EURO Greece

HPC Training Series

Setting up and running an optimization case in OpenFOAM.

Dr. Evangelos (Vaggelis) Papoutsis-Kiachagias
Senior Researcher, NTUA

School of Mechanical Engineering, NTUA,
Parallel CFD & Optimization Unit
email: vpapout@mail.ntua.gr

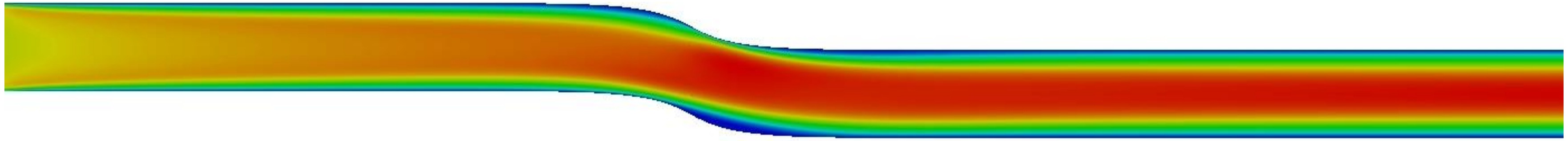
The tutorial case

- **Case is derived from**
  `$FOAM_TUTORIALS/incompressible/adjointOptimisationFoam/shapeOptimisation/sbend/laminar/opt/\`
  `unconstrained/BFGS/`
  but with a smaller mesh to get results faster

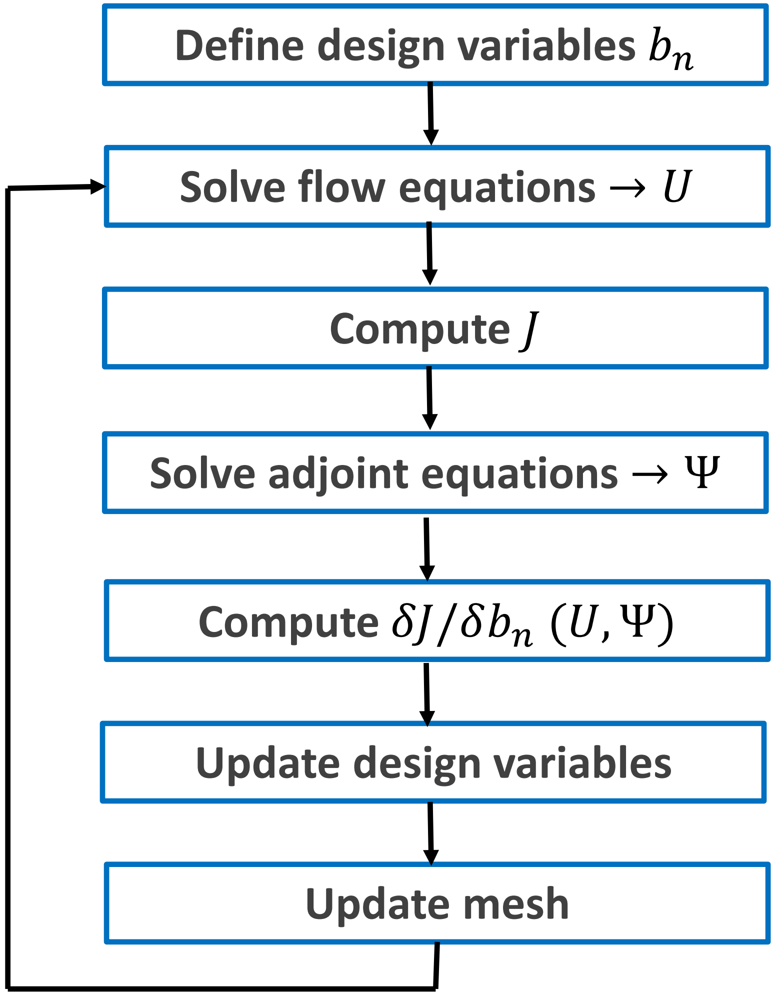
- **Laminar flow within an S-bend 2D duct, mesh is provided**

- $Re = 1000$

- **Objective: minimize volume-weighted total pressure losses** $J = -\int_{S_{I,O}} \left(p + \frac{1}{2} v_k^2\right) v_i n_i dS$



Gradient-based Shape Optimisation Loop



Define design variables b_n

Solve flow equations → U

Compute J

Solve adjoint equations → Ψ

Compute $\delta J/\delta b_n\ (U,\Psi)$

Update design variables

Update mesh

⇨ Need to define variables controlling the aerodynamic shape

⇨ Quantity to be optimized (forces, moments, losses, etc)

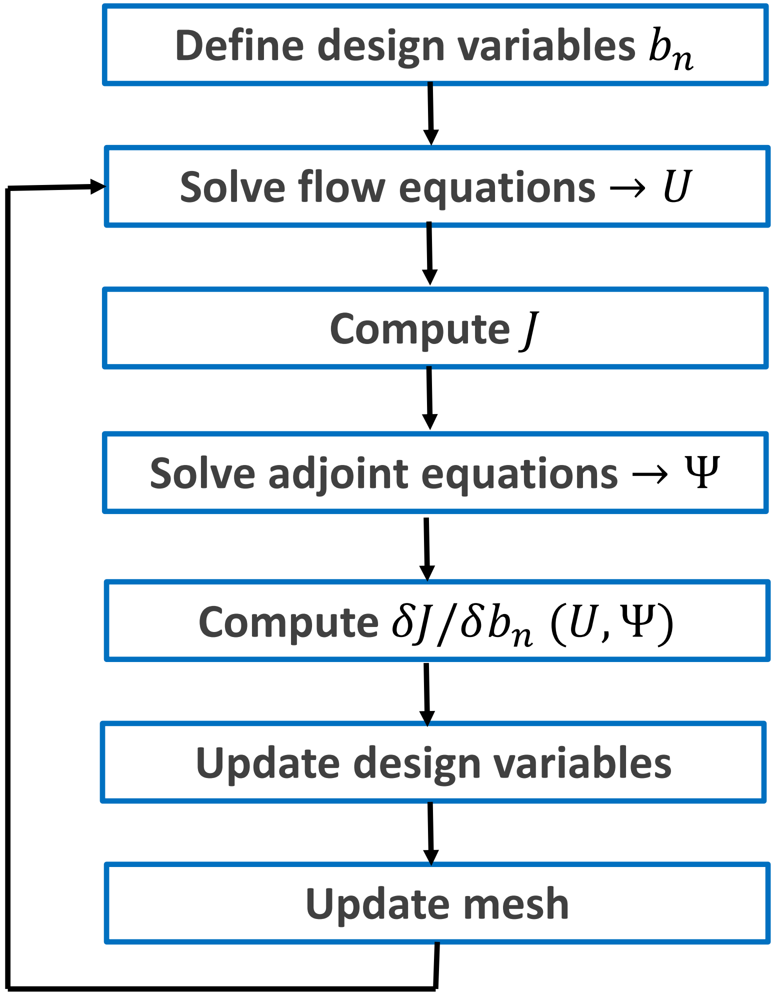
⇨ Drives the optimisation loop

⇨ Steepest descent, CG, BFGS …

⇨ PDE-based, Algebraic models …

Everything is implemented within OpenFOAM:
No need for third-party software, external scripts, etc

Gradient-based Shape Optimisation Loop



Define design variables b_n

Solve flow equations → U

Compute J

Solve adjoint equations → Ψ

Compute $\delta J/\delta b_n \ (U, \Psi)$

Update design variables

Update mesh

⇨ *constant/dynamicMeshDict*

⇨ *system/optimisationDict.primalSolvers*

⇨ *system/optimisationDict.adjointManagers,* **within each adjoint solver**

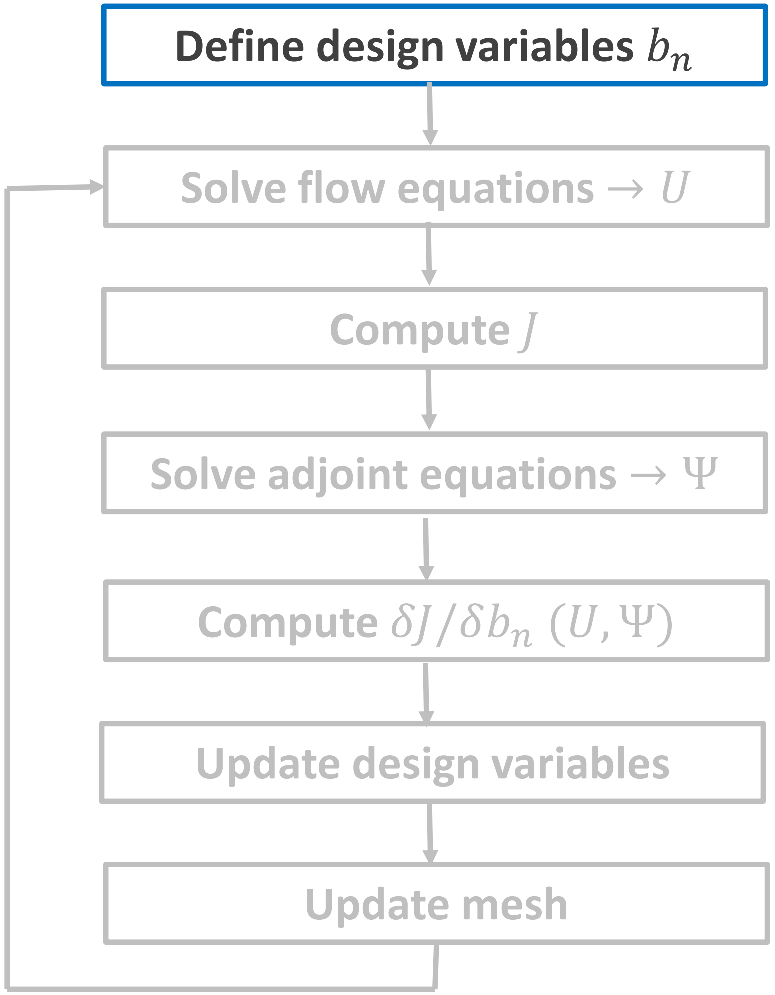
⇨ **Discretization in** *system/fvSchemes,* **Relaxation in** *system/fvSolution*

⇨ **Definition in:** *system/optimisationDict.optimisation.sensitivities*

⇨ **Definition in:** *system/optimisationDict.optimisation.updateMethod*

⇨ **Definition in:** *system/optimisationDict.optimisation.meshMovement*

Gradient-based Shape optimisation Loop



Define design variables b_n

Solve flow equations $\rightarrow U$

Compute J

Solve adjoint equations $\rightarrow \Psi$

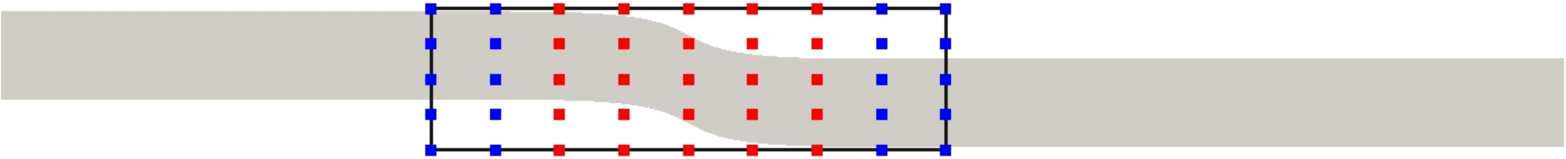
Compute $\delta J/\delta b_n \ (U,\Psi)$

Update design variables

Update mesh

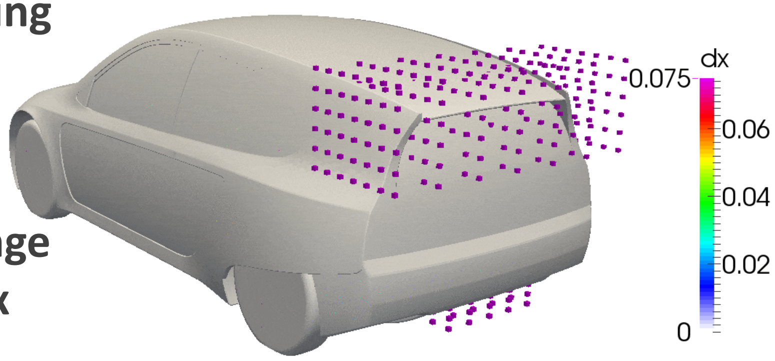
Parameterization for shape optimisation:

- **NURBS Curves (2D) and Surfaces (3D)**
- **All of the wall nodes**
- **Volumetric B-Splines (Free Form Deformation, FFD)**



Volumetric B-Splines:

- **Maps all CFD grid points within the morphing boxes from the Cartesian to a parametric space $(x,y,z) \rightarrow (u,v,w)$**
- **Mapping has to be done only once**
- **Then, changing the control points will change all CFD grid nodes within the morphing box (boundary and internal)**
- **Update is done through an algebraic relation: very fast!**



Gradient-based Shape optimisation Loop

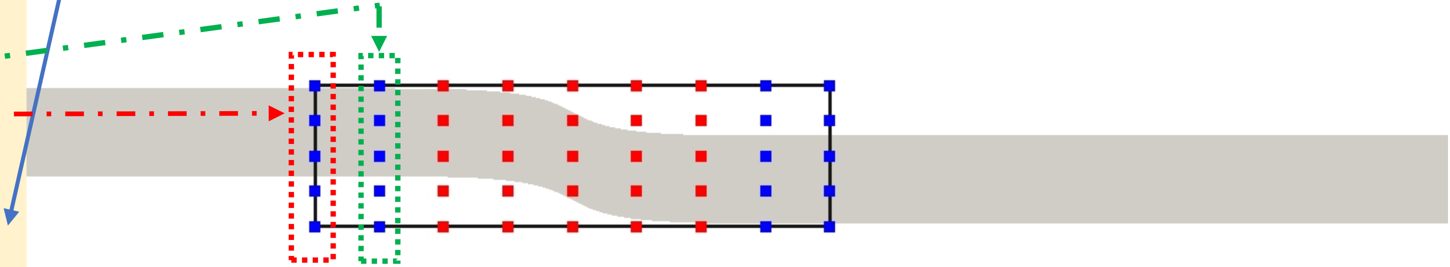
Define design variables b_n

Defined in: ***constant/dynamicMeshDict***

```
solver volumetricBSplinesMotionSolver;

volumetricBSplinesMotionSolverCoeffs
{
 duct
 {
   type    cartesian;
   nCPsU   9;
   nCPsV   5;
   nCPsW   3;
   degreeU 3; // max: nCPsU - 1
   degreeV 3; // max: nCPsV - 1
   degreeW 2; // max: nCPsW - 1

   controlPointsDefinition axisAligned;
   lowerCpBounds (-1.1 -0.21 -0.05);
   upperCpBounds ( 1.1  0.39  0.15);

   confineUMovement false;
   confineVMovement false;
   confineWMovement true;
   confineBoundaryControlPoints false;


   confineUMinCPs ( (true true true) (true true true) );
   confineUMaxCPs ( (true true true) (true true true) );
   confineWMinCPs ( (true true true) );
   confineWMaxCPs ( (true true true) );
 }
}
```

Basic entries:

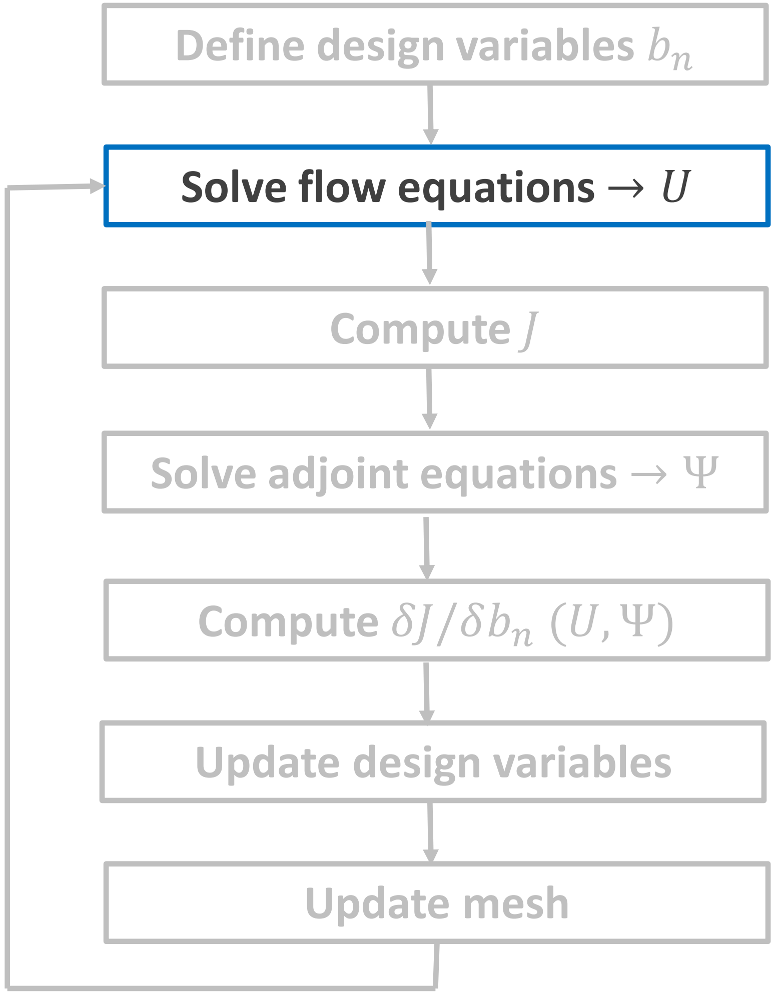
- **Number of control points (CPs) per box direction**
- **Degree per direction (smaller degree → more local support)**
- **CPs defined either aligned with a coordinate system (Cartesian, cylindrical) or given manually through a dictionary**
- **Possible to confine the movement of (some of) the CPs in certain directions**
- **Continuity with the stationary part of the mesh must be preserved! Keeping the boundary CPs constant**



>> writeMorpherCps

Writes the control points in a Paraview-readable format

Gradient-based Shape optimisation Loop



Define design variables b_n

Solve flow equations $\rightarrow U$

Compute J

Solve adjoint equations $\rightarrow \Psi$

Compute $\delta J/\delta b_n \ (U, \Psi)$

Update design variables

Update mesh

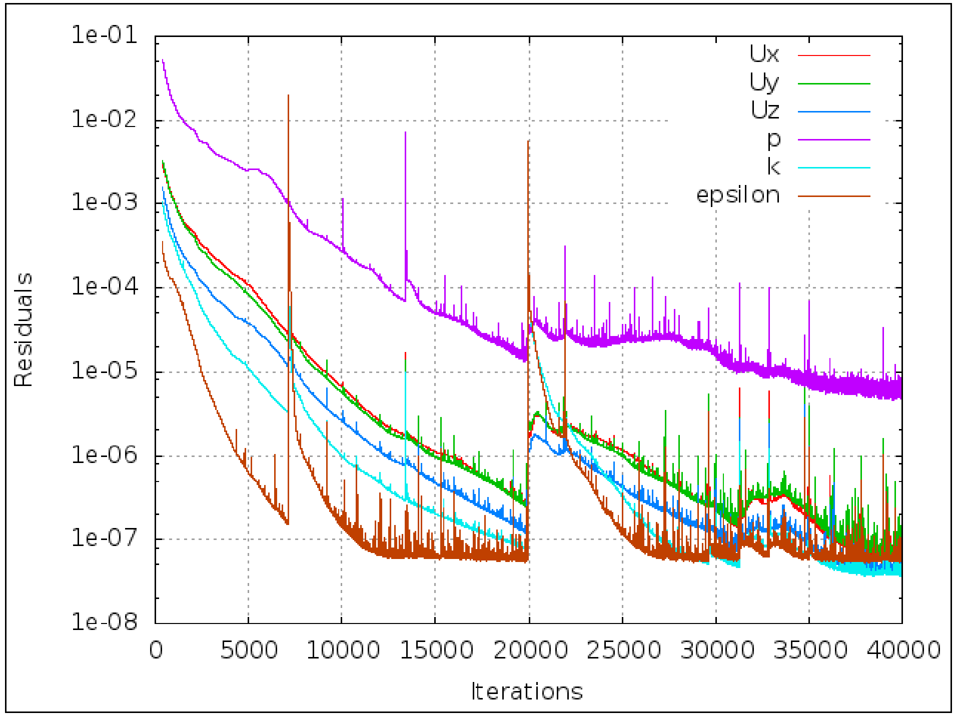
Defined in *system/optimisationDict.primalSolvers*

Incompressible, steady-state flows

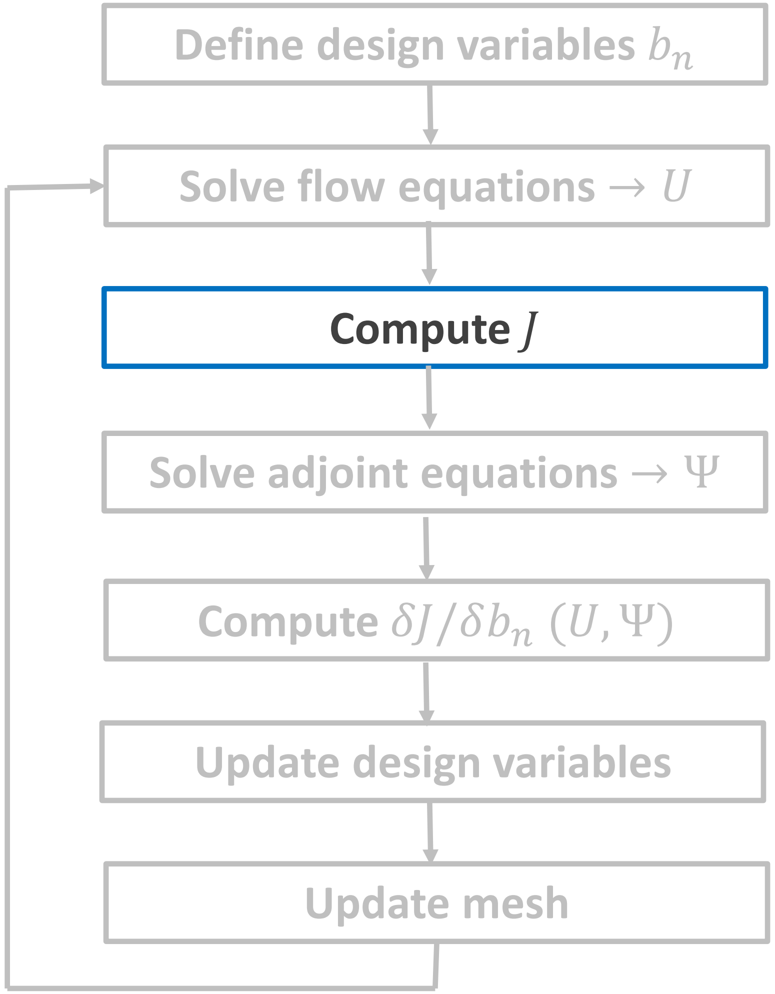
- **SIMPLE is incorporated into *adjointOptimisationFoam***
- **Multi-point optimisation supported; can define more than one primal solvers**

Desired for optimisation, if possible

- **Well converged solution (e.g. residuals of ~1.e−05, 1.e−06)**
- **Non-oscillating residuals**



Gradient-based Shape optimisation Loop



Define design variables b_n

Solve flow equations → U

Compute J

Solve adjoint equations → Ψ

Compute $\delta J/\delta b_n\ (U,\Psi)$

Update design variables

Update mesh

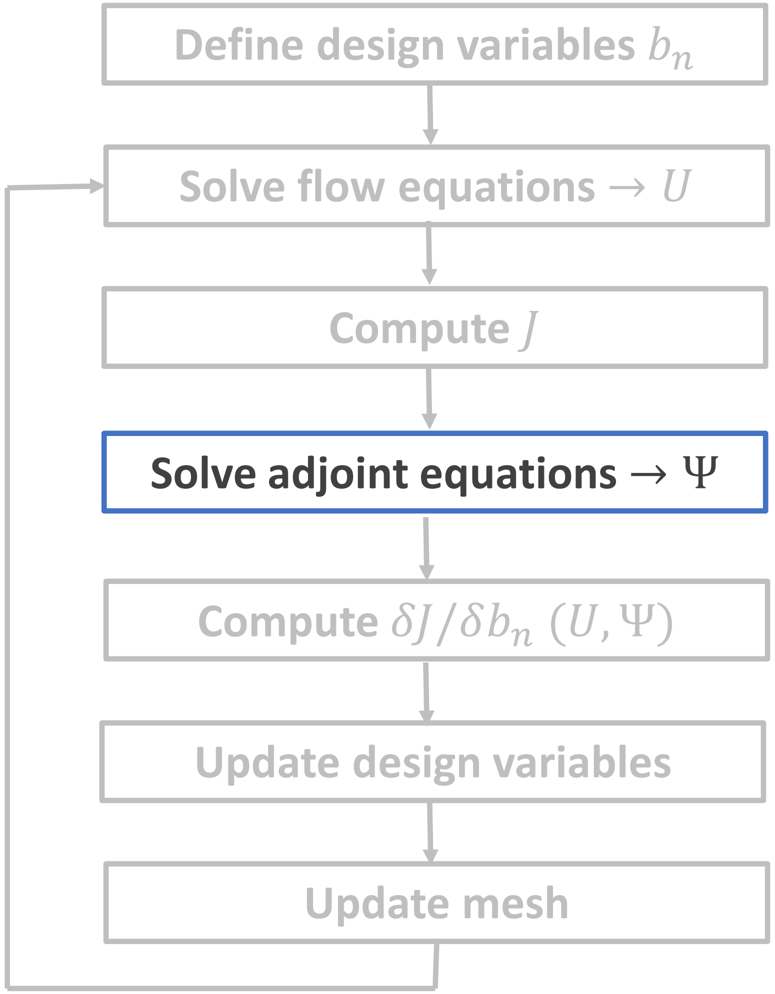
Defined in system/optimisationDict.adjointManagers, within each adjoint solver

Quantity to be optimised

- ***adjointOptimisationFoam*** **always assumes minimization**
- **Objectives can be defined as (surface or volume) integral quantities**
- **A number of objective functions are available:**
  Forces, moments, total pressure losses etc …
- **Multiple objective functions can be tackled by concatenating them into a single one using appropriate weights**

$$J = w_1 J_1 + w_2 J_2$$

Gradient-based Shape optimisation Loop


Define design variables b_n
Solve flow equations → U
Compute J
Solve adjoint equations → Ψ
Compute $\delta J/\delta b_n\ (U,\Psi)$
Update design variables
Update mesh

Discretization in system/fvSchemes, Relaxation in system/fvSolution

$$R^q = -\frac{\partial u_j}{\partial x_j} + \boxed{\frac{\partial J_{\Omega'}}{\partial p}} = 0$$

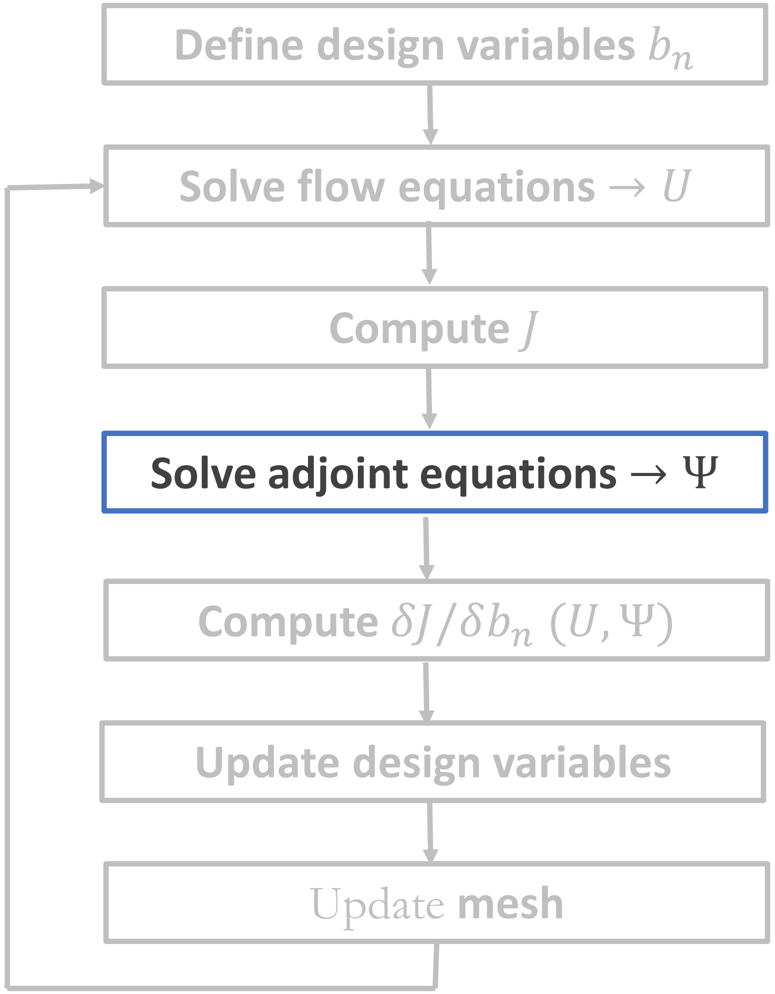
$$R_i^u = \underbrace{\boxed{u_j \frac{\partial v_j}{\partial x_i}}}_{\text{ATC}} \underbrace{\boxed{- \frac{\partial \left(u_i v_j\right)}{\partial x_j}}}_{\text{AC}} - \frac{\partial \tau^a_{ij}}{\partial x_j} + \frac{\partial q}{\partial x_i} + \boxed{\frac{\partial J_{\Omega'}}{\partial v_i}} = 0\ ,\ \ i = 1, 2(,3)$$

Adjoint PDEs (laminar flows):

- **Similar form with the Navier-Stokes equations. A few noticeable differences**
- **Adjoint convection (AC): adjoint velocity is convected by the (minus) primal velocity. Linear equations!**
- **Adjoint Transpose Convection (ATC): Non-conservative term. Numerically tricky in real-word applications.**
- **Source terms if the objective function includes volume integrals containing p or v_i**

Additional terms and equations when dealing with turbulent flows

Gradient-based Shape optimisation Loop


Define design variables b_n
Solve flow equations → U
Compute J
Solve adjoint equations → Ψ
Compute $\delta J/\delta b_n\ (U,\Psi)$
Update design variables
Update mesh

Defined in 0/pa and 0/Ua

$$u_{\langle n\rangle} = u_j n_j = -\boxed{\frac{\partial J_{S_{I-w},i}}{\partial p}}\, n_i$$

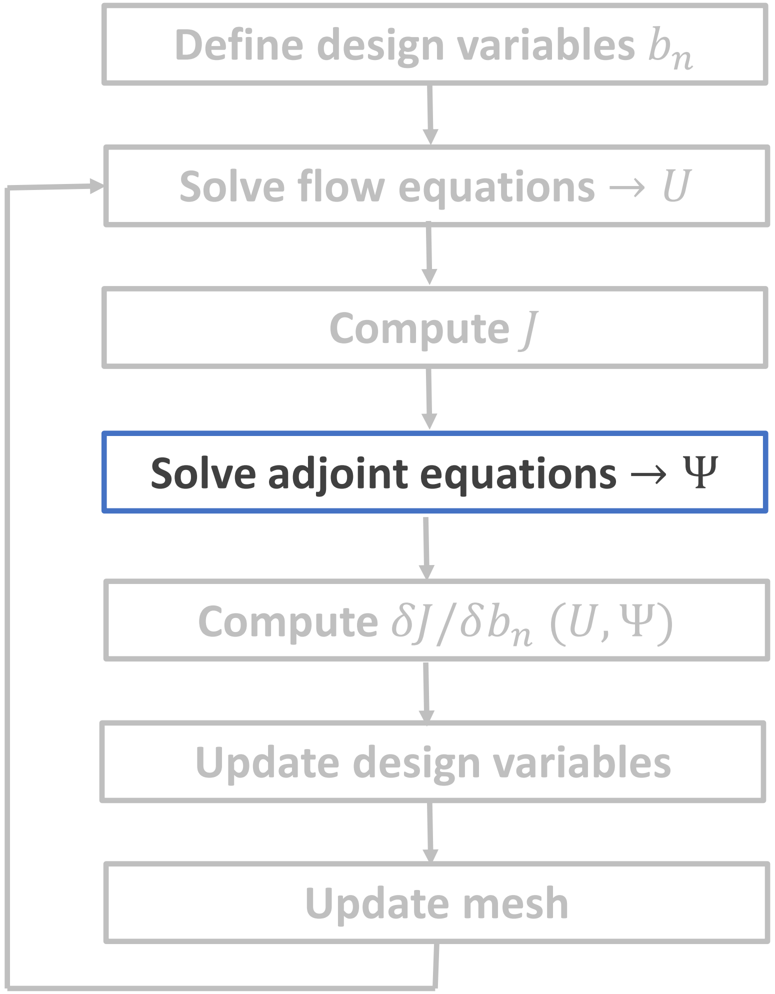
$$u^{I}_{\langle t\rangle} = u_i t^{I}_i = \boxed{\frac{\partial J_{S_{I-w},k}}{\partial \tau_{ij}}}\, n_k t^{I}_i n_j + \boxed{\frac{\partial J_{S_{I-w},k}}{\partial \tau_{ij}}}\, n_k t^{I}_j n_i$$

$$u^{II}_{\langle t\rangle} = u_i t^{II}_i = \boxed{\frac{\partial J_{S_{I-w},k}}{\partial \tau_{ij}}}\, n_k t^{II}_i n_j + \boxed{\frac{\partial J_{S_{I-w},k}}{\partial \tau_{ij}}}\, n_k t^{II}_j n_i$$

Adjoint Boundary conditions:

- **Depend on the type (not value!) of primal boundary conditions!**
- **Most common for incompressible flows: Dirichlet Inlet $\vec{v}$, Dirichlet Outlet p**
- **Depend on the derivatives of J w.r.t. the pressure, velocity and stress tensor**

Gradient-based Shape optimisation Loop



Define design variables b_n
Solve flow equations → U
Compute J
Solve adjoint equations → Ψ
Compute $\delta J/\delta b_n \ (U,\Psi)$
Update design variables
Update mesh

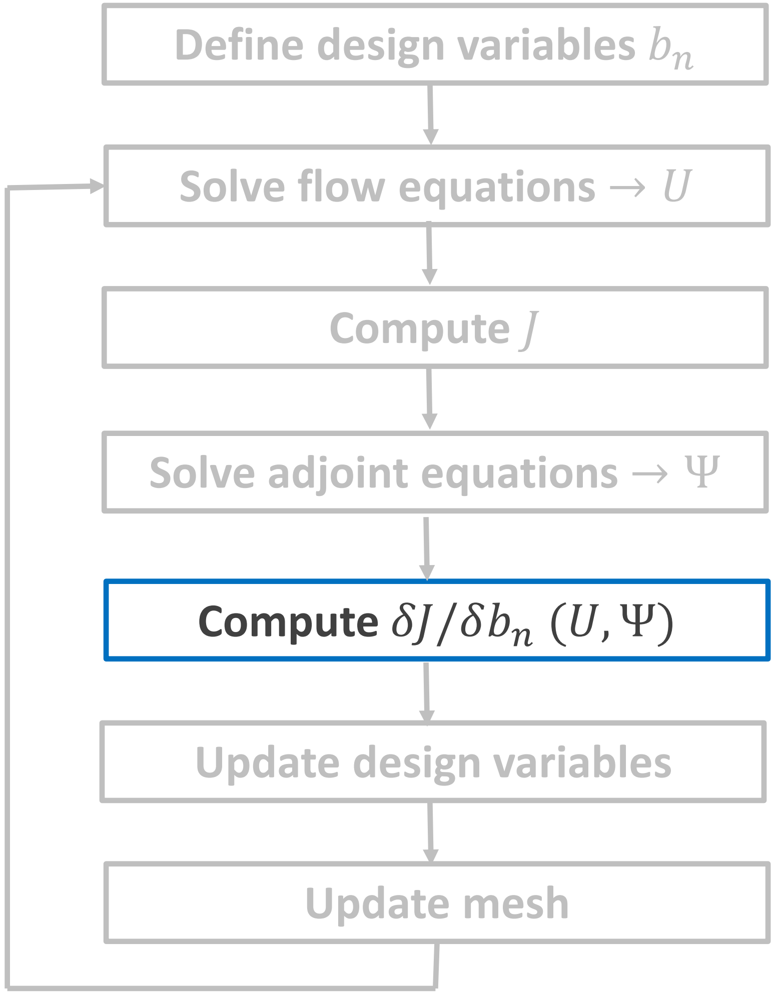
Defined in ***system/optimisationDict.adjointManagers***

How many adjoint equations do we have to solve?

- **One for each objective for which we need the gradient**
  - **Gradients of linear combinations of functions defined at a single operating point can be computed with one adjoint solution!**
  - **Advanced methods dealing with constraints (e.g. SQP, constraint projection) need the gradient of the constraint function separately**

- **(At least) One for each operating point solved**

Gradient-based Shape optimisation Loop



Define design variables b_n

Solve flow equations → U

Compute J

Solve adjoint equations → Ψ

Compute $\delta J/\delta b_n\ (U,\Psi)$

Update design variables

Update mesh

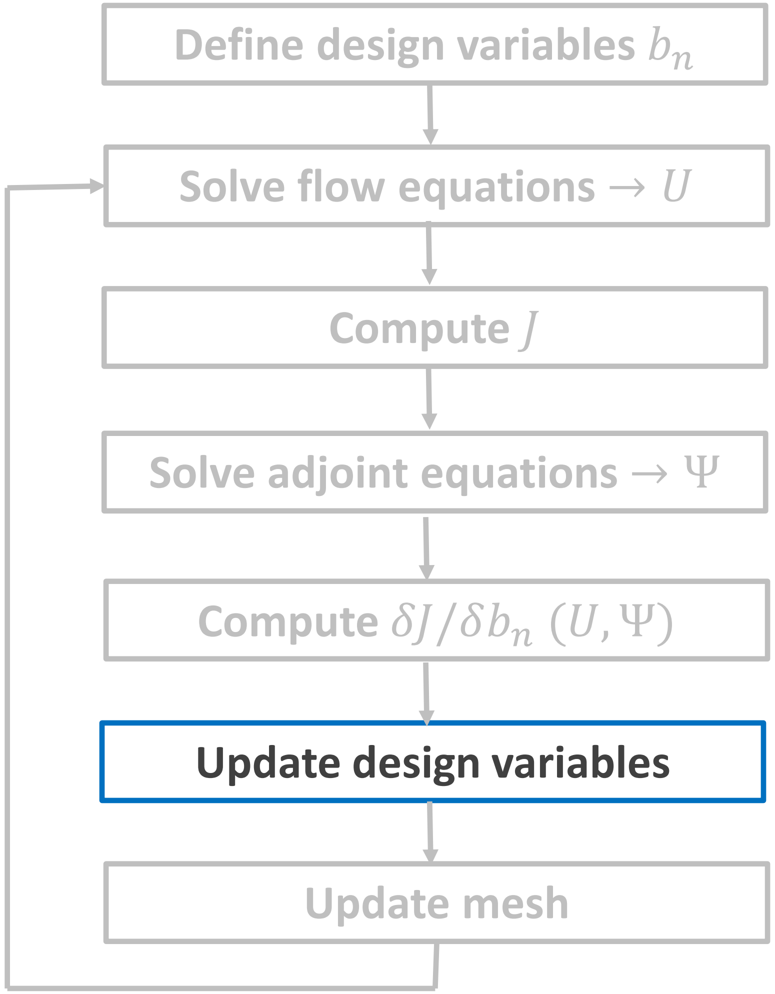
Defined in
system/optimisationDict.optimisation.sensitivities

Two mathematical formulations for shape optimisation

- **Based on Surface Integrals, (E)-SI**
  - **Need to solve an additional adjoint grid displacement PDE for m_i^a**
    - **Boundary conditions are created automatically**
    - **Need to define a linear solver in *fvSolution***
    - **No relaxation is required**
    - **Solved at a post-processing level, i.e. after the solution of the adjoint mean flow equations**

- **Based on Field Integrals, FI**
  - **Need to compute the grid sensitivities fields, i.e. $\frac{\delta x_k}{\delta b_n}$**
  - **Depending on the grid displacement model this might be computed by**
    - **solving additional PDEs (e.g. PDE-based grid displacement)**
    - **Analytically (e.g. Volumetric B-Splines)**

Gradient-based Shape optimisation Loop



Define design variables b_n

Solve flow equations $\rightarrow U$

Compute J

Solve adjoint equations $\rightarrow \Psi$

Compute $\delta J/\delta b_n\ (U,\Psi)$

Update design variables

Update mesh

Defined in
system/optimisationDict.optimisation.updateMethod

Compute the update of the design variables based on $\frac{\delta J}{\delta b_n}$ through

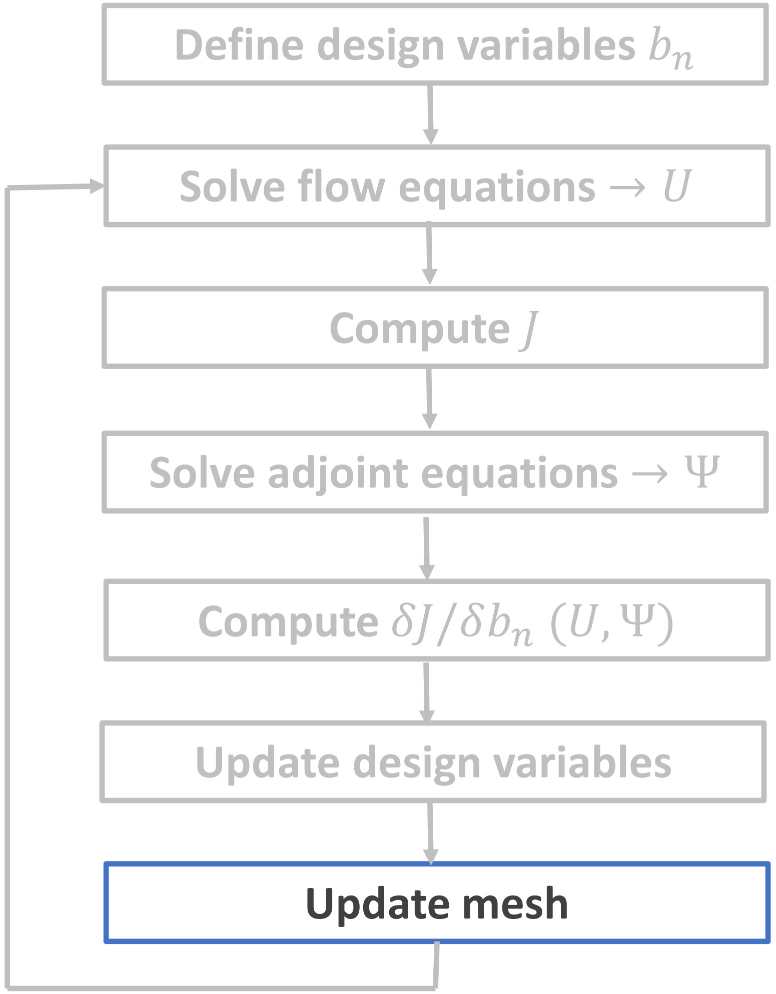
$$b_n^{new} = b_n^{old} + \eta s_n$$

- Unconstrained optimisation
  - Steepest descent
  - Conjugate Gradient
  - Quasi-Newton methods: BFGS, SR1

- Constrained optimisation
  - Constraint projection (exceptional for linear constraints)
  - SQP

- Step (η) definition
  - Direct (usually not practical)
  - Through a max. desired deformation in the initial opt. cycle

Gradient-based Shape optimisation Loop


Define design variables b_n
Solve flow equations → U
Compute J
Solve adjoint equations → Ψ
Compute $\delta J/\delta b_n\ (U, \Psi)$
Update design variables
Update mesh

Defined in
constant/dynamicMeshDict

- Need to translate $\Delta \mathrm{b}_{\mathrm{n}}$ into a new geometry and computational mesh
- Remeshing can be costly and possibly result to inconsistent sensitivity derivatives. Grid displacement is preferable
- Depends on the parameterization and chosen grid displacement method
  - Usually, one tool for parameterization (e.g. NURBS), a different one for grid displacement (e.g. Laplace PDEs)
  - Volumetric B-Splines handles both simultaneously
- checkMesh ran after each update to check mesh quality

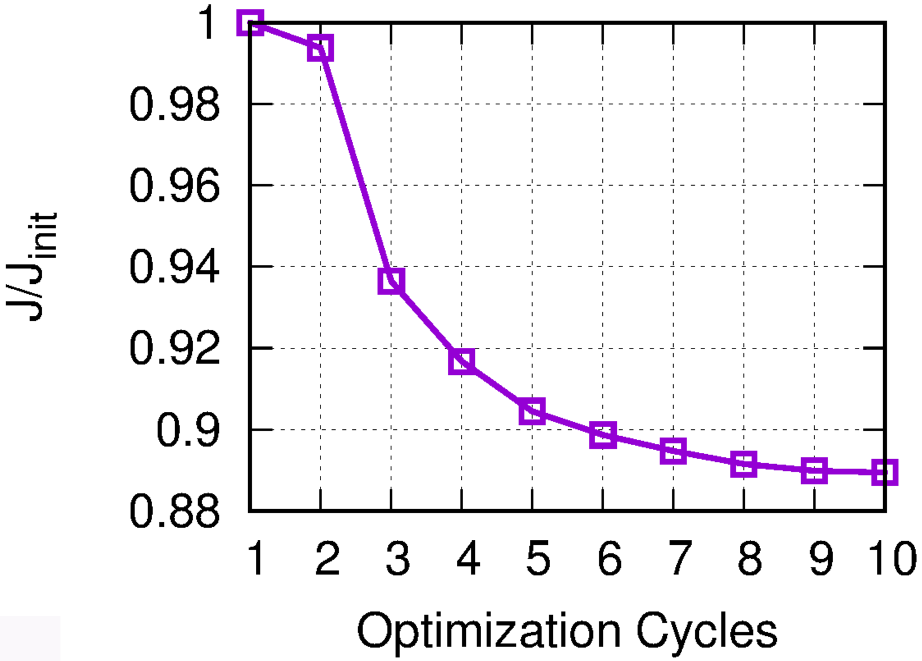
S-bend: optimisation results

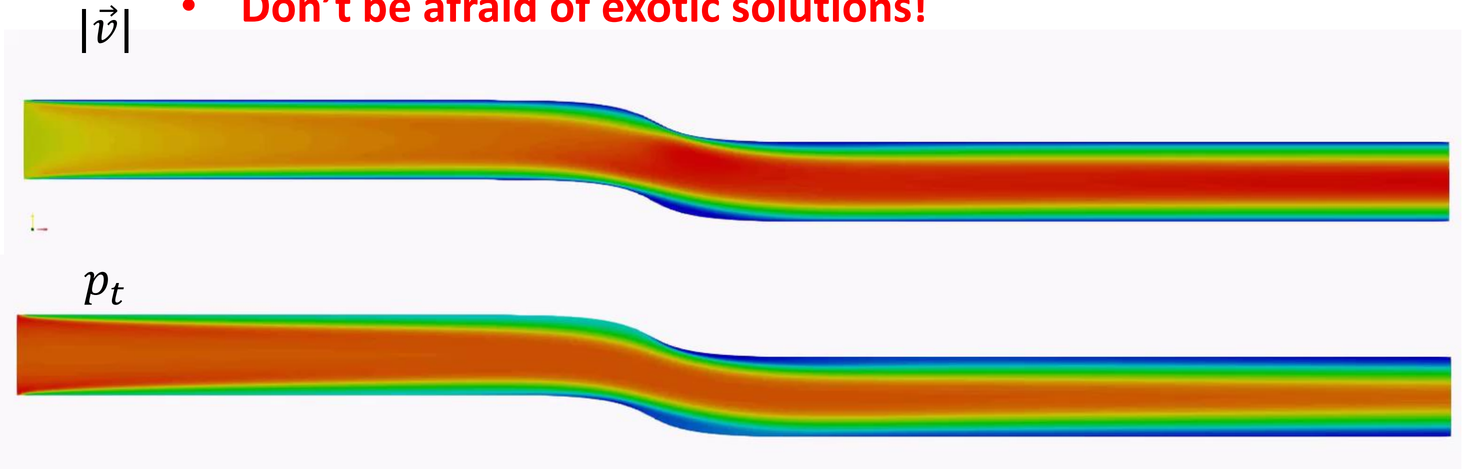
Run the optimization loop

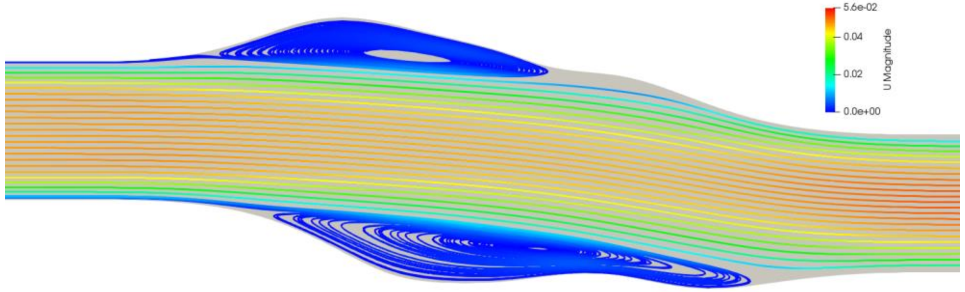
```
>> ./Allrun.parallel > log 2> err & (~2.5
min/4 procs)
```

What to examine:

- **Is J reduced?**
- **Is J converged? (history in *optimisation/objective* folder)**
- **Have the flow equations converged? (check log file)**
- **Is the mesh valid at the optimised solutions? (check log file or checkMesh)**
- **What is the mechanism behind the reduction in J?**
- **Don’t be afraid of exotic solutions!**

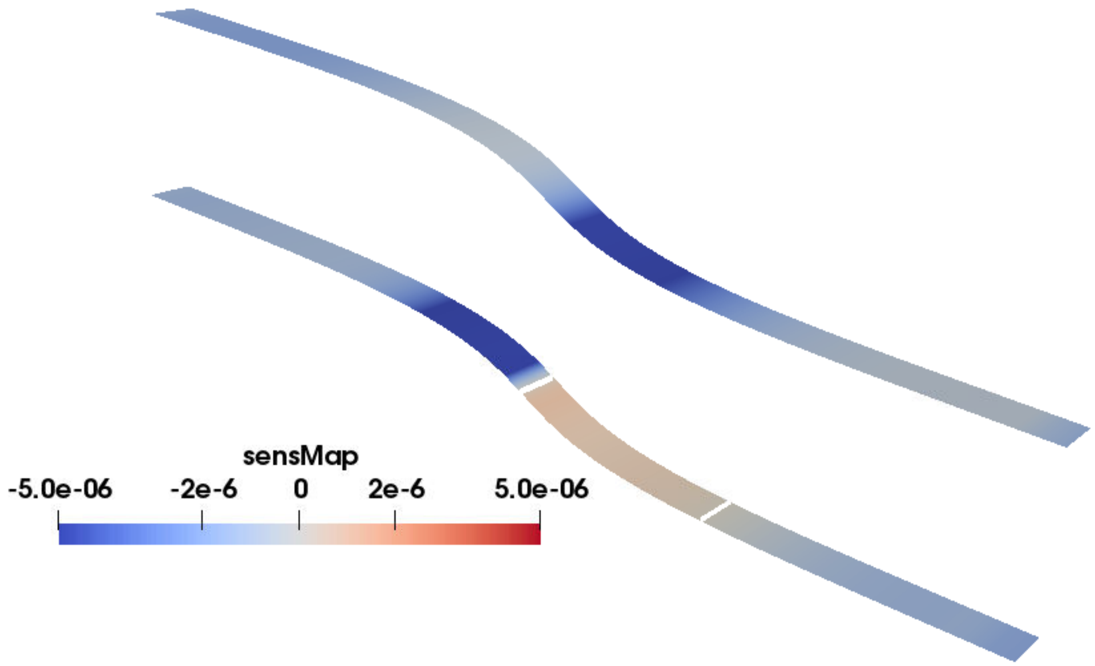






S-bend: Computing sensitivity maps

- Compute $\frac{\delta J}{\delta x_i} n_i$
- A few changes in *optimisationDict and controlDict*
- Tells us how each boundary node has to move to reduce J
  - Red: move against the surface normal (inwards)
  - Blue: move towards the surface normal (outwards)
  - White-ish: insignificant
- Computed on the initial geometry: does not mean that the optimised geometry will follow this ! …
- Good feedback towards the designer
- Useful in placing morhing boxes

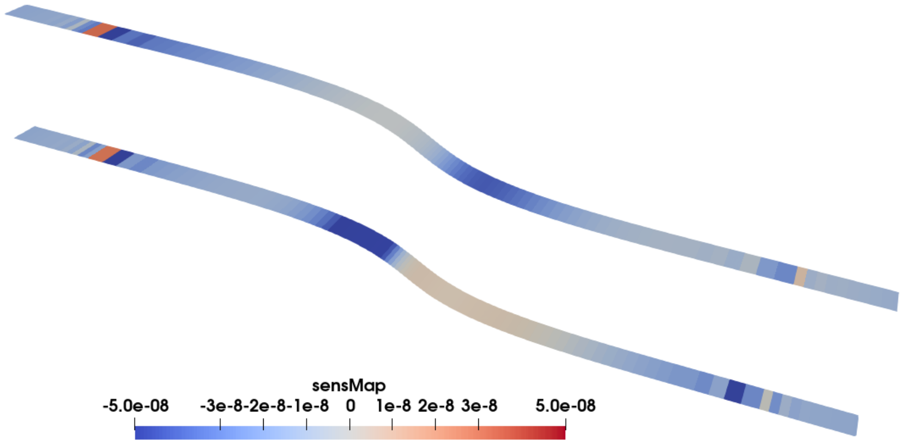


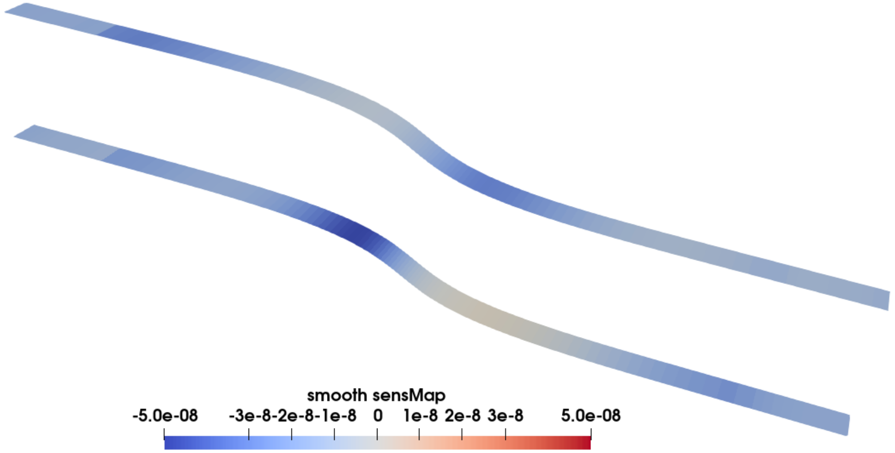
S-bend: Smoothing the sensitivity map

- **In more complex/industrial cases, checkerboards occur in the computed sensitivity maps.**
- **This problem becomes pronounced in meshes built with snappyHexMesh!**
- **The direction of favorable surface displacement becomes ambiguous...**
- **Smooth the sensitivity-map, G, by solving**

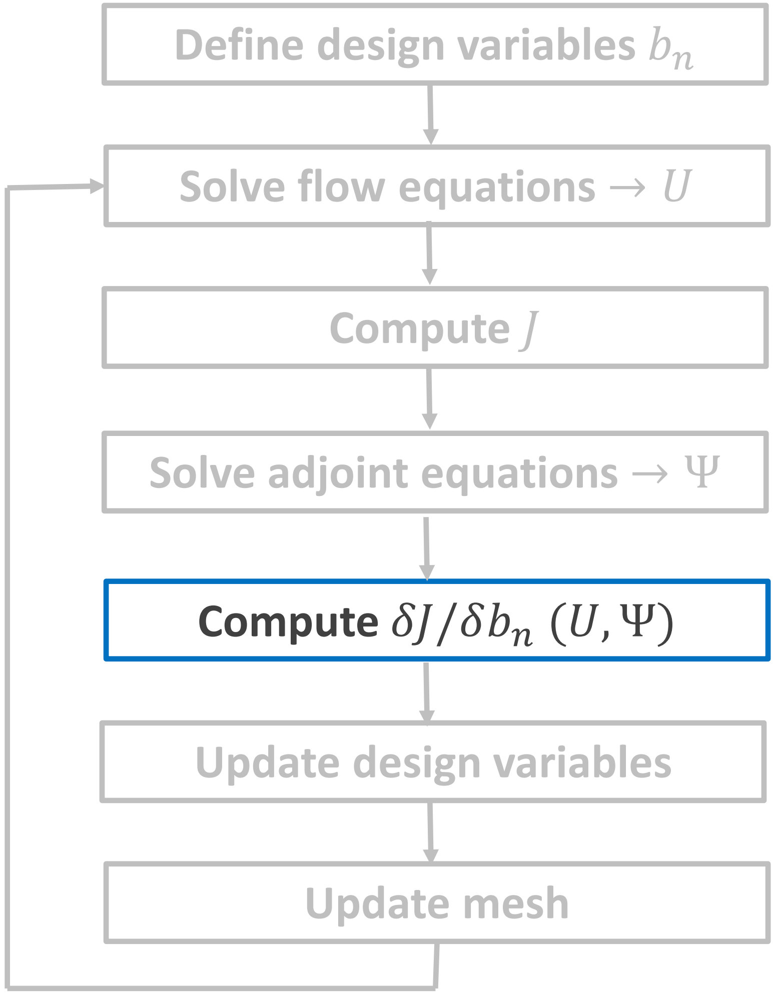
$$-R^2\frac{\partial^2 \hat{G}}{\partial x_j^2} + \hat{G} = G$$

on a *finiteArea* mesh.





S-bend: Smoothing the sensitivity map – Additional Entries



Define design variables b_n
Solve flow equations → U
Compute J
Solve adjoint equations → Ψ
Compute $\delta J/\delta b_n\ (U,\Psi)$
Update design variables
Update mesh

- **Additional entries in *system/optimisationDict.optimisation.sensitivities* related to the Laplace-Beltrami equation**

$$-R^2\frac{\partial^2\hat{G}}{\partial x_j^2}+\hat{G}=G$$

  - **The smoothing radius is either specified explicitly, or computed as a multiple of the average surface edges' length.**
  - **Boundary conditions for the smooth sensitivity field are created automatically.**

- **For the creation of the *faMesh*, an *faMeshDefinition* dictionary can be optionally provided in the *system* folder.**

- ***faSchemes* & *faSolution* should be present in the *system* directory.**

Takeaway messages:

- Adjoint supports optimisation loops at a small CPU cost ($\sim$20 cycles → ~ 40 flow solutions)
- Ideal for both early-stage development and refinement
- More optimisation types available
  - Active flow control (jet-based optimisation)
  - A Posteriori Error Analysis (optimally refine your mesh to compute an accurate objective)
  - Design under uncertainties
- Optimisation (like CFD) is not magic. Take care when defining your problem
- Before accepting (or discarding) an optimised geometry
  - Check the convergence of the flow equations
  - Check the mesh quality
- Try to understand the mechanisms behind the objective reduction
  - Often leads to better designs and/or better-defined optimisation problems!

Additional topics covered through the tutorials under `$FOAM_TUTORIALS/incompressible/adjointOptimisationFoam`

- **Effect of the update method**
  `shapeOptimisation/sbend/laminar/opt/unconstrained`

- **Constrained optimisation**
  `shapeOptimisation/naca0012/lift/opt/constraintProjection`

- **3D, industrial-like cases**
  `shapeOptimisation/motorbike`

When in doubt about the case settings, you can consult the manual