NATIONAL TECHNICAL UNIVERSITY OF ATHENS (NTUA)
SCHOOL OF MECHANICAL ENGINEERING
LAB. OF THERMAL TURBOMACHINES
PARALLEL CFD & Optimization UNIT (PCOpt/NTUA)

The Continuous Adjoint Method in Aero/Hydrodynamic Optimisation
Part II: Adjoint Optimisation in OpenFOAM. Hands-on Training

Dr. Kyriakos C. Giannakoglou, Professor NTUA
Dr. Evangelos (Vaggelis) M. Papoutsis-Kiachagias

November 9, 2022

Before we begin

- **Connect to the training box using the instructions and credentials provided to you**
- **Copy cases for the training to your home directory**

```
>> cp -r /opt/OpenFOAM/extra/optimization/2022_11_adjointTraining ~/.
```

- **All cases to be covered are variants of the tutorials for *adjointOptimisationFoam,* found under**

```
$FOAM_TUTORIALS/incompressible/adjointOptimisationFoam
```

Many variants of these cases exist there, showcasing different code features. Make sure you explore them!

Throughout the training, you will be asked to run a number of tutorial cases. Depending on the number of the participants, these might be run in serial or in parallel. Use the
`Allrun` **scripts to run in serial or**
`Allrun.parallel` **scripts to run in parallel,**
according to your instructor's guidelines

What we will discuss: adjointOptimisationFoam

- **An all-in-one OpenFOAM executable implementing an integrated, gradient-based optimisation workflow**
- **Product of a 12 years of development at PCOpt/NTUA and FOSS**
- **Integrated into the official OpenFOAM version in collaboration with OpenCFD**
- **Focus on shape optimisation through some simple examples**
- **Adjoint code corresponds to the one in v2206**
- **User manual:**
  https://openfoam.com/documentation/files/adjointOptimisationFoamManual_v2006.pdf
  Covers all functionality up until v2106

Acknowledgments:

Themis Skamagkis
Andreas Margetis
Nikolaos Galanos

Dr. Konstantinos Gkaragkounis
Dr. Ioannis Kavvadias
Dr. Alexandros Zymaris
James Koch
Dr. Andrew Heather

Current and future status of adjointOptimisationFoam

OpenFOAM v1906

- **Adjoint to incompressible, steady-state flows**
- **Differentiation of the Spalart-Allmaras turbulence model**
- **Computation of sensitivity maps with the E-SI approach (see first part of the lecture)**

OpenFOAM v1912

- **Surface and volume parameterization using volumetric B-Splines**
- **Automated shape optimisation loops**
- **Computation of sensitivity derivatives using the FI approach (see first part of the lecture)**

OpenFOAM v2006

- **New objective function related to the qualitative evaluation and minimization of noise**
- **Sensitivity contributions from rotating boundaries**

OpenFOAM v2112

- **Smoothing of sensitivity maps**

OpenFOAM v2206

- **Adjoint to the k-ω SST turbulence model**

Beyond

- **Plenty of more capabilities available in-house (topology optimisation, unsteady adjoints, CHT, etc)**

The tutorial case

- **Change directory to the laminar variant of the sbend case**

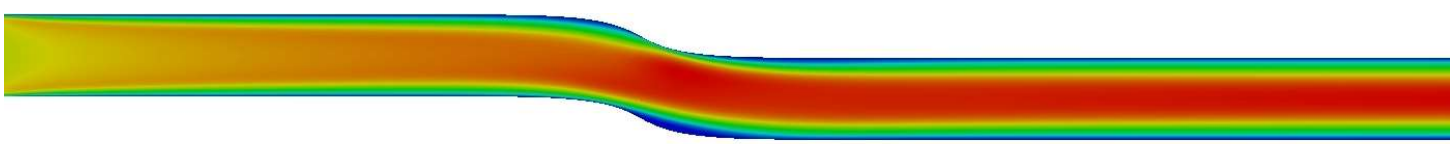
```
>> cd ~/2022_11_adjointTraining/sbend/laminar/optimisation
```

- **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** ⇨ **Need to define variables controlling the aerodynamic shape**
- **Solve flow equations $\rightarrow U$**
- **Compute J** ⇨ **Quantity to be optimized (forces, moments, losses, etc)**
- **Solve adjoint equations $\rightarrow \Psi$**
- **Compute $\delta J/\delta b_n \ (U,\Psi)$** ⇨ **Drives the optimisation loop**
- **Update design variables** ⇨ **Steepest descent, CG, BFGS …**
- **Update mesh** ⇨ **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** ⇨ *constant/dynamicMeshDict*
- **Solve flow equations → U** ⇨ *system/optimisationDict.primalSolvers*
- **Compute J** ⇨ *system/optimisationDict.adjointManagers,* **within each adjoint solver**
- **Solve adjoint equations → Ψ** ⇨ **Discretization in** *system/fvSchemes,* **Relaxation in** *system/fvSolution*
- **Compute $\delta J/\delta b_n\ (U,\Psi)$** ⇨ **Definition in:** *system/optimisationDict.optimisation.sensitivities*
- **Update design variables** ⇨ **Definition in:** *system/optimisationDict.optimisation.updateMethod*
- **Update mesh** ⇨ **Definition in:** *system/optimisationDict.optimisation.meshMovement*

(After updating the mesh, the loop returns to "Solve flow equations".)

Gradient-based Shape optimisation Loop

Define design variables b_n

Solve flow equations $\rightarrow U$

Compute J

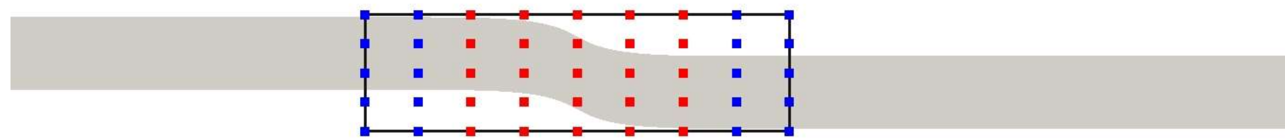
Solve adjoint equations $\rightarrow \Psi$

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

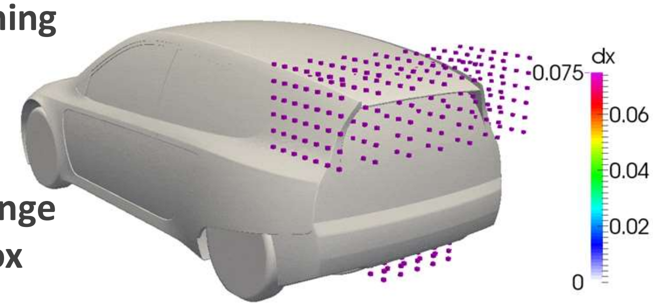
Update design variables

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

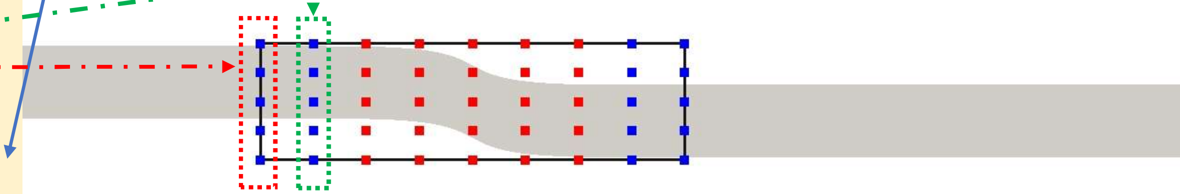
Define design variables b_n

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:

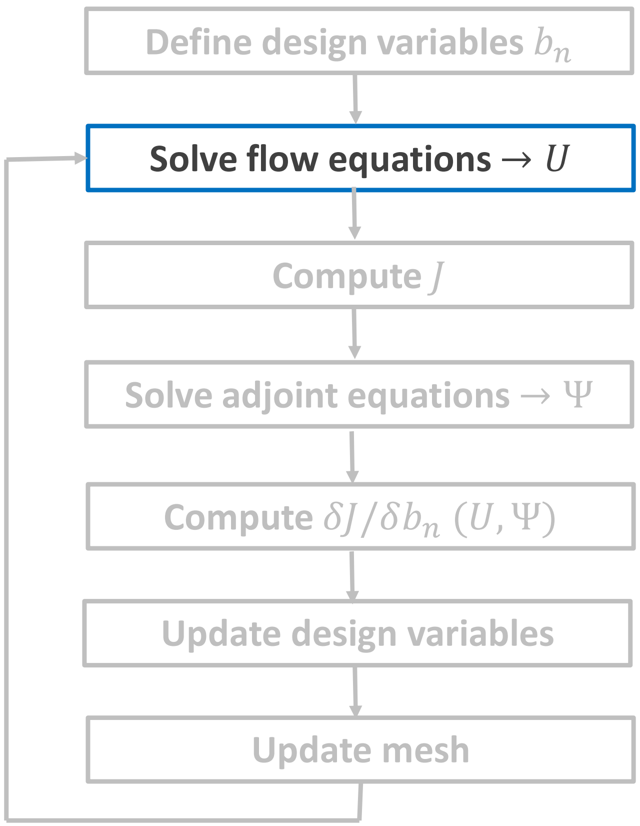
- **Number of control points (CPs) per box direction**
- **Degree per direction (smaller degree → more local support)**
- **CPs defined either aligned with a coordinate system (Cartesian, cylindrical) or given manually through a dictionary**
- **Possible to confine the movement of (some of) the CPs in 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



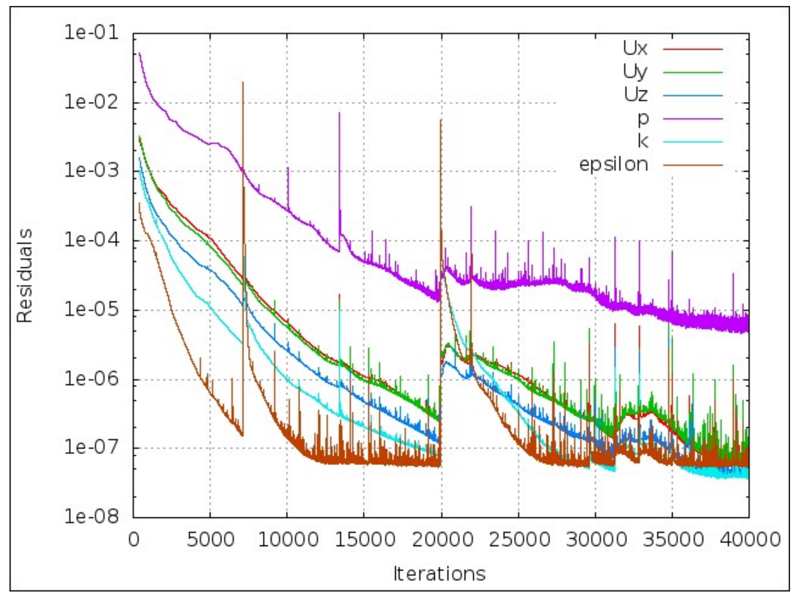
Defined in *system/optimisationDict.primalSolvers*

Incompressible, steady-state flows

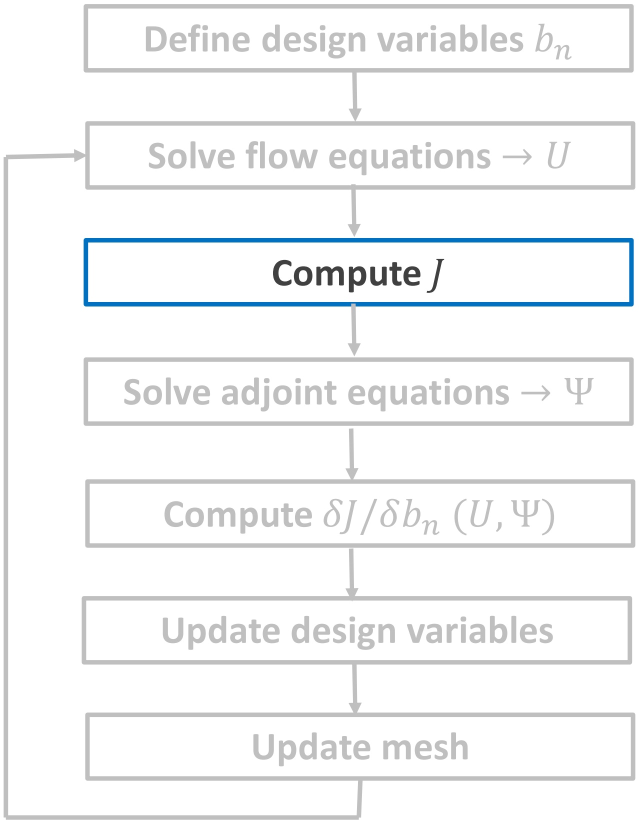
- **SIMPLE is incorporated into *adjointOptimisationFoam***
- **Multi-point optimisation supported; can define more than one primal solvers**

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



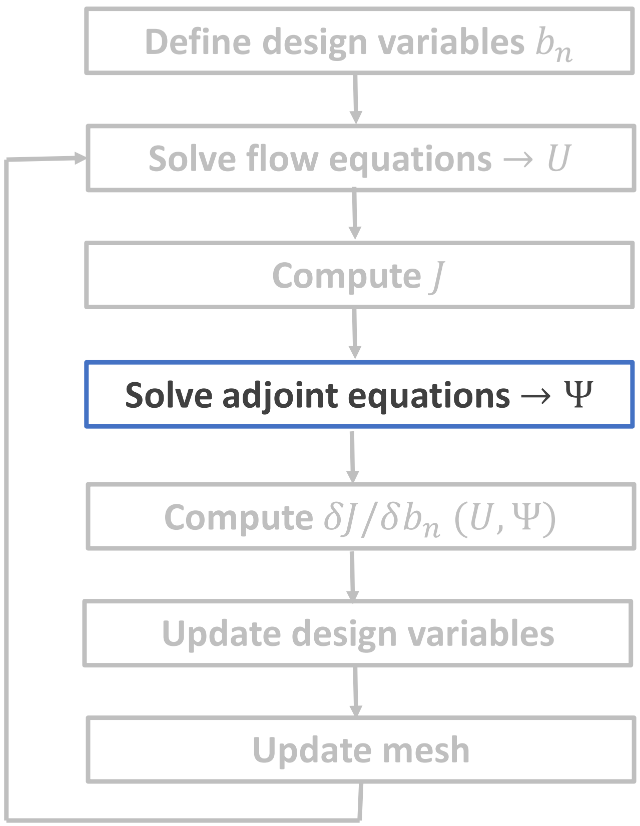
Defined in system/optimisationDict.adjointManagers, within each adjoint solver

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



Discretization in system/fvSchemes, Relaxation in system/fvSolution

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

$$R_i^u = u_j \frac{\partial v_j}{\partial x_i} - \frac{\partial (u_i v_j)}{\partial x_j} - \frac{\partial \tau_{ij}^a}{\partial x_j} + \frac{\partial q}{\partial x_i} + \frac{\partial J_{\Omega'}}{\partial v_i} = 0\ ,\ \ i = 1, 2(, 3)$$

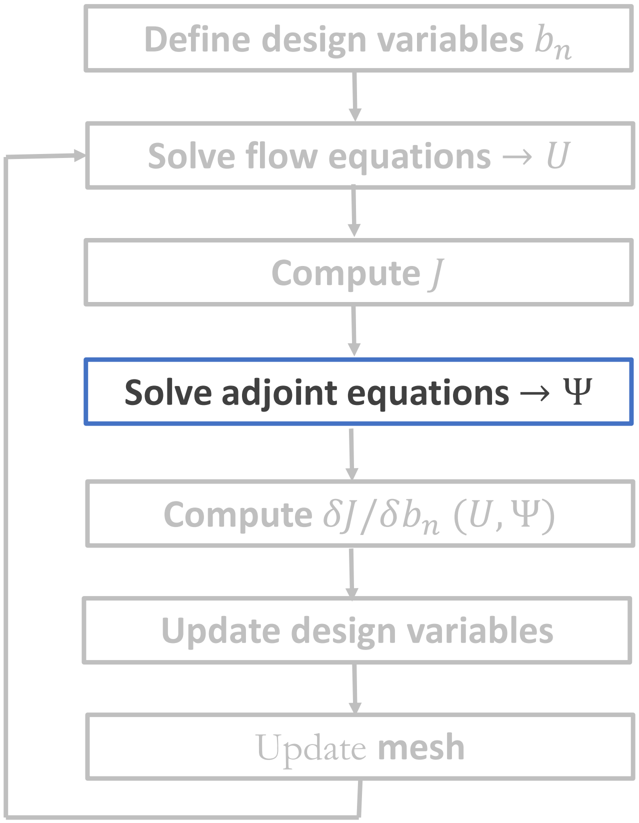
ATC AC

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



Defined in 0/pa and 0/Ua

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

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

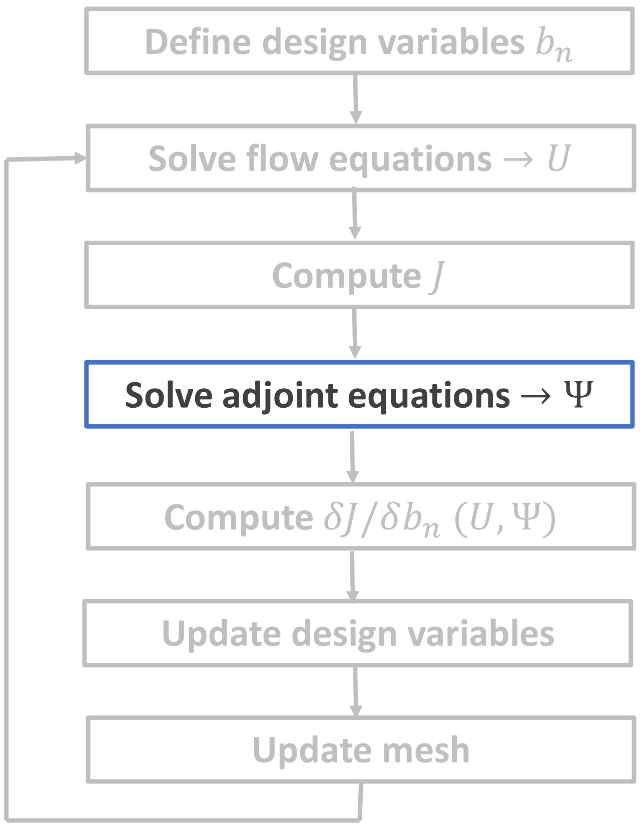
$$u^{II}_{\langle t \rangle} = u_i t^{II}_i = \frac{\partial J_{S_{I-W},k}}{\partial \tau_{ij}} n_k t^{II}_i n_j + \frac{\partial J_{S_{I-W},k}}{\partial \tau_{ij}} n_k t^{II}_j n_i$$

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**

Prof. K.C. Giannakoglou, kgianna@mail.ntua.gr, Dr. E. Papoutsis-Kiachagias, vaggelisp@gmail.com

Gradient-based Shape optimisation Loop



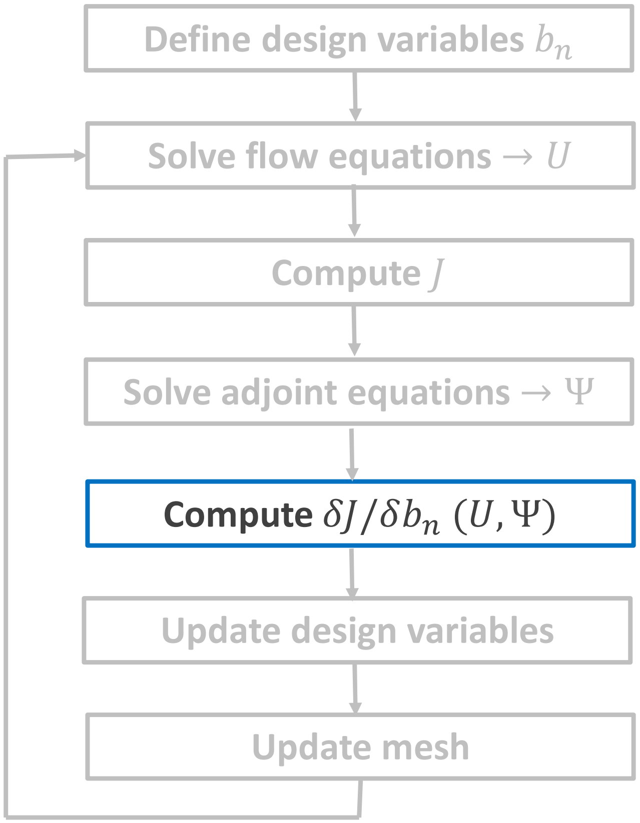
Defined in *system/optimisationDict.adjointManagers*

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



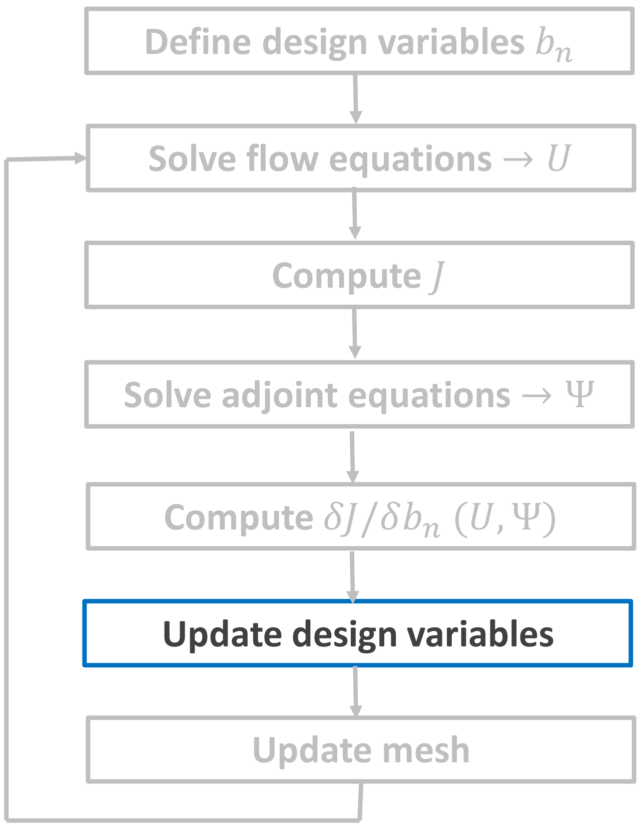
Defined in
system/optimisationDict.optimisation.sensitivities

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



Defined in
system/optimisationDict.optimisation.updateMethod

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

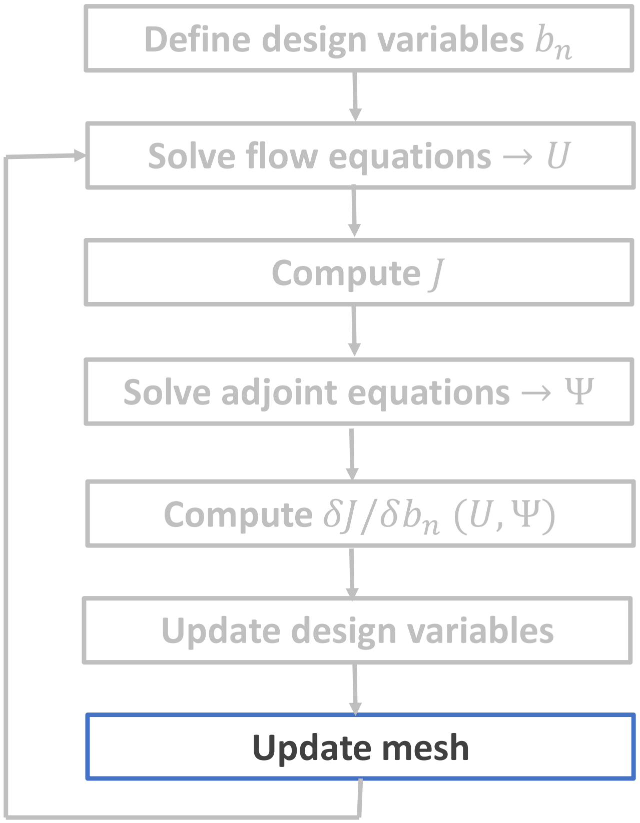
$$b_n^{new} = b_n^{old} + \eta s_n$$

- **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



Defined in
system/optimisationDict.optimisation.meshMovement

- **Need to translate $\Delta \mathrm{b_n}$ 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**

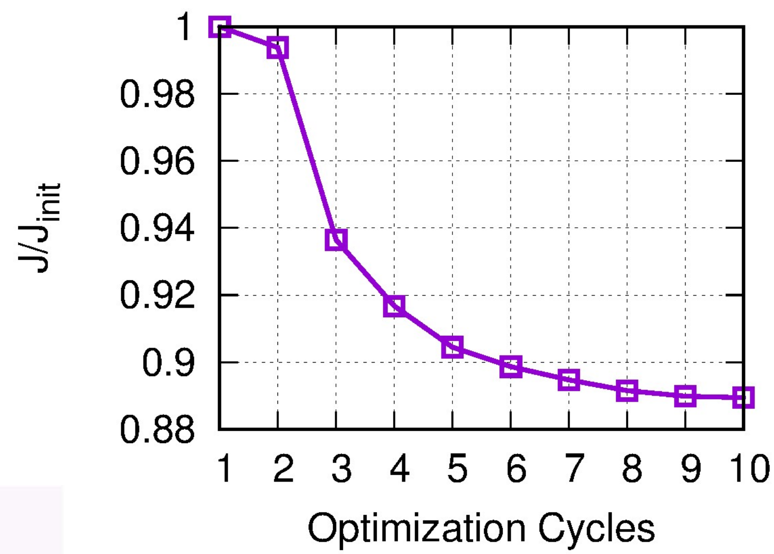
S-bend: optimisation results

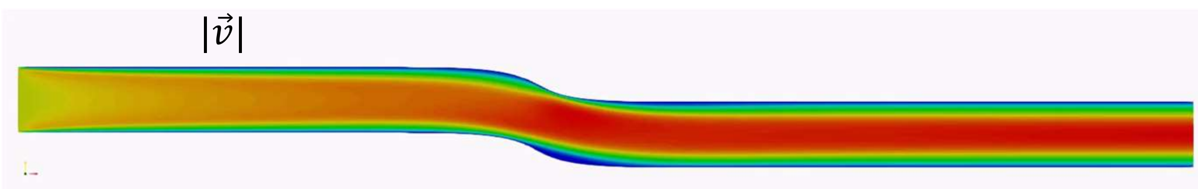
Run the optimization loop

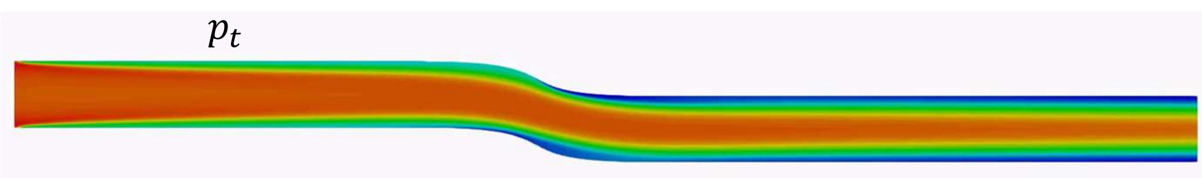
```
>> ./Allrun > log 2> err & (~2.5 min/4 procs)
```

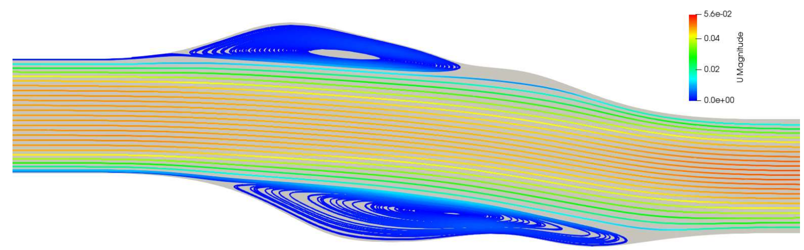
What to examine:

- **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!**



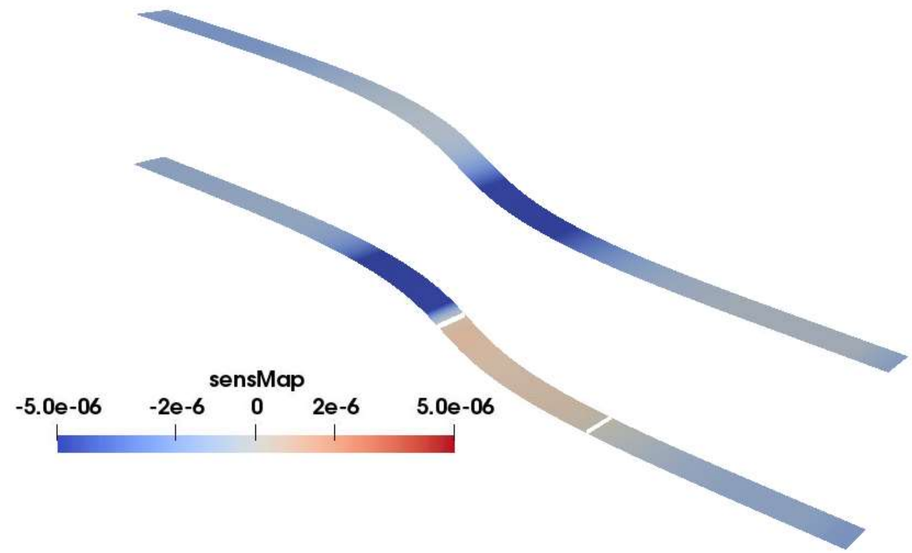






S-bend: Computing sensitivity maps

- **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: Computing sensitivity maps

- **Change to the sensitivity map folder**
  ```
  >> cd ~/2022_11_adjointTraining/sbend/laminar/sensitivityMap
  ```
- **Run *adjointOptimisationFoam*, configured to compute the sensitivity map**
  ```
  >> ./Allrun > log 2> err & (~0.5 min/4 processors)
  ```
- **Visualize the `pointSensNormalas1ESI` field in paraview. Use a symmetric scale!**
- **Inspect the differences in *optimisationDict and controlDict* between the optimisation and sensivitiy map cases**
  ```
  >> cd ~/2022_11_adjointTraining/sbend/laminar

  >> vim -d {optimisation,sensitivityMap}/system/optimisationDict

  >> vim -d {optimisation,sensitivityMap}/system/controlDict
  ```

If you are not familiar with the vim text editor, you may use **meld** or **kdiff3** visual editors to compare files

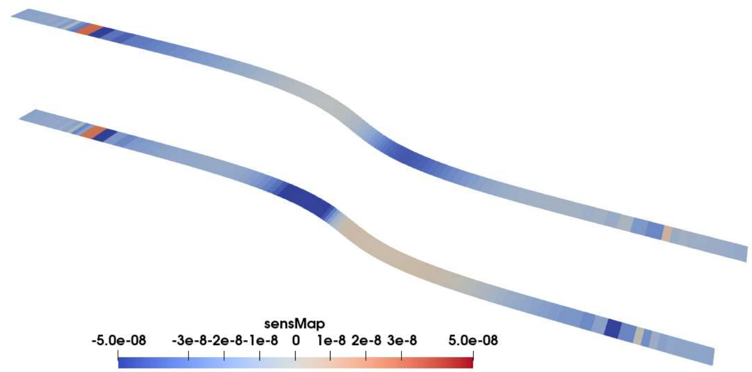
S-bend: Smoothing the sensitivity map

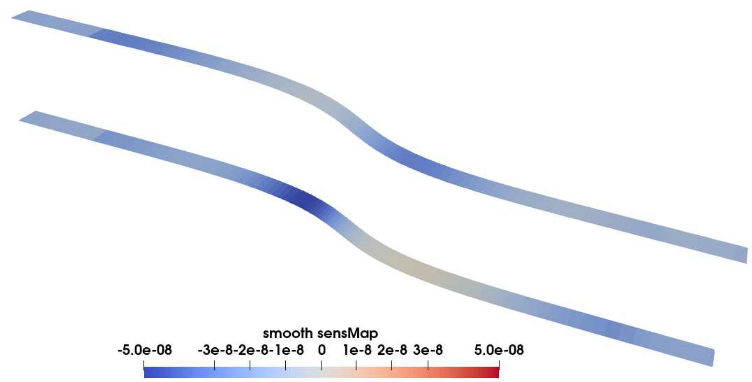
- **In more complex/industrial cases, checkerboards occur in the computed sensitivity maps.**
- **This problem becomes pronounced in meshes built with snappyHexMesh!**
- **The direction of favorable surface displacement becomes ambiguous...**
- **Smooth the sensitivity-map, G, by solving**

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

on a *finiteArea* mesh.

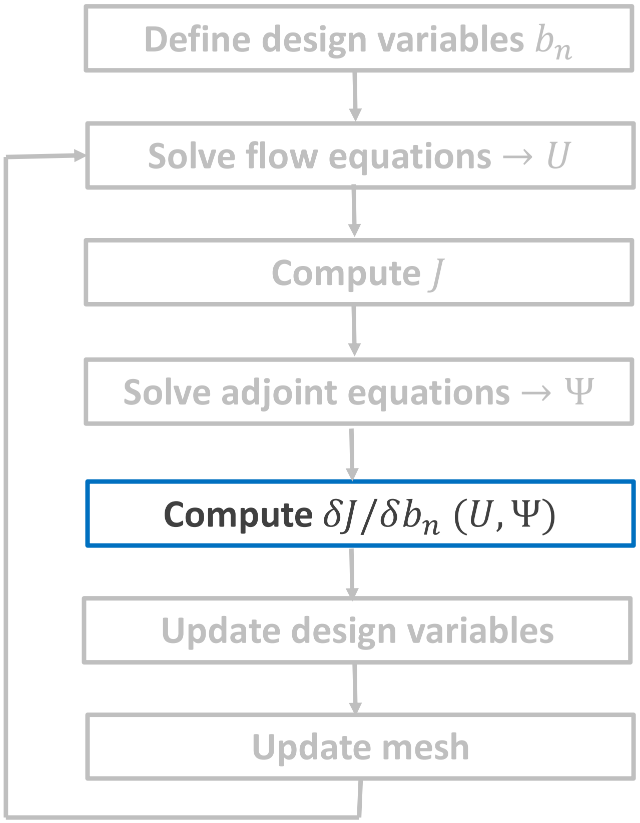
- **Be careful when comparing smoothed sensitivity maps on different meshes!...**





Prof. K.C. Giannakoglou, kgianna@mail.ntua.gr, Dr. E. Papoutsis-Kiachagias, vaggelisp@gmail.com

S-bend: Smoothing the sensitivity map – Additional Entries



- **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.**

S-bend: Smoothing the sensitivity map

- **Change to the smooth sensitivity map folder**
  `>> cd ~/2022_11_adjointTraining/sbend/laminar/smoothSensMap`

- **Run *adjointOptimisationFoam*, configured to compute the sensitivity map and smooth it for various radiuses**
  `>> ./Allrun > log 2> err & (~75 sec/1 processor)`

- **Visualize the `faceSensNormalas1ESIRmult10` and `smoothedSurfaceSensas1ESIRmult10` fields in paraview. Use a symmetric scale!**

- **Inspect the differences in *optimisationDict* between the sensitivity map and smooth sensivitiy map cases**
  `>> cd ~/2022_11_adjointTraining/sbend/laminar`

  `>> vim -d {sensitivityMap,smoothSensMap}/system/optimisationDict`

If you are not familiar with vim text editor, you may use **meld** or **kdiff3** visual editors to compare files

Revisiting the tutorial case – Turbulent flows

- **Change to the turbulent variant of the sbend case**
  `>> cd ~/2022_11_adjointTraining/sbend/turbulent/losses/`

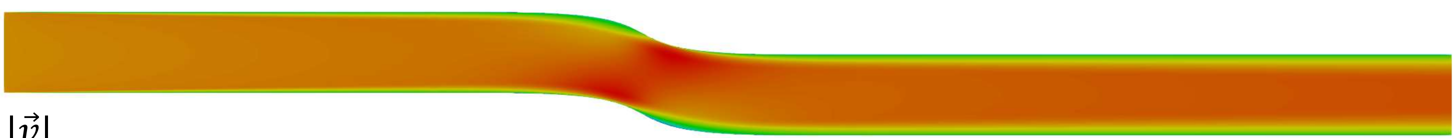
- **Case from** `$FOAM_TUTORIALS/incompressible/adjointOptimisationFoam/\`
  `shapeOptimisation/sbend/turbulent/opt/BFGS/op1`
  with the addition of the adjoint to k-Omega SST

- **Turbulent flow within an S-bend duct.**

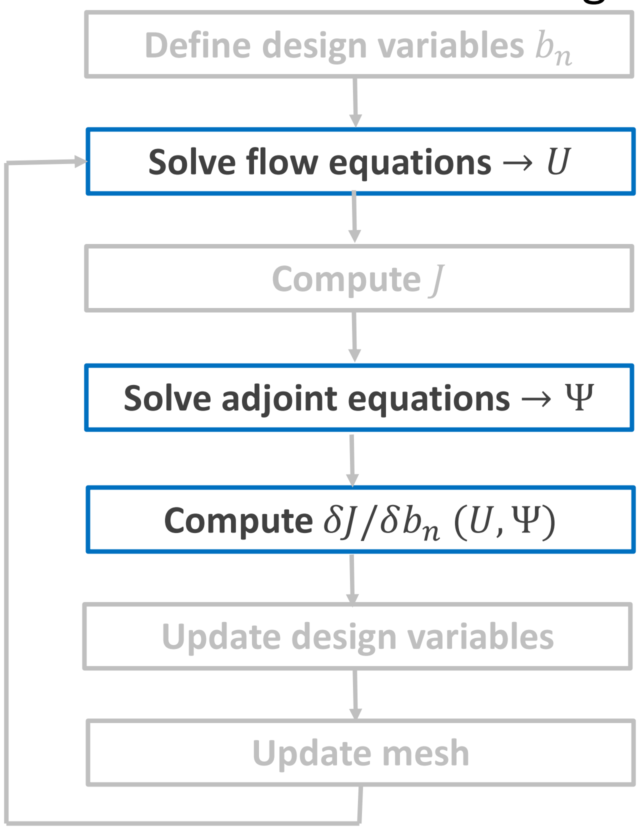
- $Re = \mathbf{1 \times 10^5}$

- **Spalart-Allmaras & k-ω SST turbulence models**

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



Turbulent flows: Changes – Additional entries

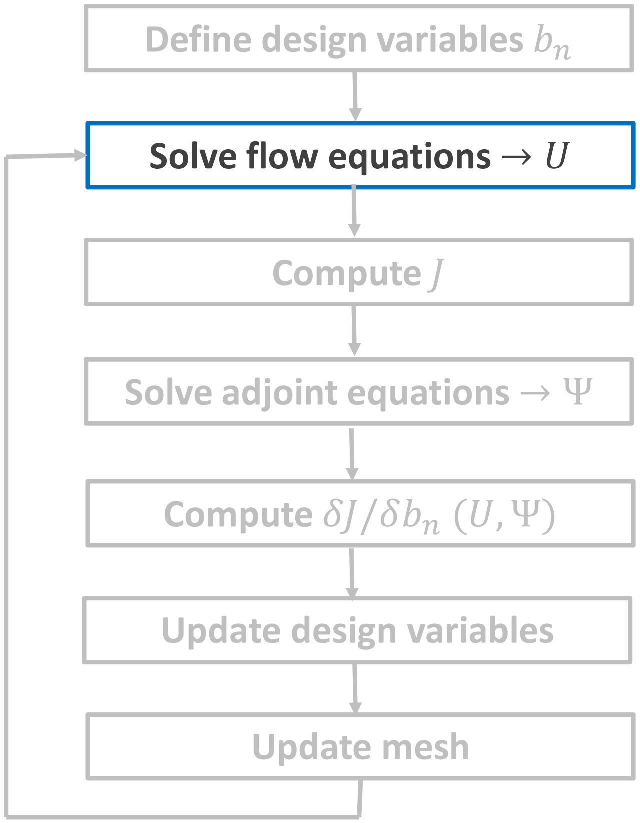


- **Flow equations:**
  SA: additional entries in *system/fvSchemes* related to distance calculations. Why is it important?

- **Adjoint equations: new PDEs to be solved for the adjoint turbulence model variables**

- **New terms to the sensitivity derivatives**

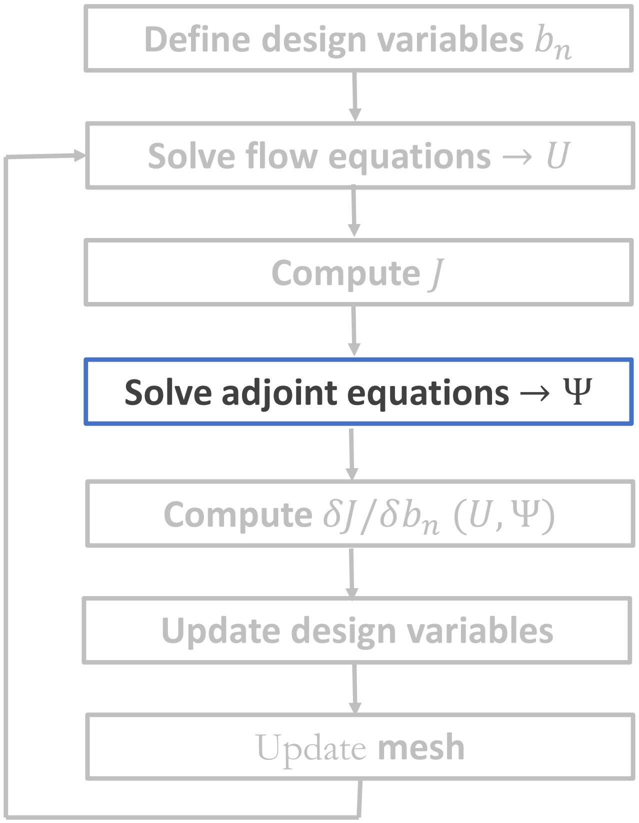
Turbulent flows: Changes – Additional entries



- **The Spalart-Allmaras PDE includes the distance from the wall in its production and destruction terms.**
- **Distance field changes due to changes in the geometry during the optimisation. Need to account for it in the adjoint formulation.**
- **A number of methods to compute the distance field. Choice through *system/fvSchemes.wallDist*.**
- **Typical method is *meshWave*. Not easily differentiable using adjoints since it is an algebraic method, not a PDE.**
- **Preferred method in combination with adjoints: *advectionDiffusion* (the eikonal or Hamiltion-Jacobi equation).**
- **Solves a PDE for *yWall* so additional entries are required in *fvSolution/fvSchemes*. Boundary conditions created automatically.**
- **Do not use bounded schemes for the convection term!**

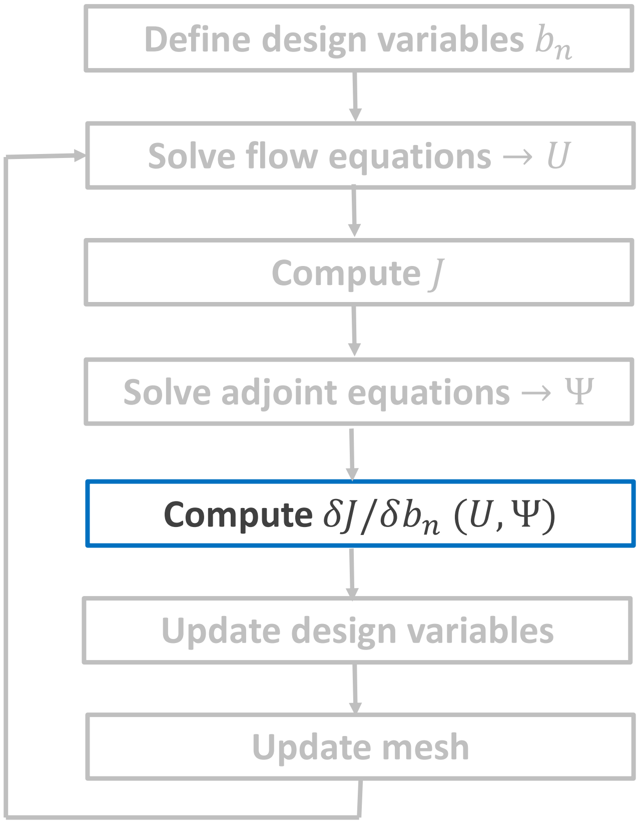
Prof. K.C. Giannakoglou, kgianna@mail.ntua.gr, Dr. E. Papoutsis-Kiachagias, vaggelisp@gmail.com

Turbulent flows: Changes – Additional entries



- **New PDE(s) to be solved for the adjoint turbulence model variable(s)**
  SA: nuaTilda,
  k-ω SST: ka, wa
  - **Adjoint turbulence model defined in *constant/adjointRASProperties***
  - **New entries in *fvSolution, fvSchemes***
  - **Boundary conditions defined in**
    SA: 0/nuaTilda,
    k-ω SST: 0/{ka, wa}

Turbulent flows: Changes – Additional entries



- **Adjoint to the eikonal PDE gives additional contributions to $\delta J/\delta b_n$**
  - **The adjoint eikonal PDE is decoupled from the rest of the adjoint PDEs. Solved when computing sensitivity derivatives to compute the adjoint distance field, *da*.**

$$R^{\Delta_a} = -2\frac{\partial}{\partial x_j}\left(\Delta_a \frac{\partial \Delta}{\partial x_j}\right) + \tilde{\nu}_a \tilde{\nu} C_\Delta = 0$$

  - **New entries in *fvSolution, fvSchemes*.**
  - **Boundary conditions created automatically.**
  - **(Optional) additional entries in the sensitivities part of *optimisationDict*. All have default values and can be omitted to ease the setup.**

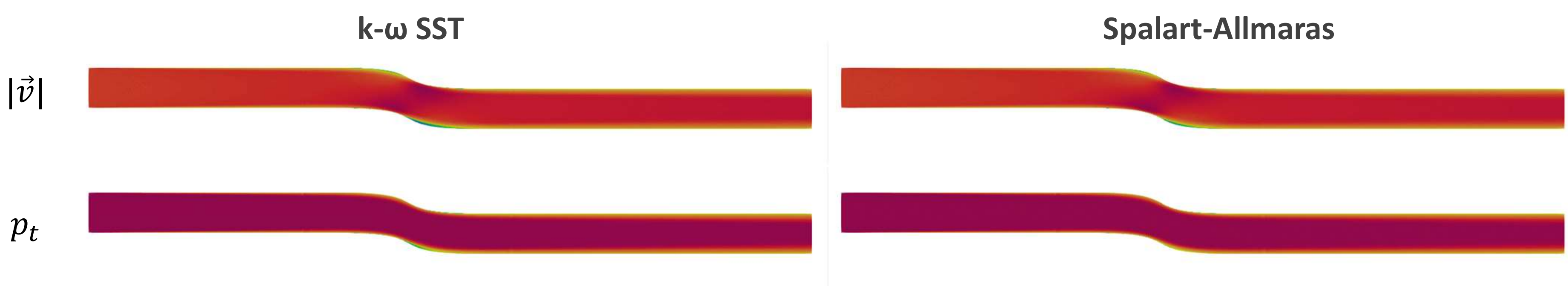
S-bend: optimisation results for a turbulent flow (1)

- **Run the optimization loops**

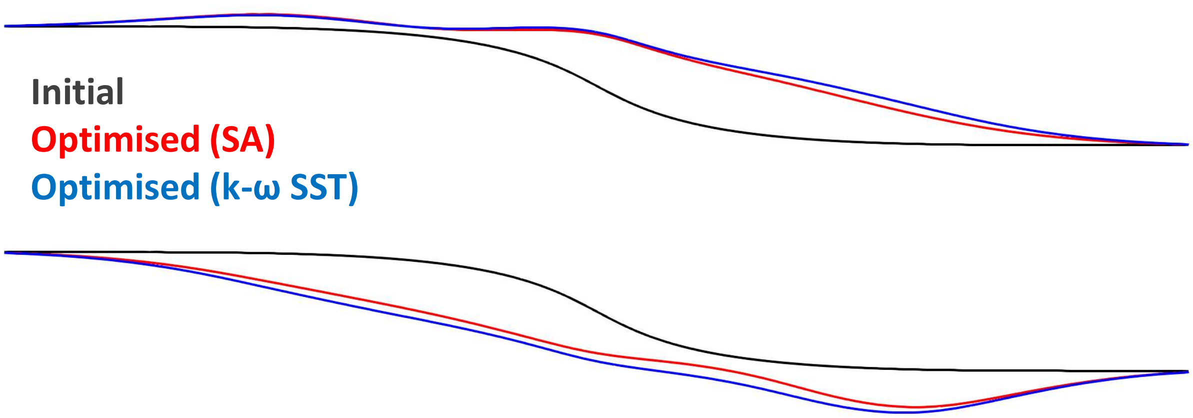
```
>> cd /2022_11_adjointTraining/sbend/turbulent/losses/SA
>>./Allrun.parallel > log 2> err &
(~8 min/4 procs)
>> cd /2022_11_adjointTraining/sbend/turbulent/losses/kOmegaSST
>> ./Allrun.parallel > log 2> err &
(~12 min/4 procs)
```

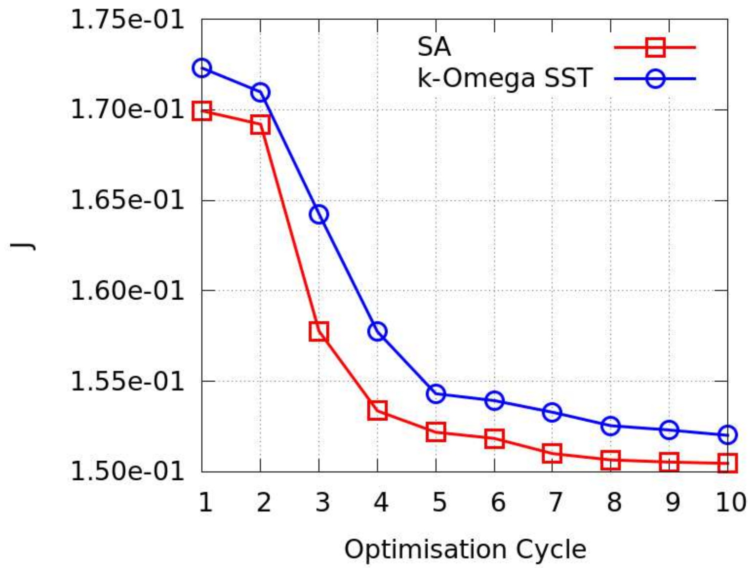
- **Same objective and parameterization as in the laminar case**

At a first glance, both the final geometries and the course of the optimisation look similar for both turbulence models



S-bend: optimisation results for a turbulent flow (2)





- **A close examination reveals minor differences in the optimised geometries**
- **They do follow the same trend however**
- **The objective is reduced by almost the same percentage (~11.5 %) with both turbulence models**

S-bend: comparison of optimal geometries for laminar and turbulent flows

Visualize the optimised geometries from the k-ω SST and Spalart-Allmaras cases in Paraview

- **Create the blank Paraview files**
```
>> touch ~/2022_11_adjointTraining/sbend/turbulent/kOmegaSST/foam.foam
>> touch ~/2022_11_adjointTraining/sbend/turbulent/losses/SA/foam.foam
```

- **Open them both in Paraview**

- **Overlay the two optimised geometries and their flow fields**

Takeaway messages:

- **Adjoint supports optimisation loops at a small CPU cost ($\sim$20 cycles $\rightarrow$ ~ 40 flow solutions)**

- **Ideal for both early stage development and refinement**

- **More optimisation types available**
  - **Topology optimisation (design of internal flows with known inlets/outlets)**
  - **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`

- **Adjoint to turbulent flows**
  `shapeOptimisation/sbend/turbulent/opt/BFGS/op1`

- **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