

Handling Neumann boundary conditions in WLSQM

September 24, 2021

1 Example: The heat equation

Consider the general initial boundary value problem (IBVP) of the heat equation, with both Dirichlet and Neumann boundary conditions, and an initial condition:

$$\frac{\partial u}{\partial t} = c \nabla \cdot \nabla u \quad \text{in } \Omega \times [0, t_f] , \quad (1)$$

$$u = u_D \quad \text{on } \Gamma_D \times [0, t_f] , \quad (2)$$

$$\hat{n} \cdot \nabla u = q_N \quad \text{on } \Gamma_N \times [0, t_f] , \quad (3)$$

$$u = u_0 \quad \text{on } \Omega \times \{t = 0\} , \quad (4)$$

where $\Omega \subset \mathbb{R}^2$, $\Gamma_D \subset \partial\Omega$, $\Gamma_N \subset \partial\Omega$, and $\Gamma_D \cap \Gamma_N = \emptyset$.

When solving this problem with WLSQM, the main technical issue is implementing the Neumann boundary conditions. The initial condition and the Dirichlet boundary conditions are trivial; one can simply set the function values at the nodes.

In this document, we assume that we can easily obtain a pseudonormal (in the computer graphics sense) for each Neumann boundary point in the discretization. The main issue of interest is then how to insert the information from equation (3) into the discretization.

Consider now the standard (x, y) coordinate system, and a local tangential–normal coordinate system (τ, n) rotated counterclockwise in the xy plane by an angle θ (with the normal direction taken to coincide with the local outer normal of Γ_N). Let

$$c := \cos \theta , \quad s := \sin \theta . \quad (5)$$

We can express the unit vectors of the rotated system in terms of the unit vectors of the standard system by (###TODO: add picture)

$$\hat{\tau} = c\hat{e}_x + s\hat{e}_y , \quad (6)$$

$$\hat{n} = -s\hat{e}_x + c\hat{e}_y . \quad (7)$$

Let us invert (6)–(7). Multiplying the first equation by c and the second by s , we have

$$\begin{aligned} c\hat{\tau} &= c^2\hat{e}_x + cs\hat{e}_y , \\ s\hat{n} &= -s^2\hat{e}_x + cs\hat{e}_y . \end{aligned}$$

Subtracting these, we obtain

$$\hat{e}_x = c\hat{\tau} - s\hat{n} . \quad (8)$$

Similarly, multiplying the first equation by s and the second by c , we have

$$\begin{aligned} s\hat{\tau} &= cs\hat{e}_x + s^2\hat{e}_y , \\ c\hat{n} &= -cs\hat{e}_x + c^2\hat{e}_y , \end{aligned}$$

which leads to

$$\hat{e}_y = s\hat{\tau} + c\hat{n} . \quad (9)$$

Let u be a scalar field and $\mathbf{q} := \nabla u$ its gradient. The gradient itself is invariant with respect to rotations, but its component representations are not. We can express the gradient in either coordinate system, as

$$\mathbf{q} = q_\tau \hat{\boldsymbol{\tau}} + q_n \hat{\mathbf{n}} = q_x \hat{\mathbf{e}}_x + q_y \hat{\mathbf{e}}_y . \quad (10)$$

In WLSQM, when handling Neumann boundary conditions for arbitrarily oriented boundaries, we would like to work in the xy coordinate system, because this would save us the trouble of rotating the coordinate system locally at each boundary point in the discretization.

The coordinate transformations (6)–(7) allow us to write

$$\begin{aligned} \mathbf{q} &= q_\tau \hat{\boldsymbol{\tau}} + q_n \hat{\mathbf{n}} \\ &= q_\tau (c\hat{\mathbf{e}}_x + s\hat{\mathbf{e}}_y) + q_n (-s\hat{\mathbf{e}}_x + c\hat{\mathbf{e}}_y) \\ &= (cq_\tau - sq_n)\hat{\mathbf{e}}_x + (sq_\tau + cq_n)\hat{\mathbf{e}}_y , \end{aligned} \quad (11)$$

whence

$$q_x = cq_\tau - sq_n , \quad q_y = sq_\tau + cq_n . \quad (12)$$

Alternatively, the inverse transformation (8)–(9) gives

$$\begin{aligned} \mathbf{q} &= q_x \hat{\mathbf{e}}_x + q_y \hat{\mathbf{e}}_y \\ &= q_x (c\hat{\boldsymbol{\tau}} - s\hat{\mathbf{n}}) + q_y (s\hat{\boldsymbol{\tau}} + c\hat{\mathbf{n}}) \\ &= (cq_x + sq_y)\hat{\boldsymbol{\tau}} + (-sq_x + cq_y)\hat{\mathbf{n}} , \end{aligned} \quad (13)$$

whence

$$q_\tau = cq_x + sq_y , \quad q_n = -sq_x + cq_y . \quad (14)$$

In the general case, if we use (14) on the second line of (11), we obtain

$$\begin{aligned} \mathbf{q} &= q_\tau (c\hat{\mathbf{e}}_x + s\hat{\mathbf{e}}_y) + q_n (-s\hat{\mathbf{e}}_x + c\hat{\mathbf{e}}_y) \\ &= (cq_x + sq_y)(c\hat{\mathbf{e}}_x + s\hat{\mathbf{e}}_y) + (-sq_x + cq_y)(-s\hat{\mathbf{e}}_x + c\hat{\mathbf{e}}_y) \\ &= (c^2q_x + csq_y)\hat{\mathbf{e}}_x + (csq_x + s^2q_y)\hat{\mathbf{e}}_y + (s^2q_x - csq_y)\hat{\mathbf{e}}_x + (-csq_x + c^2q_y)\hat{\mathbf{e}}_y \\ &= (c^2 + s^2)q_x\hat{\mathbf{e}}_x + (s^2 + c^2)q_y\hat{\mathbf{e}}_y \\ &= q_x\hat{\mathbf{e}}_x + q_y\hat{\mathbf{e}}_y , \end{aligned} \quad (15)$$

which only tells us that the transformations are correct.

A typical Neumann boundary condition is the zero normal gradient condition:

$$(\hat{\mathbf{n}} \cdot \nabla u)|_{\Gamma_N} = (\hat{\mathbf{n}} \cdot \mathbf{q})|_{\Gamma_N} = 0 . \quad (16)$$

On the other hand, in terms of the (τ, n) component representation of $\mathbf{q}$ in (10), we have

$$\hat{\mathbf{n}} \cdot \mathbf{q} = \hat{\mathbf{n}} \cdot (q_\tau \hat{\boldsymbol{\tau}} + q_n \hat{\mathbf{n}}) = q_n . \quad (17)$$

By (16) and (17), on Γ_N it holds that $q_n = 0$. Using this on the second line of (11), we have only the first term on the first line of (15), obtaining

$$\begin{aligned} \mathbf{q}|_{\Gamma_N} &= q_\tau (c\hat{\mathbf{e}}_x + s\hat{\mathbf{e}}_y) \\ &= (cq_x + sq_y)(c\hat{\mathbf{e}}_x + s\hat{\mathbf{e}}_y) \\ &= (c^2q_x + csq_y)\hat{\mathbf{e}}_x + (csq_x + s^2q_y)\hat{\mathbf{e}}_y . \end{aligned} \quad (18)$$

Now because $q_n = 0$, the extra terms do not cancel.

Finally, recall that $\mathbf{q} = \nabla u$, i.e. $q_x = \partial u / \partial x$ and $q_y = \partial u / \partial y$. Thus, using (18), we can write

$$\begin{aligned} (\nabla \cdot \nabla u)|_{\Gamma_N} &= \frac{\partial}{\partial x}(c^2q_x + csq_y) + \frac{\partial}{\partial y}(csq_x + s^2q_y) \\ &= \frac{\partial}{\partial x}(c^2 \frac{\partial u}{\partial x} + cs \frac{\partial u}{\partial y}) + \frac{\partial}{\partial y}(cs \frac{\partial u}{\partial x} + s^2 \frac{\partial u}{\partial y}) \\ &= c^2 \frac{\partial^2 u}{\partial x^2} + 2cs \frac{\partial^2 u}{\partial x \partial y} + s^2 \frac{\partial^2 u}{\partial y^2} . \end{aligned} \quad (19)$$

Equation (19) lets us evaluate — directly in the original (x, y) coordinate system — the right-hand side of (1) at the Neumann boundary points of the discretization, with the boundary condition (16) “baked in”.

(This is similar in spirit to how Neumann boundary conditions are handled in the finite element method; the PDE itself is enforced also for the Neumann boundary degrees of freedom, with additional data from the boundary condition inserted.)

Note that physically, timestepping (1) with (19) inserted into the RHS will smooth out curvature only in the $\hat{\tau}$ direction (because (18) forces $\mathbf{q} = \nabla u$ to be aligned with $\hat{\tau}$), ignoring any curvature near the boundary along the $\hat{n}$ direction — which is exactly how a temperature field should behave at a Neumann (heat-insulating) boundary.

For nonzero prescribed $q_n = q_N$:

$$\begin{aligned} \mathbf{q}|_{\Gamma_N} &= q_\tau(c\hat{\mathbf{e}}_x + s\hat{\mathbf{e}}_y) + q_n(-s\hat{\mathbf{e}}_x + c\hat{\mathbf{e}}_y) \\ &= (cq_x + sq_y)(c\hat{\mathbf{e}}_x + s\hat{\mathbf{e}}_y) + q_n(-s\hat{\mathbf{e}}_x + c\hat{\mathbf{e}}_y) \\ &= (c^2q_x + csq_y)\hat{\mathbf{e}}_x + (csq_x + s^2q_y)\hat{\mathbf{e}}_y + q_n(-s\hat{\mathbf{e}}_x + c\hat{\mathbf{e}}_y) \\ &= (c^2q_x + csq_y - sq_N)\hat{\mathbf{e}}_x + (csq_x + s^2q_y + cq_N)\hat{\mathbf{e}}_y. \end{aligned} \quad (20)$$

Again, $\mathbf{q} = \nabla u$, i.e. $q_x = \partial u / \partial x$ and $q_y = \partial u / \partial y$, so we obtain

$$\begin{aligned} (\nabla \cdot \nabla u)|_{\Gamma_N} &= \frac{\partial}{\partial x}(c^2q_x + csq_y - sq_N) + \frac{\partial}{\partial y}(csq_x + s^2q_y + cq_N) \\ &= \frac{\partial}{\partial x}(c^2\frac{\partial u}{\partial x} + cs\frac{\partial u}{\partial y} - sq_N) + \frac{\partial}{\partial y}(cs\frac{\partial u}{\partial x} + s^2\frac{\partial u}{\partial y} + cq_N) \\ &= c^2\frac{\partial^2 u}{\partial x^2} + 2cs\frac{\partial^2 u}{\partial x \partial y} + s^2\frac{\partial^2 u}{\partial y^2} + \left(-s\frac{\partial}{\partial x} + c\frac{\partial}{\partial y}\right)q_N \\ &= c^2\frac{\partial^2 u}{\partial x^2} + 2cs\frac{\partial^2 u}{\partial x \partial y} + s^2\frac{\partial^2 u}{\partial y^2} + \hat{\mathbf{n}} \cdot \nabla q_N. \end{aligned} \quad (21)$$

If we have a prescribed value for $\hat{\mathbf{n}} \cdot \nabla q_N$, we can enforce it using (21).

2 Application to Euler flow

Similar tricks can be used to treat boundary conditions in other IBVPs. For Euler flow, the *no-penetration* boundary condition is similar to the Neumann condition of the heat equation in that the normal component of a quantity is known (zero!), but the tangential component remains free (to be computed from the PDE).

In Euler flow, instead of $\mathbf{q}$, we have the vector-valued velocity field $\mathbf{u}$:

$$\hat{\mathbf{n}} \cdot \mathbf{u} = 0.$$

Let

$$\mathbf{u} = u_\tau \hat{\boldsymbol{\tau}} + u_n \hat{\mathbf{n}} = u_x \hat{\mathbf{e}}_x + u_y \hat{\mathbf{e}}_y.$$

By the above coordinate transformation considerations, we can write

$$\mathbf{u} = u_\tau(c\hat{\mathbf{e}}_x + s\hat{\mathbf{e}}_y) + u_n(-s\hat{\mathbf{e}}_x + c\hat{\mathbf{e}}_y).$$

Again, similarly to (18), at the no-penetration boundary Γ_N we have

$$\begin{aligned} \mathbf{u}|_{\Gamma_N} &= u_\tau(c\hat{\mathbf{e}}_x + s\hat{\mathbf{e}}_y) \\ &= (cu_x + su_y)(c\hat{\mathbf{e}}_x + s\hat{\mathbf{e}}_y) \\ &= (c^2u_x + csu_y)\hat{\mathbf{e}}_x + (csu_x + s^2u_y)\hat{\mathbf{e}}_y. \end{aligned}$$

Here u_x and u_y are unknowns, to be updated based on the PDE. The additional condition that connects u_x and u_y , representing $u_n = 0$, is given by equation (14) as

$$-su_x + cu_y = 0. \quad (22)$$

###TODO: do this properly:

- write out the Euler flow PDE (for velocity) on Γ_N
- expand in the (τ, n) system, writing in terms of $u_\tau, u_n, \hat{\tau}, \hat{n}$
- use $u_n = 0$, this eliminates a few terms
- express remaining stuff in terms of $u_x, u_y, \hat{e}_x, \hat{e}_y$ (as was done for the heat equation above)
- obtain velocity PDE that holds on Γ_N (note: one equation only, the other is (22))
- do the same stuff for the density equation

2.1 Euler flow

Compressible flow. Time evolution equations for fluid density and flow velocity, in an Eulerian frame, in index notation:

$$(\partial_t + u_j \partial_j) \rho = -\rho \partial_j u_j \quad (23)$$

$$(\partial_t + u_j \partial_j) u_i = -\frac{\alpha}{\rho} \partial_i \rho + g_i \quad (24)$$

We may also write this in direct notation:

$$\left(\frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla \right) \rho = -\rho \nabla \cdot \mathbf{u} \quad (25)$$

$$\left(\frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla \right) \mathbf{u} = -\frac{\alpha}{\rho} \nabla \rho + \mathbf{g} \quad (26)$$

To keep WLSQM fast (to avoid the need to re-generate neighborhoods with upwinding), we use operator splitting (second-order symmetrized Strang) to separate the transport into a subproblem that can be handled by a specialized algorithm.

Note that in the splitting, each subproblem gets its own copy of the $\partial/\partial t$ term; one cannot simply sum the subproblems to get the original problem.

Subproblems for fluid density ρ :

$$\frac{\partial \rho}{\partial t} = -\rho \nabla \cdot \mathbf{u} \quad (27)$$

$$\left(\frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla \right) \rho = 0 \quad (28)$$

Subproblems for (Eulerian) flow velocity $\mathbf{u}$:

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{\alpha}{\rho} \nabla \rho + \mathbf{g} \quad (29)$$

$$\left(\frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla \right) \mathbf{u} = 0 \quad (30)$$

For both ρ and $\mathbf{u}$, the second subproblem is pure advection at velocity $\mathbf{u}$.

The advection subproblem for $\mathbf{u}$, equation (30), is fully decoupled; the field $\mathbf{u}$ transports itself, with ρ having no effect. In the advection problem for ρ , equation (28), there is one-way coupling; the field $\mathbf{u}$ transports the field ρ .

The other two subproblems, equations (27) and (29), are bidirectionally coupled in $(\rho, \mathbf{u})$. Equation (27) accounts for changes in density caused by sources and sinks in the flow velocity field, while equation (29) accounts for local acceleration due to body forces (such as gravity) and the pressure response.

Note that $(1/\rho) \nabla \rho \equiv \nabla(\ln \rho)$, but the left-hand side is more convenient for the solver. Note also that α is large (for air, on the order of 10^5); the pressure response of the velocity field is thus very fast when compared to the advection. (Physically: compare typical flow velocities with the speed of sound.) Thus, this is a stiff problem, where the pressure response term limits the maximum possible timestep size.

This suggests that we may internally use a smaller timestep for the stiff coupled subproblems, and then use a single large timestep for the advection subproblems (for numerical stability, pure advection only needs $CFL < 1$).

The stiff coupled part requires solving for $\nabla \cdot \mathbf{u}$ and $\nabla \rho$ at each small timestep (via WLSQM), but global model interpolation is only needed by the advection part.

From the viewpoint of information flow, equation (27) has no preferred direction in space, so symmetric (central) neighborhoods in WLSQM are expected to work well.

Likewise, there should be no preferred direction of information flow in equation (29), so symmetric neighborhoods (taking the gradient centrally) should work also there.

Only in the advection subproblems (28) and (30), the differential operator has a preferred direction, along streamlines. Moreover, the combination $\partial/\partial t + \mathbf{u} \cdot \nabla$ makes this preferred direction *asymmetric in time*. In physical terms, this operator transports a field, by a velocity field $\mathbf{u}$.

Hence, when updating an Eulerian description of the target field $\partial/\partial t + \mathbf{u} \cdot \nabla$ operates on, one should only look at information that is *yet to arrive* to the point under consideration; information that has *already passed* the point should have no effect on the update.

- for Euler flows, *a priori* there is not much difference between stationary obstacles and axially moving obstacles, because the tangential obstacle BC is of the perfect-slip type.
- but also *a priori*, from the viewpoint of the structure model, the addition of the nonlinear term into the flow model (compared with potential flow) may change the pressure field drastically.
 - indeed, the velocity fields of Euler flows typically look completely different from those of potential flows. Although the Euler model neglects viscosity, it already captures the asymmetry between upstream and downstream (which comes from the advection term) that is an important feature of actual physical flows.
- the solver needs a way to save/load flow fields to provide for custom ICs
- practical notes on using the WLSQM solver for this:
 - make a generic advection solver, use that as a component
 - x_i doesn't have to be one of the x_k ; it can be chosen freely
 - the problem instances are fully independent, so even in the same problem set, *each instance can be expressed in a different coordinate system*
 - * we can thus use the local (τ, n) system for each x_i on a boundary
 - if the obstacle does not rotate (or undergoes only small vibrations), the point clouds for the boundary problems can be preprocessed (to generate the matrices)
 - in the case of small vibrations, use the reference configuration
 - * from the solution of the boundary problems, we really only need f_i
 - from the “fi” array returned by the solver, copy just the function value into (the appropriate position in) the global array holding the function values at **all** discretization nodes
 - the derivatives in the “fi” array are expressed in the “wrong” coordinate system (local (τ, n) instead of (x, y))
 - we *could* convert if needed. It's just a rotation; the gradient itself is invariant although its components are not...
 - ...but we don't really need the gradient info at the boundary nodes (unless we want to visualize it)
 - set up the inside-domain nodes the usual way
 - * the sets x_k may contain boundary nodes
 - * f_k needs only function value data, this is available directly also for boundary nodes
 - set up the boundary nodes so that each problem instance has its x_i on the boundary, and **all** x_k inside the domain
 - * the sets x_k contain no boundary nodes
 - * use the local (τ, n) coordinate system to express x_k ; we can choose $x_i = (0, 0)$ for each problem instance

- at each timestep: **###NO, don't use this algorithm — correct algorithm provided further below. But the general notes here are fine.**
 - * first update advection for the inside-domain DOFs
 - on the RHS, both the domain and the boundaries use **old** data, as they should in an explicit time-integration scheme
 - use semi-Lagrangian advection via interpolation by the global surrogate model
 - the global model is generated using the local models for only the inside-domain DOFs, but the boundary information is fed there via the x_k
 - thus the global model optimally approximates the boundary data (in the least-squares sense), so the interpolation is correct also near boundaries
 - in the streamline traceback, must check for ray intersections with boundary segments; if hit, copy the value from that point (or resample and proceed upstream for the remaining time)
 - in MacCormack correction, in the clamping step for numerical stabilization, take the min and max of the old data over all f_k from the (pre-computed) neighborhood of the node whose model the (“nearest”-type) interpolator picked. (The interpolator must return this information for each point $x[m]$ in the output array.) This saves a relatively expensive search for the k closest nodes from the point at which the interpolant was computed.
 - * then update the boundary values with $\partial f / \partial n$ as known (from BC), f as unknown
 - at this step, the data inside the domain has already been updated, so **all** f_k for the boundary update are **new** data, as they should!
 - this takes advantage of the local coordinate systems so that locally, “ $\partial f / \partial y$ ” is known — this is supported by the solver
 - this approach satisfies the boundary condition consistently at each value of t seen by the discretization.
- BCs: the advection subproblems (28) and (30) for both ρ and $\mathbf{u}$ require (only) inflow BCs.
 - Obstacles must be handled sensibly. When the streamline traceback ray hits an obstacle, stop the traceback, and sample the velocity field at that point. Either return that value (fast), or continue in the new upstream direction for the remaining time of the trace (maybe more accurate). Repeat until trace complete; sample the velocity field at the final point and return that value.
 - In the streamline traceback, we can treat the whole area outside the bounding box of the computational region as free stream, with $\mathbf{u} = \mathbf{u}_\infty$ and $\rho = \rho_\infty$. This way we can dynamically handle parts of the outer boundary that may alternate between in- and outflow due to oscillating flow patterns inside the domain.
 - Obviously, any information that exits the computational domain is lost. The purpose of the previous point is to provide sensible “dummy data” for any inflow occurring at the edges, provided that the computational domain is large enough to capture the actually interesting behavior.
- IC for $\mathbf{u}$:
 - In order for the original problem to make physical sense, we must arbitrarily choose **some** valid flow field to begin with.
 - * Due to how semi-Lagrangian advection works, we cannot initialize the flow velocity to zero inside the domain.
 - The streamline traceback of SLA is based on the flow velocity *at the node to be updated*. If the velocity there is zero, SLA sees the situation as no flow occurring **to** that point.
 - Considering the analytical properties of inviscid flows, this is actually somewhat sensible (an inviscid flow initially at rest remains at rest; see Lighthill's book).
 - * Thus we need a nonzero flow field to use as an IC.
 - * Because Euler flow is nonlinear, the safest option is to start with a physically valid flow field.
 - * OTOH, a zero IC for $\mathbf{u}$ *might* also work, since beside the advection subproblem, there is also the pressure response subproblem that **will** see the inflow. The kinematically forced inflow at the boundary will cause $\nabla \cdot \mathbf{u} < 0$ in the first inside-domain layer of nodes (because $\partial u_x / \partial x < 0$),

causing $\partial\rho/\partial t > 0$ there. Then, at the next timestep, in the third layer of nodes (on the other side of the density peak) $\partial\rho/\partial x < 0$, whence $u_x > 0$. This should cause a compressive pressure wave to propagate across the domain, causing the flow velocity to become nonzero, thus starting up the advection. Try this first.

- To bootstrap, we solve a steady-state potential flow problem, $\mathbf{u} = \nabla\phi$, with $\nabla \cdot \nabla\phi = 0$. We can use WLSQM via the heat equation:

$$\frac{\partial\phi}{\partial t} + \nabla \cdot \nabla\phi = 0$$

- At all outer boundaries of the domain, use the free-stream BC: $\hat{\mathbf{n}} \cdot \mathbf{u} = \hat{\mathbf{n}} \cdot \nabla\phi = \hat{\mathbf{n}} \cdot \mathbf{u}_\infty$.
 - * In practice, at left and right, $u_x = u_\infty$; at top and bottom, $u_y = 0$.
 - * But we actually must prescribe $\partial\phi/\partial n$. The outer edges are aligned with the coordinate axes, so just use $\partial\phi/\partial x = u_\infty$ (left and right); $\partial\phi/\partial y = 0$ (top and bottom).
 - * At the boundary, leave ϕ as unknown and let WLSQM compute its value.
- At obstacle boundaries, use the no-penetration BC: $\hat{\mathbf{n}} \cdot \mathbf{u} = \hat{\mathbf{n}} \cdot \nabla\phi = 0$.
 - * Use local (τ, n) coordinate systems and set up WLSQM with “ $\partial\phi/\partial y = 0$ ”.
- IC for the bootstrap problem: uniform flow field ignoring obstacles, $\mathbf{u} = \mathbf{u}_\infty$, where $\mathbf{u}_\infty = (cu_\infty, su_\infty)$; i.e. $\phi(x, y) = cu_\infty x + su_\infty y$. The problem is numerically easy, so this should rather quickly converge to the final state (that accounts for the obstacles).
- The final state of the bootstrap problem is a steady-state approximation for incompressible and irrotational flow. We assume that our flow is “close enough” to incompressible so that this approximation is physically at least somewhat sensible to use as an initial state for the original problem.
- This model also already gives us a lift prediction, if we want one.
- In the paper vibration case, the undisturbed obstacle geometry is flat and the flow is axial, so we can actually skip this bootstrap step and just directly use a uniform free-stream flow field as the IC for the original problem. But in the general case (e.g. flow around a cylinder; better for testing the flow solver component), we need to bootstrap if we want a nonzero initial flowfield.

- IC for ρ :

- First, note that in a steady-state ($\partial\mathbf{u}/\partial t \equiv 0$) incompressible flow in the absence of body forces, velocity along each streamline is constant (Bernoulli’s principle, https://en.wikipedia.org/wiki/Bernoulli%27s_principle), $(\mathbf{u} \cdot \nabla)\mathbf{u} \equiv 0$.
- A steady state for ρ obviously exists. From equation (25), taking $\partial\rho/\partial t \rightarrow 0$, we have

$$\mathbf{u} \cdot \nabla\rho = -\rho \nabla \cdot \mathbf{u} .$$

Alternatively, we may write this as

$$\mathbf{u} \cdot \left(\frac{1}{\rho} \nabla\rho \right) = -\nabla \cdot \mathbf{u} ,$$

or

$$(\mathbf{u} \cdot \nabla)(\ln\rho) = -\nabla \cdot \mathbf{u} .$$

In the absence of body forces, potential flow is divergence-free, $\nabla \cdot \mathbf{u} \equiv 0$, so $(\mathbf{u} \cdot \nabla)(\ln\rho) = 0$. In this situation, *density remains constant along streamlines*.

- From (26), using $\partial\mathbf{u}/\partial t \equiv 0$ and $(\mathbf{u} \cdot \nabla)\mathbf{u} \equiv 0$, we have

$$0 = \mathbf{g} - \frac{\alpha}{\rho} \nabla\rho ,$$

whence

$$\frac{1}{\alpha} \mathbf{g} = \frac{1}{\rho} \nabla\rho = \nabla(\ln\rho) .$$

We thus obtain $\nabla(\ln\rho)$ algebraically.

- If $\mathbf{g} \equiv 0$, we can choose $\ln\rho = \text{const.}$ and hence $\rho = \text{const.}$ as the corresponding density IC, as is indeed expected for incompressible flow.

- If $\mathbf{g} \neq 0$, let us assume that the gravitational field $\mathbf{g}$ is uniform. We can start from the “downmost” point in the domain, set a value for ρ there, and then linearly increase $\ln \rho$ moving “upward” in the gravitational field.
 - * Doing this accounts for hydrostatic pressure. Maybe not necessary to include nonzero $\mathbf{g}$ in our application?
 - * This is not completely rigorous; taking $(\mathbf{u} \cdot \nabla)\mathbf{u} \equiv 0$ we have already assumed the absence of body forces. Thus this is an ad hoc approximation, applicable when the body forces are “small”.
- Summary: for $\mathbf{g} \equiv 0$, the IC for ρ is $\rho = \text{const}$.
- BCs: physics dictates $\hat{\mathbf{n}} \cdot (\mathbf{u} - \mathbf{u}_{\text{obst}}) \equiv 0$ (for all t) at obstacles.

- For simplicity, let us first consider stationary obstacles, with $\mathbf{u}_{\text{obst}} \equiv 0$, and $\hat{\boldsymbol{\tau}}, \hat{\mathbf{n}}$ constant in time. Consider the coupled subproblems. Let us start with the pressure subproblem for $\mathbf{u}$, equation (29), written at the obstacle boundary:

$$\frac{\partial \mathbf{u}}{\partial t}|_{\Gamma_N} = \mathbf{g} - \frac{\alpha}{\rho} \nabla \rho .$$

Since $u_n \equiv 0$ along the whole boundary, also $\partial_\tau u_n \equiv 0$, and we have

$$\begin{aligned} \frac{\partial u_\tau}{\partial t}|_{\Gamma_N} &= g_\tau - \frac{\alpha}{\rho} \hat{\boldsymbol{\tau}} \cdot \nabla \rho , \\ 0 &= \frac{\partial u_n}{\partial t}|_{\Gamma_N} = g_n - \frac{\alpha}{\rho} \hat{\mathbf{n}} \cdot \nabla \rho . \end{aligned}$$

The first equation gives an update formula for u_τ at the boundary, whereas the second gives (a trivial update for u_n and) a constraint that must be satisfied by $\hat{\mathbf{n}} \cdot \nabla \rho$. If $\mathbf{g} \equiv 0$, the constraint simplifies into

$$(\hat{\mathbf{n}} \cdot \nabla \rho)|_{\Gamma_N} = 0 .$$

- * All these relations hold at the beginning of the timestep, $t = t_0$.
- * In practice, to mitigate discretization error (especially at curved boundaries), on input it is preferable to use a trick: take $u_\tau = \text{sgn}(\hat{\boldsymbol{\tau}} \cdot \mathbf{u}) |u|$ and $u_n = 0$ (instead of applying coordinate system conversion to the actual (u_x, u_y) representation).
- * If the representation of $\mathbf{u}$ at the boundaries is stored in (τ, n) coordinates, this input conversion is equivalent to a do-nothing.
- * On output, use the updated u_τ^* , and set $u_n^* = 0$, before applying conversion to (x, y) coordinates (now using the coordinate conversion formulas).
- * When updating u_τ , we evaluate $\hat{\boldsymbol{\tau}} \cdot \nabla \rho$ by WLSQM. At the boundary, $\hat{\boldsymbol{\tau}} \cdot \nabla \rho$ is conveniently available in the local (τ, n) representation.
- * To prescribe $\partial \rho / \partial n = 0$, use local (τ, n) coordinates for the boundary nodes (also for the density field), which allows setting “ $\partial \rho / \partial y = 0$ ” in the solver. WLSQM then outputs the appropriate value for ρ that makes the field satisfy $\partial \rho / \partial n = 0$ at the boundary.
- Just to be sure, consider also the source/sink subproblem for ρ at the obstacle boundary:

$$\frac{\partial \rho}{\partial t}|_{\Gamma_N} = -\rho \nabla \cdot \mathbf{u} = -\rho \left(\frac{\partial u_\tau}{\partial \tau} + \frac{\partial u_n}{\partial n} \right)$$

The (physically appropriate) BCs tell us nothing about the components of $\nabla \cdot \mathbf{u}$; thus this equation produces no new constraints.

- * This also holds in the case of moving obstacles.
- Summary: for the coupled subproblems, with $\mathbf{g} \equiv 0$, the obstacle boundary conditions **for stationary obstacles** are

$$\begin{aligned} \frac{\partial u_\tau}{\partial t}|_{\Gamma_N} &= -\frac{\alpha}{\rho} \hat{\boldsymbol{\tau}} \cdot \nabla \rho , \\ u_n &= 0 , \\ (\hat{\mathbf{n}} \cdot \nabla \rho)|_{\Gamma_N} &= 0 . \end{aligned}$$

- * In fact, if the second and third equation are applied first, the first equation is then just the PDE itself (the coupled subproblem for $\mathbf{u}$, equation (29)), written at the obstacle boundary.
 - * The second equation is the original BC for $\mathbf{u}$.
 - * The third equation is a consequence of $u_n = \text{const.}$ (whence $\partial u_n / \partial t = 0$) and the PDE (29).
 - * In the solver, at the boundary, $\hat{\tau} \cdot \nabla \rho$ is conveniently available in the local (τ, n) representation.
- Conversion of vectors from (τ, n) components to (x, y) components.
- * Once we have the new tangential velocity u_τ^* , in order to obtain the new u_x^* and u_y^* , we have the linear equation system

$$\begin{aligned} cu_x^* + su_y^* &= u_\tau^* , \\ -su_x^* + cu_y^* &= u_n^* , \end{aligned}$$

where $u_n^* = 0$. In matrix form,

$$\mathbf{A}\mathbf{x} = \mathbf{b} ,$$

where

$$\mathbf{A} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} , \quad \mathbf{x} = \begin{bmatrix} u_x^* \\ u_y^* \end{bmatrix} , \quad \mathbf{b} = \begin{bmatrix} u_\tau^* \\ u_n^* \end{bmatrix}$$

Taking note that since $\mathbf{A}$ represents a rotation, it is orthogonal, $\mathbf{A}^{-1} = \mathbf{A}^T$:

$$\mathbf{A}^T \mathbf{A} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} c^2 + s^2 & cs - sc \\ sc - cs & s^2 + c^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} ,$$

so in general,

$$\begin{bmatrix} u_x^* \\ u_y^* \end{bmatrix} = \mathbf{x} = \mathbf{A}^{-1} \mathbf{b} = \mathbf{A}^T \mathbf{b} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} u_\tau^* \\ u_n^* \end{bmatrix}$$

Thus, explicitly for the velocity,

$$u_x^* = cu_\tau^* , \quad u_y^* = su_\tau^* ,$$

since $u_n^* = 0$ by the boundary condition $\hat{\mathbf{n}} \cdot \mathbf{u}|_{\Gamma_N} = 0$.

- Consider now **moving obstacles**. (Rotation, translation and deformation are allowed.) The local time derivative of the flow velocity field in arbitrary orthonormal (τ, n) coordinates, in the case where the coordinate system is allowed to rotate, $\theta = \theta(t)$, becomes

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} &= \frac{\partial}{\partial t} (u_\tau \hat{\tau} + u_n \hat{\mathbf{n}}) \\ &= \frac{\partial u_\tau}{\partial t} \hat{\tau} + u_\tau \frac{\partial \hat{\tau}}{\partial t} + \frac{\partial u_n}{\partial t} \hat{\mathbf{n}} + u_n \frac{\partial \hat{\mathbf{n}}}{\partial t} \\ &= \frac{\partial u_\tau}{\partial t} \hat{\tau} + u_\tau \frac{\partial}{\partial t} \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix} + \frac{\partial u_n}{\partial t} \hat{\mathbf{n}} + u_n \frac{\partial}{\partial t} \begin{bmatrix} -\sin \theta \\ \cos \theta \end{bmatrix} \\ &= \frac{\partial u_\tau}{\partial t} \hat{\tau} + u_\tau \begin{bmatrix} -\sin \theta \\ \cos \theta \end{bmatrix} \frac{\partial \theta}{\partial t} + \frac{\partial u_n}{\partial t} \hat{\mathbf{n}} + u_n \begin{bmatrix} -\cos \theta \\ -\sin \theta \end{bmatrix} \frac{\partial \theta}{\partial t} \\ &= \frac{\partial u_\tau}{\partial t} \hat{\tau} + u_\tau \hat{\mathbf{n}} \frac{\partial \theta}{\partial t} + \frac{\partial u_n}{\partial t} \hat{\mathbf{n}} + u_n (-\hat{\tau}) \frac{\partial \theta}{\partial t} \\ &= \left(\frac{\partial u_\tau}{\partial t} + u_n \frac{\partial \theta}{\partial t} \right) \hat{\tau} + \left(\frac{\partial u_n}{\partial t} - u_\tau \frac{\partial \theta}{\partial t} \right) \hat{\mathbf{n}} . \end{aligned}$$

The function $\partial \theta / \partial t$ is considered known.

- With $g \equiv 0$, we obtain the tangential and normal components of equation (29) as

$$\begin{aligned} \hat{\tau} \cdot \frac{\partial \mathbf{u}}{\partial t} &= \frac{\partial u_\tau}{\partial t} + u_n \frac{\partial \theta}{\partial t} = -\frac{\alpha}{\rho} \hat{\tau} \cdot \nabla \rho , \\ \hat{\mathbf{n}} \cdot \frac{\partial \mathbf{u}}{\partial t} &= \frac{\partial u_n}{\partial t} - u_\tau \frac{\partial \theta}{\partial t} = -\frac{\alpha}{\rho} \hat{\mathbf{n}} \cdot \nabla \rho . \end{aligned}$$

Accounting for the fact that at the boundary, $\hat{\mathbf{n}} \cdot (\mathbf{u} - \mathbf{u}_{\text{obst}}) \equiv 0$, i.e. $u_n = u_{n,\text{obst}}$ (known!), and rearranging terms, we have

$$\begin{aligned}\frac{\partial u_\tau}{\partial t} &= -u_{n,\text{obst}} \frac{\partial \theta}{\partial t} - \frac{\alpha}{\rho} \hat{\boldsymbol{\tau}} \cdot \nabla \rho, \\ \hat{\mathbf{n}} \cdot \nabla \rho &= \frac{\rho}{\alpha} \left(u_\tau \frac{\partial \theta}{\partial t} - \frac{\partial u_{n,\text{obst}}}{\partial t} \right).\end{aligned}$$

- * This accounts for both normal-direction acceleration and rotation of the obstacle.
 - This reduces correctly. If the obstacle does not move (θ constant in time (but still local to each point on the surface!)), $u_{n,\text{obst}} \equiv 0$, we again have $\hat{\mathbf{n}} \cdot \nabla \rho = 0$. Note also $u_n = u_{n,\text{obst}} = 0$. The first equation reduces to the previous PDE for the tangential velocity component u_τ .
- * The LHS will be obtained for the same point of time t from which data is fed to the RHS. If we use “old” data at $t = t_0$:
 - The vectors $\hat{\boldsymbol{\tau}}$ and $\hat{\mathbf{n}}$ refer to the “old” (τ, n) coordinate system at $t = t_0$.
 - The result is also expressed in the “old” (τ, n) system.
- * For FSI problems, $\partial \theta / \partial t$ and $u_{n,\text{obst}}$ may be challenging to obtain for $t > t_0$ before the timestep is complete (these are needed for RK2 integration of u_τ). Must model somehow.
 - Fixed point iteration possible (IG = use the values at $t = t_0$), but very slow, since it must iterate between fluid and structure solvers.
 - Trivial model (values from the structural model remain constant across the timestep) also possible. Fast, but not very accurate. Maybe good enough?
 - Structure solver must in any case provide not only $\mathbf{u}_{\text{obst}}$ (velocity, not displacement!) and θ , but also $\partial \theta / \partial t$ and $\partial \mathbf{u}_{\text{obst}} / \partial t$.
 - If the structural discretization uses a first-order form with displacement and velocity DOFs, θ and $\partial \theta / \partial t$ are easy to obtain. The acceleration can be computed for each structural DOF from the first-order ODE system (to get it as accurately as possible). The result can be converted to obstacle acceleration in a format that the fluid solver understands.
- * If we want to have $\hat{\mathbf{n}} \cdot \nabla \rho$ at the end of the timestep, we can (after applying the explicit time integration) just re-evaluate the RHS of the above expression, using the new $\hat{\mathbf{n}}|_{t=t_0+\Delta t}$ instead of the old $\hat{\mathbf{n}}|_{t=t_0}$.
 - Obviously, then $t = t_0 + \Delta t$ also in all terms on the RHS (use the updated solution fields!).
 - This is $\hat{\mathbf{n}} \cdot \nabla \rho$ at the beginning of the next timestep, so we need to compute it anyway.
- * For $\mathbf{u}$, provided that the rotation increment $\Delta \theta$ during the timestep is small, we can use the $u_n = 0$, $u_\tau = \text{sgn}(\hat{\boldsymbol{\tau}} \cdot \mathbf{u}) |\mathbf{u}|$ trick (i.e. just rotate the flow vector, keeping it tangential, now with the updated surface).
 - Doesn’t matter whether we use $\hat{\boldsymbol{\tau}}|_{t=t_0+\Delta t}$ or $\hat{\boldsymbol{\tau}}|_{t=t_0}$, since we only take $\text{sgn}(\dots)$, and the rotation is small.
 - In practice, we may performance-optimize further. If $\mathbf{u}$ at the boundary is stored in (τ, n) coordinates, this is equivalent to a do-nothing!
 - We simply re-interpret (u_τ, u_n) as if it was already expressed in the updated (τ, n) coordinates; this effectively rotates the vector into the new (τ, n) coordinates.
 - Only if the (x, y) representation of $\mathbf{u}$ is needed at the end of the timestep, $t = t_0 + \Delta t$, we must use $\hat{\boldsymbol{\tau}}|_{t=t_0+\Delta t}$ and $\hat{\mathbf{n}}|_{t=t_0+\Delta t}$ in the coordinate system conversion.
- * Some obstacles may remain stationary (no advection with flow) but undergo small vibrations (in response to the flow or otherwise) around a reference configuration.
 - Each surface segment oscillates around its local reference position and orientation. In this case we can probably get away with not actually updating the point clouds into the new (τ, n) system at each timestep, since the distances and directions are always close to those in the “local reference (τ, n) system” (computed for the reference configuration).
 - This gives major savings in computation time, because the boundary point clouds can be pre-processed just like for classical rigid stationary obstacles.
 - This makes the fluid solver just as fast as for the rigid obstacle case. Useful for realtime applications in FSI of structures undergoing small vibrations?

- In the fluid model, the obstacle then remains geometrically stationary, but its vibration is seen by the no-penetration boundary conditions, thereby affecting the flow. This is an adaptation (for Euler flows) of the approach that was used for potential flows in our earlier studies.
- This can be used in the papermaking application having small transverse deflections. The axial motion of the paper web is not seen by the Euler flow surrounding it.
- Even for a Navier–Stokes flow, in this application the structural geometry remains stationary in the Eulerian frame, and only the “zero relative velocity” BC is affected (since the obstacle surface moves axially).
- Could test this with a simpler problem, e.g. a kinematically oscillating rigid cylinder. This gives a one-way coupling, with a trivial structural model driving the fluid model. (With applications to 2D musical instruments and/or spatial audio processing? Record the numerical pressure response at some point(s) in the domain. Probably too expensive computationally, though; Helmholtz is much cheaper than an Euler flow.)
- * For compressible Navier–Stokes, doing this is even easier: then at the obstacle surface, $\mathbf{u} - \mathbf{u}_{\text{obst}} \equiv 0$; this fixes both components of $\mathbf{u}$. Add the $\nu \nabla \cdot \nabla \mathbf{u}$ diffusion term to the subproblem that was used to derive the boundary conditions. Obtain a condition for $\nabla \rho$. Something like:

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{\alpha}{\rho} \nabla \rho + \nu \nabla \cdot \nabla \mathbf{u} ,$$

whence

$$\frac{\partial \mathbf{u}_{\text{obst}}}{\partial t} = -\frac{\alpha}{\rho} \nabla \rho + \nu \nabla \cdot \nabla \mathbf{u}_{\text{obst}} ,$$

and by rearranging,

$$\nabla \rho = \frac{\rho}{\alpha} \left(\nu \nabla \cdot \nabla \mathbf{u}_{\text{obst}} - \frac{\partial \mathbf{u}_{\text{obst}}}{\partial t} \right) .$$

- Note that $\mathbf{u}_{\text{obst}}$ is not a rigid-body motion, since the obstacle will deform while vibrating.
 - To evaluate the laplacian, it may be convenient to use (x, y) coordinates, because then the axis vectors $\hat{\mathbf{e}}_x$ and $\hat{\mathbf{e}}_y$ are constant in space. The result can then be rotated into the local (τ, n) coordinates.
 - The $\partial \mathbf{u}_{\text{obst}} / \partial t$ term can be expressed in (τ, n) coordinates as before.
- Inflow boundaries. Here we have $\mathbf{u} = \mathbf{u}_{\infty}$, whence $\partial \mathbf{u} / \partial t = \partial \mathbf{u}_{\infty} / \partial t = 0$. Thus

$$0 = \frac{\partial \mathbf{u}}{\partial t} = \mathbf{g} - \frac{\alpha}{\rho} \nabla \rho ,$$

from which (if $\mathbf{g} \equiv 0$)

$$\nabla \rho = 0 .$$

- No-inflow free-stream boundaries (top, bottom, right): maybe we can just ignore these?
 - * Use the PDE to update the solution as if the node was inside the domain.
- Consider now the advection subproblems. For ρ , equation (28), we have

$$\frac{\partial \rho}{\partial t} = -\mathbf{u} \cdot \nabla \rho = -(u_{\tau} \hat{\boldsymbol{\tau}} + u_n \hat{\mathbf{n}}) \cdot \nabla \rho ,$$

but because $\hat{\mathbf{n}} \cdot \mathbf{u} = u_n \equiv 0$ at the obstacle boundary, the second term vanishes, leaving only

$$\begin{aligned} \frac{\partial \rho}{\partial t} |_{\Gamma_N} &= -u_{\tau} \hat{\boldsymbol{\tau}} \cdot \nabla \rho \\ &= -(cu_x + su_y) (c\hat{\mathbf{e}}_x + s\hat{\mathbf{e}}_y) \cdot \nabla \rho \\ &= - \left[(c^2 u_x + csu_y) \hat{\mathbf{e}}_x + (csu_x + s^2 u_y) \hat{\mathbf{e}}_y \right] \cdot \nabla \rho \\ &= - \left[(c^2 u_x + csu_y) \frac{\partial \rho}{\partial x} + (csu_x + s^2 u_y) \frac{\partial \rho}{\partial y} \right] . \end{aligned}$$

We have obtained a separate update formula for the boundary nodes.

* **But we don't need it — we already have the BC update above. (See algorithm below.)**

- On the other hand, the first line says this is 1D advection of the field ρ , along the local tangent direction of the boundary:

$$\frac{\partial \rho}{\partial t} + u_\tau \hat{\tau} \cdot \nabla \rho = 0 .$$

- Thus it is equivalent to treat it as such in the semi-Lagrangian advection procedure. Compute 2D advection inside the domain; compute 1D advection along the boundary at the obstacle boundaries.

* A small error will be introduced due to discretization, especially at curved boundaries.

* How to prevent the discretization error from corrupting u in the pressure subproblem?

- It may be enough that the pressure subproblem corrects for it by imposing $\hat{n} \cdot \nabla \rho = 0$?
- **We may simply impose that already here**, considering that the next update for the coupled subproblem (stiff subproblem) would do it anyway. If the result is always overwritten, it makes no sense to bother advecting ρ along the boundary.
- We can do the same for u . Enforce the boundary conditions already here; the advection along the boundary does not matter, because the next update of the coupled subproblem overwrites it.
- Thus the complete **solver algorithm** for one timestep becomes:
 - 1) *advect (for large timestep), ignore BCs except at inflow boundaries*
 - 2) *reset time counter (how much of the large timestep processed) for the stiff part*
 - 3) *update boundary nodes (using BCs from the coupled subproblem) with the currently latest available data*
 - 4) *if counter says the whole large timestep has been processed, DONE*
 - 5) *update the stiff part, inside-domain nodes*
 - 6) *jump to step 3*
- in practice, must account for symmetric Strang splitting (the above is just first-order Marchuk–Yanenko). Process the stiff part for $\Delta t/2$; then advect for Δt , then again process the stiff part for $\Delta t/2$. Snapshot the solution fields for output. Repeat.

- advection with MacCormack is $O((\Delta t)^2)$.

- stiff part could be integrated using RK2, also $O((\Delta t)^2)$.

- boundary nodes can easily get the old value, but how about the u_τ update that gets only $\partial u_\tau / \partial t$ and needs to integrate in time? It seems we must interleave the RK2 integration of the inside-domain part with the boundary update?

* Let's do this... RK2:

$$\begin{aligned} k_1 &= f(u_n, t_n) , \\ k_2 &= f(u_n + \beta \Delta t k_1, t_n + \beta \Delta t) , \\ u_{n+1} &= u_n + \Delta t \left[\left(1 - \frac{1}{2\beta}\right) k_1 + \frac{1}{2\beta} k_2 \right] , \end{aligned}$$

where $\beta \in (0, 1]$. Classical choices $\beta = 1/2$ (explicit midpoint method), $\beta = 2/3$ (Ralston's method), $\beta = 1$ (Heun's method / explicit trapezoid rule).

* Exiting advection step, entering stiff part update.

* At the beginning of a timestep, enforce boundary conditions for u_n and $\hat{n} \cdot \nabla \rho$ at $t = t_0$

* Compute k_1 simultaneously in the interior and on the boundaries

- This needs data only from $t = t_0$
- We have also u_τ at $t = t_0$
- k_1 contains $\partial u_\tau / \partial t$ at $t = t_0$

* Repeat for k_2

- As input, we have "candidate data" $u_n + \beta \Delta t k_1$ (with $\beta = 1$, this is the forward Euler prediction)
- Enforce boundary conditions for u_n and $\hat{n} \cdot \nabla \rho$ at $t = t_0 + \beta \Delta t$

- k_2 contains an approximation of $\partial u_\tau / \partial t$ at $t_0 + \beta \Delta t$
- How about FSI? Must model the behavior of the structure across the timestep before we actually have it... ($\mathbf{u}_{\text{bst}}$ and θ not yet available at $t_0 + \beta \Delta t$)
- * RK2 also outputs updated u_τ
- * u_τ behaves otherwise like an inside-domain DOF, but with a different update formula.
- * After computing the last $\mathbf{u}_{n+1}$ for this stiff part update, enforce boundary conditions for u_n and $\hat{\mathbf{n}} \cdot \nabla \rho$ at $t_0 + \Delta t$
 - May be important, because the advection step comes next
- **The stuff below here is not needed.**
- From the advection subproblem for $\mathbf{u}$, equation (30), we can write

$$\frac{\partial \mathbf{u}}{\partial t} = -\mathbf{u} \cdot \nabla \mathbf{u} = -(u_\tau \hat{\boldsymbol{\tau}} + u_n \hat{\mathbf{n}}) \cdot \nabla \mathbf{u} .$$

To be completely unambiguous, let us use index notation:

$$\frac{\partial u_i}{\partial t} = -u_j \partial_j u_i = -(u_\tau \partial_\tau + u_n \partial_n) u_i .$$

Due to BCs, $u_n \equiv 0$ at the boundary. For $i = \{\tau, n\}$, this gives two update formulas in (τ, n) coordinates:

$$\frac{\partial u_\tau}{\partial t} |_{\Gamma_N} = -u_\tau \partial_\tau u_\tau$$

(representing 1D advection of the tangential component along the boundary, as is physically expected) and

$$\frac{\partial u_n}{\partial t} |_{\Gamma_N} = -u_\tau \partial_\tau u_n .$$

But since $u_n \equiv 0$ along the whole boundary, also $\partial_\tau u_n \equiv 0$ and hence

$$\frac{\partial u_n}{\partial t} |_{\Gamma_N} = 0 ,$$

which just confirms that the normal velocity remains at zero.

- * For rotating obstacles, this becomes more complicated, because $\hat{\boldsymbol{\tau}}$ and $\hat{\mathbf{n}}$ no longer remain constant. The local (τ, n) coordinate system rotates with the obstacle, and it becomes important to distinguish whether we use the “old” (at t) or the “new” (at $t + \Delta t$) local coordinate system.
- Consider 1D advection along the boundary. Now we don’t have direct access to the tangential velocity field u_τ , but we can compute it as $u_\tau = cu_x + su_y$. Since the old solution fulfills the no-penetration BC, it will also hold that $-su_x + cu_y = u_n = 0$.
 - * Thus a way to correct for small error $u_n = O(\varepsilon)$ arising from discretization: take the magnitude $|\mathbf{u}|$ and use that as u_τ , correcting for the sign, $\text{sgn}(u_\tau) = \text{sgn}(\hat{\boldsymbol{\tau}} \cdot \mathbf{u})$. Do the same when converting back.
 - * Explicitly, take $u_\tau = \text{sgn}(\hat{\boldsymbol{\tau}} \cdot \mathbf{u}) |\mathbf{u}|$ and $u_n = 0$ on input (do not actually use the coordinate conversion formulas). On output, use the updated u_τ^* , and set $u_n^* = 0$, before applying conversion to (x, y) coordinates (now using the coordinate conversion formulas).
 - * Maybe compute u_τ as a field in a small neighborhood and use semi-Lagrangian advection along the boundary direction?
 - We already have fields for u_x and u_y that we can interpolate. Compute u_τ at the appropriate point using the interpolated u_x and u_y .
 - * Alternatively: converting the RHS partially to (x, y) coordinates, we have

$$\begin{aligned} \frac{\partial u_\tau}{\partial t} &= -u_\tau (c \frac{\partial}{\partial x} + s \frac{\partial}{\partial y}) (cu_x + su_y) \\ &= -u_\tau \left[c^2 \frac{\partial u_x}{\partial x} + cs \left(\frac{\partial u_y}{\partial x} + \frac{\partial u_x}{\partial y} \right) + s^2 \frac{\partial u_y}{\partial y} \right] . \end{aligned} \quad (31)$$

The parenthetical expression can be evaluated using WLSQM.

- But this “fights” the physical direction of information flow, because in WLSQM we use symmetric neighborhoods to generate the derivatives. Thus numerical stability can be bad.
- The rest is better to keep in (τ, n) coordinates. We have the numerical value of the old u_τ . We then compute the time derivative of this u_τ using the above equation, and perform the time integration to obtain the new value u_τ^* .
- For curved obstacles, we must update this slightly, since c and s are then no longer constant in space. But maybe this is ok for piecewise linear obstacles with the nodes placed at segment midpoints (or anywhere except at a corner)?

The advection problems can be handled by the semi-Lagrangian approach with MacCormack correction to obtain second-order accuracy. Global patched surrogate model interpolation gives access to the required interpolated values.

The clamping required for MacCormack can be achieved in a fast solver by using the data values from the (pre-computed) neighborhood of the node whose model the interpolator picked. (The interpolator must return this information!) This approach saves an (expensive) search for the k closest nodes from the point at which the interpolant was computed.

To match the accuracy, for the time integration, we choose (parametric) RK2. It has second-order accuracy, and it is computationally light, requiring only two stages.

- For the initial flowfield, we can achieve no-penetration by solving a potential flow problem. To do this using WLSQM, convert the Poisson problem (for the velocity potential) to the time-dependent diffusion problem (heat equation) by adding the time-dependent term, set zero Neumann BCs at the obstacles, and integrate until the steady state is approximately reached.
- Thus, assume that the old solution fulfills the no-penetration BC.
- Fit the model. This enables interpolation, needed for the semi-Lagrangian approach.
- The advection step — which strictly requires only inflow BCs — will introduce a small error (nonzero normal velocity component) especially near curved parts of the boundary (discretized into straight segments).
 - Can we simply re-align the velocity at the boundary nodes (turn the vector until it is aligned), and then re-fit only the affected neighborhoods (those having at least one obstacle node)?
 - * (Requires a per-problem-instance flag array in the solver, to switch off solving for instances that need no updating.)