{\text{no-slip}}$
Γ_C
$\Gamma_{\text{no-slip}}$
f
Γ_C
Γ_{fixed}

Fluid participant $\mathcal{F}$

OpenFOAM, SU2, or Nutils

coupling via preCICE

Solid participant $\mathcal{S}$

CalculiX, deal.II, or FEniCS

Divide

Conquer

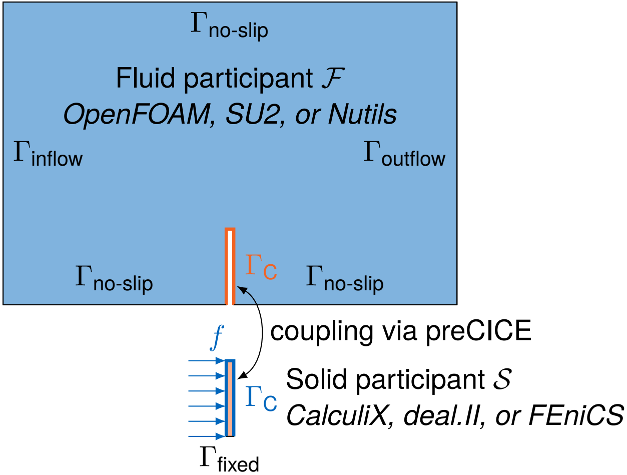
- Fluid: $\mathcal{F}(d) = f$
- Solid: $\mathcal{S}(f) = d$

Boundary response maps

($=$ Poincaré-Steklov operator)

Combine

Prototype: Perpendicular flap



Fluid participant $\mathcal{F}$

OpenFOAM, SU2, or Nutils

coupling via preCICE

Solid participant $\mathcal{S}$

CalculiX, deal.II, or FEniCS

$\Gamma_{\text{no-slip}}$
Γ_{inflow}
Γ_{outflow}
$\Gamma_{\text{no-slip}}$
Γ_C
$\Gamma_{\text{no-slip}}$
f
Γ_C
Γ_{fixed}

Divide

Conquer

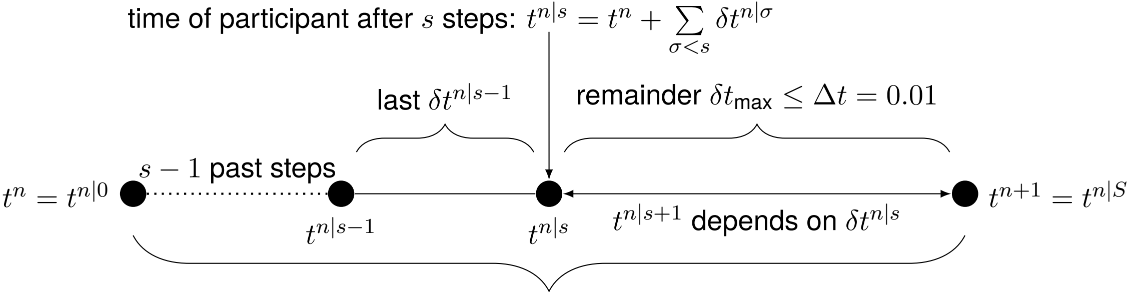
Combine

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

Picard iteration + acceleration

(implicit/tight/two-way coupling)

Black-box subcycling



time of participant after s steps: $t^{n|s} = t^n + \sum_{\sigma<s} \delta t^{n|\sigma}$

last $\delta t^{n|s-1}$

remainder $\delta t_{\text{max}} \leq \Delta t = 0.01$

$s-1$ past steps

$t^n = t^{n|0}$

$t^{n|s-1}$

$t^{n|s}$

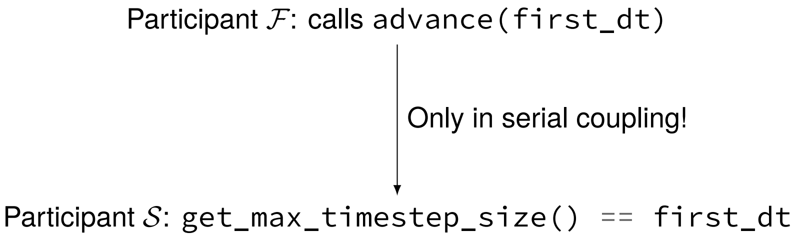
$t^{n|s+1}$ depends on $\delta t^{n|s}$

$t^{n+1} = t^{n|S}$

```
<time-window-size value="0.01" />
```

Superspontaneous slide

What is `<time-window-size method="first-participant"`?



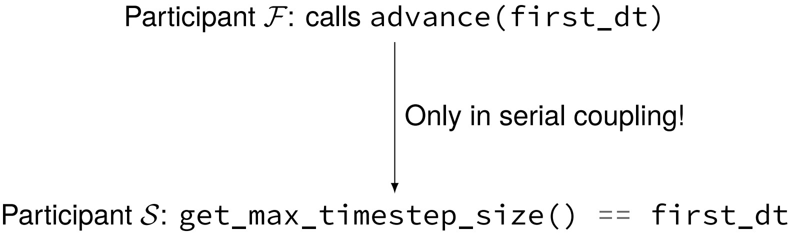
Participant $\mathcal{F}$: calls `advance(first_dt)`

Only in serial coupling!

Participant $\mathcal{S}$: `get_max_timestep_size() == first_dt`

Superspontaneous slide

What is `<time-window-size method="first-participant"`?



Participant $\mathcal{F}$: calls `advance(first_dt)`

Only in serial coupling!

Participant $\mathcal{S}$: `get_max_timestep_size() == first_dt`

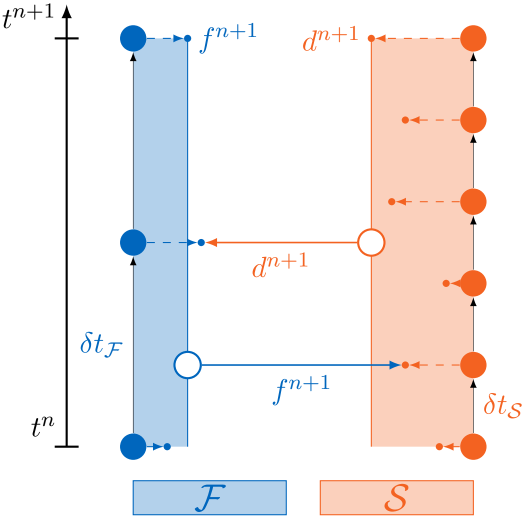
Quiz: What does `get_max_timestep_size()` return for $\mathcal{F}$?

Black-box subcycling

```
participant = precice.Participant("Fluid", "precice-config.xml")
u, t = init()  # initial velocity field $u^0$ at time $t^0$

while participant.is_coupling_ongoing():
  # checkpointing
  ...
  solver_dt = check_CFL_condition(u)  # use adaptive time stepping
  precice_dt = participant.get_max_time_step_size()  # $\delta t_{max} \leq \Delta t = 0.01$, time window size
  dt = np.min([precice_dt, solver_dt])  # $\delta t_{\mathcal{F}}$ for this time step

  # perform time step
  d = participant.read_data("Displ")
  u, t = solve(u, t, d, dt)
  f = compute_forces(u)
  participant.write_data(f, "Force")

  # conclude time step
  participant.advance(dt)  # blocking at end of window
```

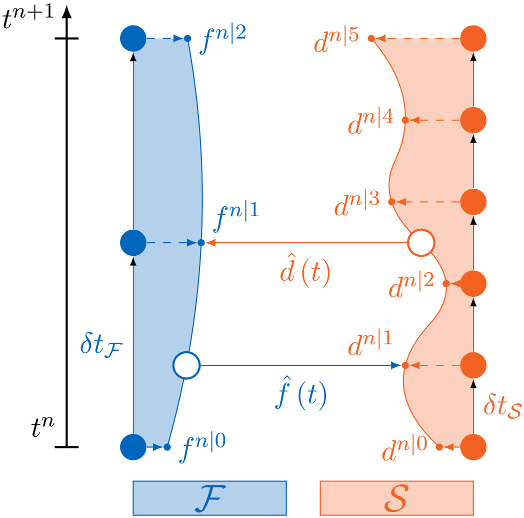
preCICE v2: single-value coupling

```
<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>
```



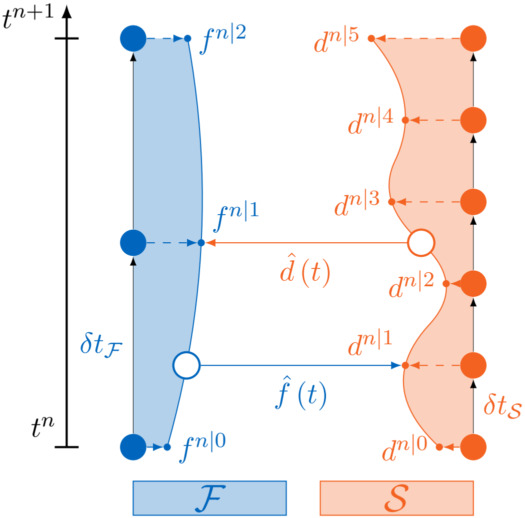
preCICE v3: waveform iteration

$$\begin{tikzcd}
\mathcal{F}\left(\mathcal{S}\left(f^k\right)\right) \overset{+\mathcal{A}}{=} f^{k+1} \arrow[d, "\text{upgrade to preCICE v3}"] \\
\mathcal{F}(\mathcal{S}(\hat{f}(t)^k)) \overset{+\mathcal{A}+\mathcal{I}}{=} \hat{f}(t)^{k+1}
\end{tikzcd}$$



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>
```



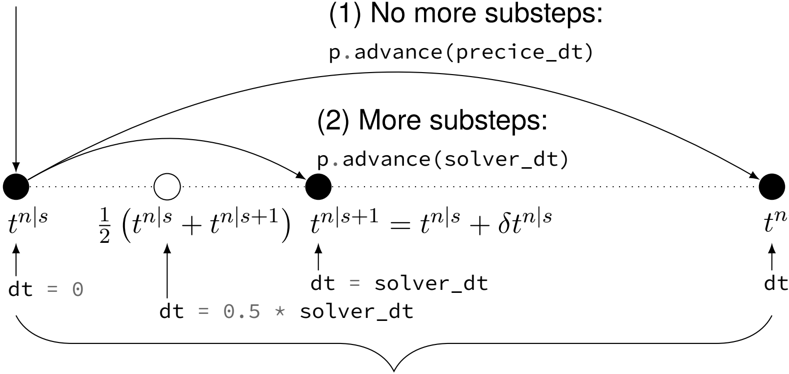
Black-box subcycling + waveform iteration

```python
participant = precice.Participant("Fluid", "precice-config.xml")
u, t = init()  # initial velocity field $u^0$ at time $t^0$

while participant.is_coupling_ongoing():
  # checkpointing
  ...
  solver_dt = check_CFL_condition(u)  # use adaptive time stepping
  precice_dt = participant.get_max_time_step_size()  # $\delta t_{max} \leq \Delta t = 0.01$, time window size
  dt = np.min([precice_dt, solver_dt])  # $\delta t_{\mathcal{F}}$ for this time step

  # perform time step
  d = participant.read_data("Displ", dt)  # <-- reads from waveform $dt \in [0, \delta t_{max}]$
  u, t = solve(u, t, d, dt)
  f = compute_forces(u)
  participant.write_data(f, "Force")

  # conclude time step
  participant.advance(dt)  # <-- buffers substeps
```

Intermediate summary in a figure

Read interpolated data at current time $t = t^{n|s}$:

```
p.read_data(relative_read_time=dt)
```



(1) No more substeps:

```
p.advance(precice_dt)
```

(2) More substeps:

```
p.advance(solver_dt)
```

$t^{n|s}$

$\frac{1}{2}\left(t^{n|s} + t^{n|s+1}\right)$

$t^{n|s+1} = t^{n|s} + \delta t^{n|s}$

$t^{n+1} = t^{n} + \Delta t$

```
dt = 0
```

```
dt = 0.5 * solver_dt
```

```
dt = solver_dt
```

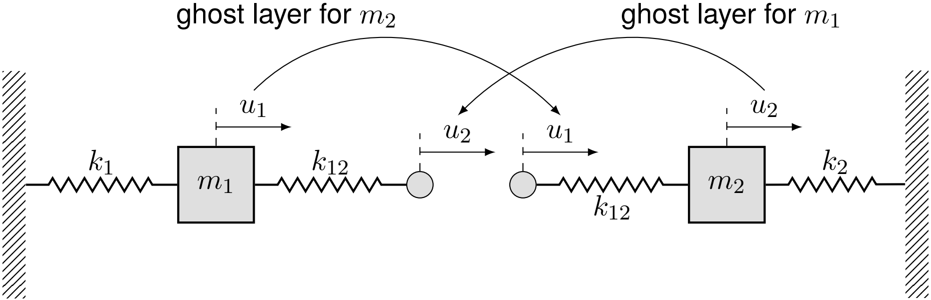
```
dt = precice_dt
```

Complete time window size Δt or $\delta t_{\text{max}} = t^{n+1} - t^{n|s}$

Tutorials overview

- `tutorials/oscillator` (Higher-order & multirate ODE)
- `tutorials/partitioned-heat-equation` (Higher-order PDE)
- `tutorials/resonant-circuit` (use adaptive `solve_ivp` from `scipy.integrate`)
- `tutorials/perpendicular-flap` (Subcycling PDE)

tutorials/oscillator



tutorials/oscillator

ODE System for m_i:

$$g(t,x) = \frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} u_i \\ \dot{u}_i \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k_i}{m_i} & 0 \end{bmatrix} \cdot \begin{bmatrix} u_i \\ \dot{u}_i \end{bmatrix} + \begin{bmatrix} 0 \\ F_{ij}(t) \end{bmatrix} \rightarrow g\left(t,x\right) = \dot{x} = Ax + f(t)$$

solver-python/oscillator.py

```python
precice_dt = participant.get_max_time_step_size()
dt = np.min([precice_dt, my_dt])
def f(t): return participant.read_data(t)
u_new, v_new, a_new = time_stepper.do_step(u, v, a, f, dt)
```

tutorials/oscillator

Classic Runge-Kutta method (RK4)

Right-hand size:

$$\dot{x} = g\left(t, x\right), \text{given } t^n, x^n$$

Butcher tableau:

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} \rightarrow \begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Stages:

$$\begin{aligned} k_1 &= g\left(t^n, x^n\right) \\ k_2 &= g\left(t^n + 1/2\Delta t, x^n + 1/2\Delta t\, k_1\right) \\ k_3 &= g\left(t^n + 1/2\Delta t, x^n + 1/2\Delta t\, k_2\right) \\ k_4 &= g\left(t^n + \Delta t, x^n + \Delta t\, k_3\right) \end{aligned}$$

Update:

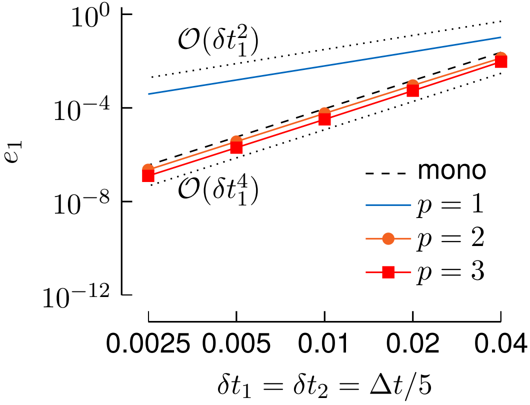
$$x^{n+1} = x^n + \delta t \sum_{i=1}^{s=4} b_i k_i$$

tutorials/oscillator

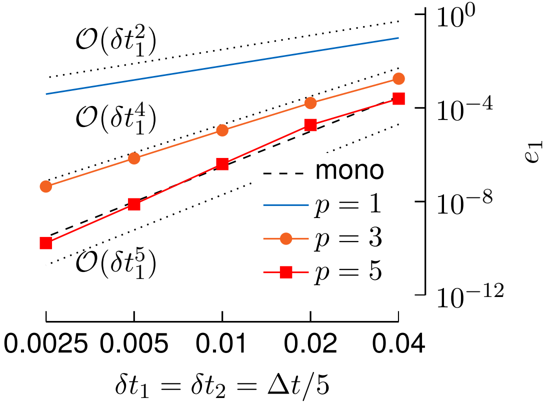
solver-python/timeSteppers.py

```python
    def do_step(self, u0: float, v0: float, a0: float, f: Callable[[float], float],
    ↪   dt: float) -> Tuple[float, float, float]:
        k = 4 * [None]  # store stages in list

        x0 = np.array([u0, v0])
        def g(t, x): return self.ode_system.dot(x) + np.array([0, f(t)])

        k[0] = g(self.c[0] * dt, x0)
        k[1] = g(self.c[1] * dt, x0 + self.a[1, 0] * k[0] * dt)
        k[2] = g(self.c[2] * dt, x0 + self.a[2, 1] * k[1] * dt)
        k[3] = g(self.c[3] * dt, x0 + self.a[3, 2] * k[2] * dt)

        x1 = x0 + dt * sum(b_i * k_i for k_i, b_i in zip(k, self.b))

        return x1[0], x1[1], g(dt, x1)[1]
```

tutorials/oscillator

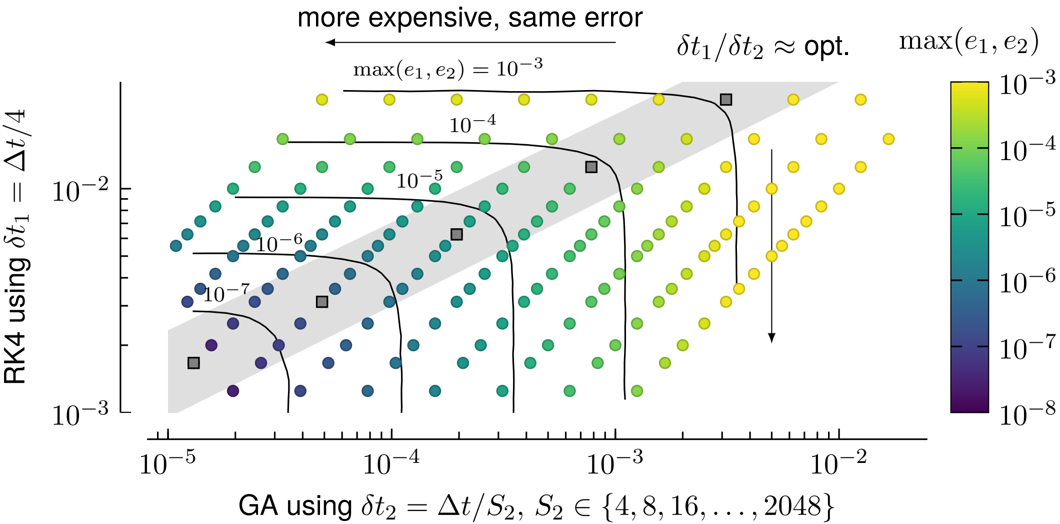


RK4



Radau IIA

tutorials/oscillator



tutorials/partitioned-heat-conduction

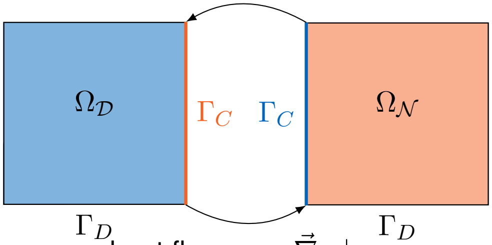
Similar for PDE: `tutorials/blob/develop/partitioned-heat-conduction/solver-fenics/heatHigherOrder.py`. See *Niklas Vinnitchenko. (2024). Evaluation of Higher-Order Coupling Schemes with FEniCS-preCICE. Bachelor's thesis. https://mediatum.ub.tum.de/1732367*

Dirichlet boundary condition on Γ_C for $\mathcal{D}$

temperature u_C

Dirichlet solver $\mathcal{D}$

(FEniCS)



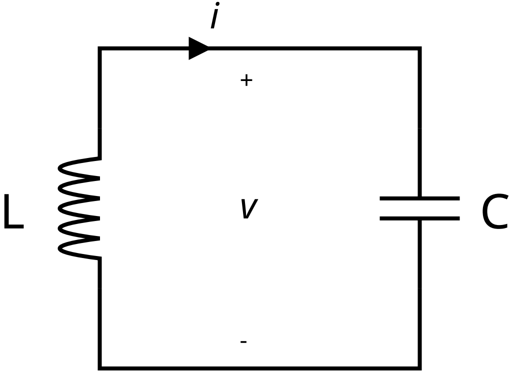
Neumann solver $\mathcal{N}$

(FEniCS)

heat flux $q_C = \vec{\nabla} u \mid_{\Gamma_C}$

Neumann boundary condition on Γ_C for $\mathcal{N}$

tutorials/resonant-circuit



tutorials/resonant-circuit

coil-python/capacitor.py

```python
from scipy.integrate import solve_ivp

def f(t,y):  # Time derivative of U
  I = participant.read_data(t)
  return -I / C

while participant.is_coupling_ongoing():
  ret = solve_ivp(f, [0, dt], U0, method="RK4", dense_output=True, tol=1e-12)
  ts = ret.t
  dense_output = ret.sol

  for i in range(len(ts)):
    U = dense_output((i + 1) * dt / len(ts))
    participant.write_data(np.array(U))
    participant.advance(dt / len(ts))
```

tutorials/perpendicular-flap

Generalized α scheme

Solve elasticity equation:

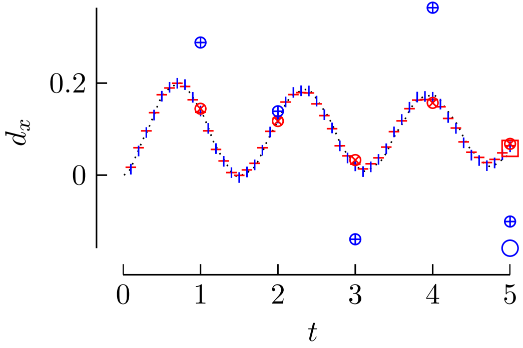
$$M\ddot{u}^{n+1-\alpha_m} + K u^{n+1-\alpha_f} = F\left(t^{n+1-\alpha_f}\right)$$

solid-fenics/solid.py

```
read_data = precice.read_data((1 - float(alpha_f)) * dt)
```

See *Jeremy Bleyer. (2018). Numerical Tours of Computational Mechanics with FEniCS. Zenodo. https://doi.org/10.5281/zenodo.1287832* for details on FEniCS structure solver.

tutorials/perpendicular-flap



	Δt	$\delta t_{\mathcal{F}} = \delta t_{\mathcal{S}}$	substeps
⋯	0.01	0.01	irrelevant
|	0.1	0.01	"false"
–	0.1	0.01	"true"
⊕	1	0.01	"false"
⊗	1	0.01	"true"
○	5	0.01	"false"
□	5	0.01	"true"

Experimental QN `https://github.com/precice/precice/pull/2005` (Niklas Kotarsky)

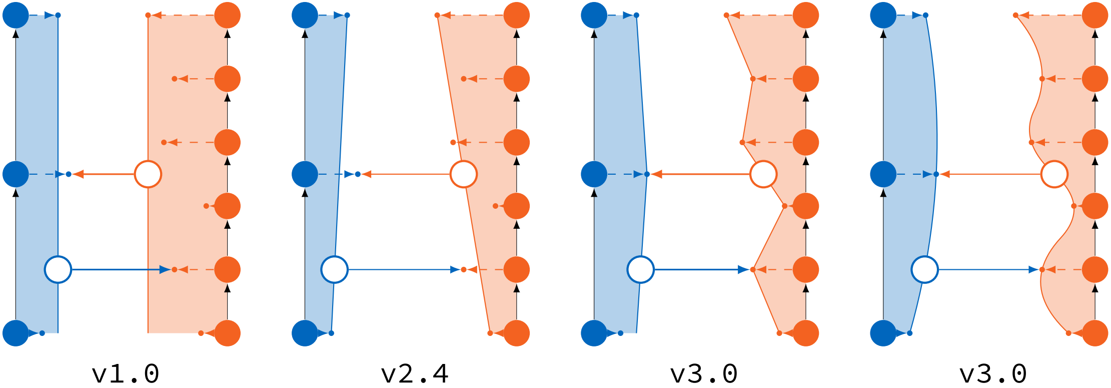
Summary

- Stable API for time interpolation
- Supports higher-order & multirate time stepping
- Tutorials illustrate usage
- **You now know where to look for examples**

Outlook (a.k.a. what will happen tomorrow after lunch)

- Quasi-Newton (`substeps="true"`) & adaptivity → Niklas Kotarsky
- Magnetothermal application & adaptive black-box time stepping → Michael Wiesheu

Bringing waveforms to preCICE


v1.0
v2.4
v3.0
v3.0