Introduction of AdjointShapeOptimizationFoam

Y. Takagi

November 10, 2012, OF b.@Kansai

Brief history of adjoint method

- 1969 Lions
  - Optimal control of systems governed by partial differential equations
- 1974 Pironneau
  - On optimum design in fluid mechanics
- 1988 Jameson
  - Aerodynamics design via control theory
- 1997 Giles, Pierce
  - Adjoint equations in CFD: duality, boundary conditions and solution behavior

Brief history of adjoint method

- 1997 Anderson, Venkatakrishnan
  - Aerodynamics design optimization on unstructured grids with a continuous adjoint formulation
- 2003 Borrvall, Peterson
  - Topology optimization of fluids in Stokes flow
- 2007 Othmer, Villiers, Weller
  - Implementation of a continuous adjoint for topology optimization of ducted flows
- 2008 Othmer
  - A continuous adjoint formulation for the computation of topological and surface sensitivities of ducted flows

Formulation of adjoint mothed

- C. Othmer, “A continuous adjoint formulation for the computation of topological and surface sensitivities of ducted flows”, Int. J. Num. Methods Fluids, 58, pp.861-877 (2008).

Formulation of adjoint method

- Optimization problem

  Minimize $J = J(\alpha,\mathbf{v},p)$ subject to $R(\alpha,\mathbf{v},p) = 0$

  where

  J : cost function

  α : porosity

  $\mathbf{v}$: velocity

  p : pressure

State equations

Incompressible, steady-state Navier-Stokes equations with porosity

$$(R_1, R_2, R_3)^{\mathrm{T}} = (\mathbf{v} \cdot \nabla)\mathbf{v} + \nabla p - \nabla \cdot (2\nu \mathrm{D}(\mathbf{v})) + \alpha \mathbf{v}$$

$$R_4 = -\nabla \cdot \mathbf{v}$$

where *R* is the state equations,

$$R = (R_1, R_2, R_3, R_4)^{\mathrm{T}}$$

Introduce a Lagrangian function *L*,

$$L := J + \int_{\Omega} (\mathbf{u}, q) R \, d\Omega$$

$$(\mathbf{u}, q) = (u_1, u_2, u_3, q)$$ (Lagrangian multipliers)

Variation of Lagrangian function

Total variation of *L*,

$$\delta L = \delta_\alpha L + \delta_{\mathbf{v}} L + \delta_p L$$

(Lagrangian multipliers are chosen to satisfy $\delta_{\mathbf{v}} L + \delta_p L = 0$)

$$= \delta_\alpha L = \delta_\alpha J + \int_\Omega (\mathbf{u}, q) \delta_\alpha R d\Omega$$

Then,

$$\frac{\partial L}{\partial \alpha_i} = \frac{\partial J}{\partial \alpha_i} + \int_\Omega (\mathbf{u}, q) \frac{\partial R}{\partial \alpha_i} d\Omega$$

Without explicit dependence of the cost function on the porosity,

$$\frac{\partial J}{\partial \alpha_i} = 0$$

Sensitivity

By considering the Dercy term in cell *i*,

$$\frac{\partial R}{\partial \alpha_i} = \begin{pmatrix} \mathbf{v} \\ 0 \end{pmatrix} \chi_i$$

Therefore, the desired sensitivity for each cell can be computed by

$$\frac{\partial L}{\partial \alpha_i} = \mathbf{u}_i \cdot \mathbf{v}_i V_i$$

Deviation of adjoint equations and boundary conditions

Decompose the cost function J into contributions from the boundary Γ and from the interior of Ω,

$$J = \int_{\Gamma} J_{\Gamma} d\Gamma + \int_{\Omega} J_{\Omega} d\Omega$$

····· (omitted)

Finally, the adjoint Navier-Stokes equaitons are derived as follows:

$$-2D(\mathbf{u})\mathbf{v} = -\nabla q + \nabla \cdot (2\nu D(\mathbf{u})) - \alpha \mathbf{u} - \frac{\partial J_{\Omega}}{\partial \mathbf{v}}$$

$$\nabla \cdot \mathbf{u} = \frac{\partial J_{\Omega}}{\partial p}$$

Specialization to ducted flows

- Adjoint N-S equations:

$$-2\mathrm{D}(\mathbf{u})\mathbf{v} = -\nabla q + \nabla\cdot\,(2\nu\mathrm{D}(\mathbf{u})) - \alpha\mathbf{u}$$

$$\nabla\cdot\,\mathbf{u} = 0$$

- Adjoint BCs for the wall and inlet:

$$\mathbf{u}_t = 0, \quad u_n = -\frac{\partial J_\Gamma}{\partial p}$$

$$\mathbf{n}\cdot\,\nabla q = 0$$

- Adjoint BCs for the outlet:

$$q = \mathbf{u}\cdot\,\mathbf{v} + u_n v_n + \nu(\mathbf{n}\cdot\,\nabla)u_n + \frac{\partial J_\Gamma}{\partial v_n}$$

$$0 = v_n\mathbf{u}_t + \nu(\mathbf{n}\cdot\,\nabla)\mathbf{u}_t + \frac{\partial J_\Gamma}{\partial \mathbf{v}_t}$$

Example 1: Dissipated power

Cost function:

$$J := -\int_\Gamma d\Gamma \left(p + \frac{1}{2}v^2\right)\mathbf{v}\cdot\mathbf{n}$$

$$J_\Omega = 0, \quad J_\Gamma = -\left(p + \frac{1}{2}v^2\right)\mathbf{v}\cdot\mathbf{n}$$

Derivatives for BCs:

$$\frac{\partial J_\Gamma}{\partial p} = -\mathbf{v}\cdot\mathbf{n},$$

$$\frac{\partial J_\Gamma}{\partial \mathbf{v}} = -\left(p + \frac{1}{2}v^2\right)\mathbf{n} - (\mathbf{v}\cdot\mathbf{n})\mathbf{v}$$

Adjoint BCs for the wall and inlet:

$\mathbf{u}_t = 0$ at wall

$u_n = \begin{cases} 0 \\ v_n \end{cases}$ at inlet

Adjoint BCs for the outlet:

$$q = \mathbf{u}\cdot\mathbf{v} + u_n v_n + \nu(\mathbf{n}\cdot\nabla)u_n - \frac{1}{2}v^2 - v_n^2$$

$$0 = v_n(\mathbf{u}_t - \mathbf{v}_t) + \nu(\mathbf{n}\cdot\nabla)\mathbf{u}_t$$

adjointShapeOptimization.C

```
laminarTransport.lookup("lambda") >> lambda;
alpha +=
    mesh.relaxationFactor("alpha")
   *(min(max(alpha + lambda*(Ua & U), zeroAlpha), alphaMax) - alpha);

zeroCells(alpha, inletCells);

 // Pressure-velocity SIMPLE corrector
 {
    // Momentum predictor
    tmp<fvVectorMatrix> UEqn
    (
       fvm::div(phi, U)
     + turbulence->divDevReff(U)
     + fvm::Sp(alpha, U)
    );
```

$+\alpha\mathbf{u}$

adjointShapeOptimization.C

```
// Adjoint Pressure-velocity SIMPLE corrector
    {
        // Adjoint Momentum predictor

        volVectorField adjointTransposeConvection((fvc::grad(Ua) & U));

        zeroCells(adjointTransposeConvection, inletCells);

        tmp<fvVectorMatrix> UaEqn
        (
            fvm::div(-phi, Ua)
          - adjointTransposeConvection
          + turbulence->divDevReff(Ua)
          + fvm::Sp(alpha, Ua)
        );
```

$\nabla \mathbf{u} \cdot \mathbf{v}$

$\nabla \cdot (-\phi \mathbf{u})$

$-\nabla \mathbf{u} \cdot \mathbf{v}$

$-\nabla \cdot (2\nu D(\mathbf{u}))$

$+\alpha \mathbf{u}$

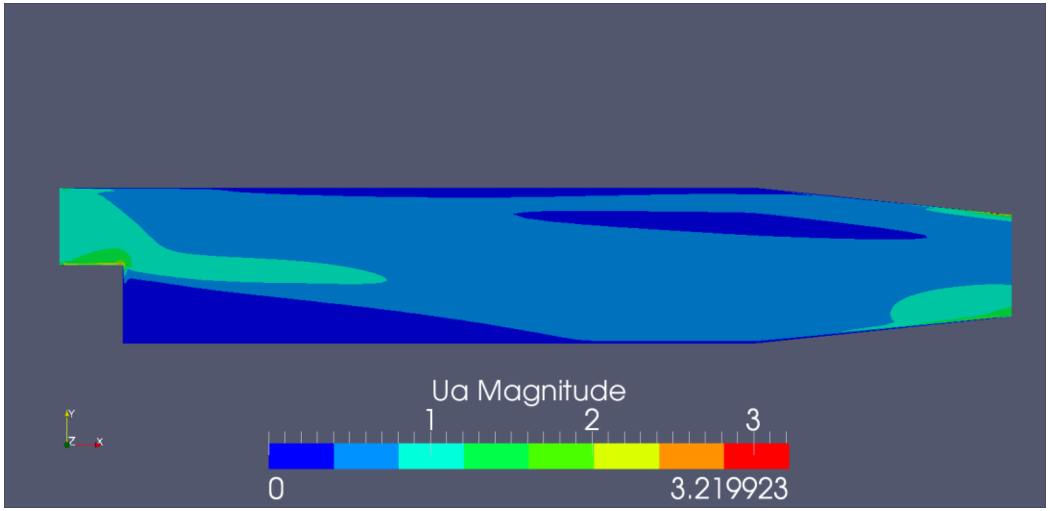
adjointOutletVelocityFvPatchVectorField.C

```
void Foam::adjointOutletVelocityFvPatchVectorField::updateCoeffs()
{
    if (updated())
    {
        return;
    }

    const fvsPatchField<scalar>& phiap =  patch().lookupPatchField<surfaceScalarField, scalar>("phia");

    const fvPatchField<vector>& Up =  patch().lookupPatchField<volVectorField, vector>("U");

    scalarField Un(mag(patch().nf() & Up));
    vectorField UtHat((Up - patch().nf()*Un)/(Un + SMALL));

    vectorField Uan(patch().nf()*(patch().nf() & patchInternalField()));

    vectorField::operator=(phiap*patch().Sf()/sqr(patch().magSf()) + UtHat);

    fixedValueFvPatchVectorField::updateCoeffs();
}
```

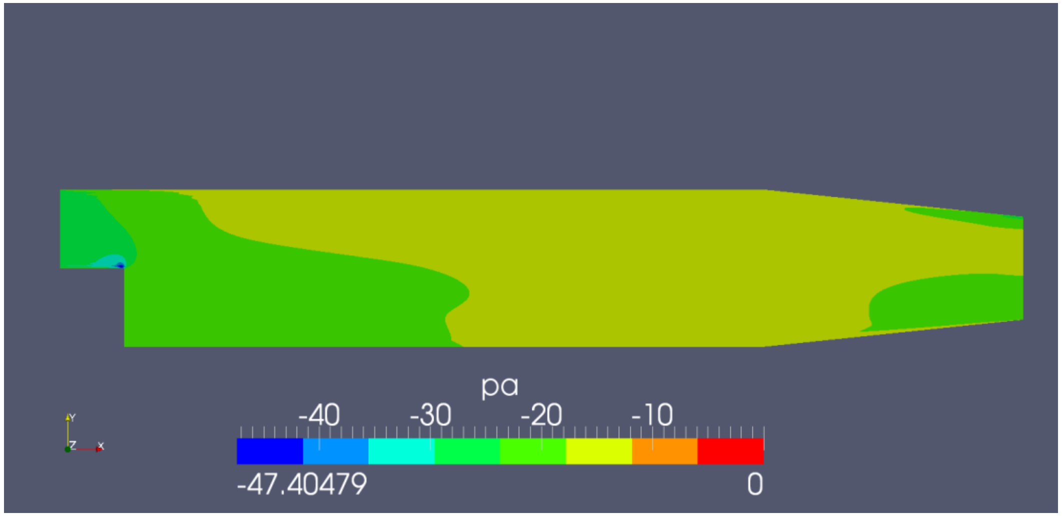
adjointOutletPressureFvPatchScalarField.C

```
void Foam::adjointOutletPressureFvPatchScalarField::updateCoeffs()
{
    if (updated())
    {
        return;
    }

    const fvsPatchField<scalar>& phip =   patch().lookupPatchField<surfaceScalarField, scalar>("phi");

    const fvsPatchField<scalar>& phiap =  patch().lookupPatchField<surfaceScalarField, scalar>("phia");

    const fvPatchField<vector>& Up =   patch().lookupPatchField<volVectorField, vector>("U");

    const fvPatchField<vector>& Uap =   patch().lookupPatchField<volVectorField, vector>("Ua");

    operator==((phiap/patch().magSf() - 1.0)*phip/patch().magSf() + (Up & Uap));

    fixedValueFvPatchScalarField::updateCoeffs();
}
```

Results: adjoint velocity



Result: adjoint pressure



Future work

- Implementation of sensitivity
- BCs for other examples
- Thermal convection problem

- Efficient optimization algorithms to deal with the computed topological and surface sensitivity maps
- Shape update algorithms to translate the shape sensitivities into a new and smooth shape