# Mode-based fluid-structure interaction simulation

Carl Troëng Niklas Wikström Mattias Liefvendahl

October 25, 2013

## 1 Introduction

In this document we present a method for fluid-structure interaction (FSI) simulation (see [1] for a not very good review of the research area). We describe both the theoretical formulation of the method, its implementation and its verification for test problems.

The method relies on a modal description of the structure, which is performed as a pre-processing step. Then the actual FSI-simulation is carred out in an `OpenFOAM`-solver which includes functionality for the modal deformation of the structure and its (time-resolved) coupling to the fluid motion.

The choice for structural solver for the initial test cases is `Comsol`. This may be revised at a later stage of developments.

In section 2, we describe the theory of the method, in section 3, we describe its implementation, and, in section 4, we describe its verification for test cases.

**DUMMY PICTURE**

Figure 1: A dummy picture ...

## 2 Theoretical formulation

We consider structures composed of an isotropic and homegeneous elastic material. We assume that the deformations are sufficiently small so that linear

elasticity is a good approximation. Furthermore, we typicall consider constant density, $\rho_s$, and elasticity parameters.

We want to represent the motion of the structure by a relatively small number of modes. For this purpose, we perform a modal analysis of the structure, as a pre-processing step before the FSI-problem is set up. The output of the modal analysis is a set of $N$ modes and $N$ frequencies,

$$\boldsymbol{\phi}_i(\mathbf{x}),\ \omega_i, \qquad i = 1, \ldots, N.$$

We use bold letters for vector quantities and normal font for scalar quantities. The modes should be normalized using the density of the solid in the following way.

$$\int_{V_s} \rho_s(\mathbf{x}) |\boldsymbol{\phi}(\mathbf{x})|^2 \, \mathrm{d}V = 1 \tag{1}$$

Here $V_S$ is the volume occupied by the structure (in the undeformed state), and we note that the density typically is constant and can be "moved outside" the integral.

Using the modes, we can represent the displacement, $\mathbf{u}$, of the structure as,

$$\mathbf{u}(\mathbf{x}, t) = \sum_{i=1}^{N} \alpha_i(t) \boldsymbol{\phi}_i(\mathbf{x}),$$

where we have introduced the time dependent mode coefficients, $\alpha_i$.

The fluid-structure coupling is obtained through the fluid forces on the surface of the structure. In the initial stage we only include the pressure forces, but at a later stage it may be useful to also include the viscous forces. Thus each structural coefficient satisfies an ODE, with a forcing term containing fluid forces.

$$\frac{\mathrm{d}^2 \alpha_i}{\mathrm{d}t^2} + \omega_i^2 \alpha_i = Q_i$$

The forcing is given by the following expression.

$$Q_i = -\int_S p \mathbf{n} \cdot \boldsymbol{\phi}_i \, \mathrm{d}S$$

Here, $\mathbf{n}$, is the local unit normal of the surface pointing into the fluid.

# 3 Implementation

The selection of implementation method is based on that the model should be useful together with any solver application supporting a dynamicFvMesh without changes to the application. The application of the model should be specific to the OpenFOAM case, not to the solver. Further, the user interface to the model should be comprehendible, and not limited to a predefined number of modes.

Therefore the entire modal analysis model is implemented as a separate class, that can be dynamically linked at runtime by the `libs ("library.so")` directive in `system/controlDict`. Since the solid structure is associated with fluid boundaries, the model is implemented as a boundary condition and since the structural modes are associated with boundary displacement, the boundary condition is a certain kind of point displacement boundary condition.

## 3.1 The modalDisplacement boundary condition

The boundary condition serves to calculate and set point displacement vectors for the patch points at each time step, based on fluid load and mode shapes achieved from a modal analysis.

Thus, the following steps must be handled:

1. Read mode shapes from modal analysis.
2. If parallel: Distribute mode shapes to processors.
3. Calculate the forcing term, $Q_i$ for each mode.
4. Solve the ODE for mode coefficients, $\alpha_i$.
5. Apply the resulting displacements to patch points.

To achieve this, the boundary condition consists of two parts; The boundary condition class itself, and an object handling the modes data and related calculations. The boundary condition class constructs the structural modes, when the boundary condition is constructed.

### 3.1.1 The structuralModes class

The class handling all modes calculations, is actually two classes: (1) A `structuralMode` class, that manages reading and calculations of a single mode and (2) a kind of a `List` class, `structuralModes`, that manages automatic construction of any number of `structuralMode` objects, read from a dictionary and that has wrapper functions, calling functions from each of its objects. The implementation is borrowed from e.g. the implementation of porous zones.

The boundary condition, `modalDisplacement`, thus constructs its own `structuralModes` list object, which in turn fills it self with all modes defined in the associated dictionary. When the boundary condition is scheduled for update (each time step), the boundary condition asks its structuralModes object for the current point displacements and applies these.

### 3.1.2 Parallelisation

A mode shape from the modal analysis, needed to create a `structuralMode`, consists of a list of vectors. Each vector in the list corresponds to the displacement of a point in the moving patch and the vector list has the same length

as the number of points in the corresponding patch. Specifically, the vector list corresponds to the the points of the patch in the *serical* case. Therefore, for parallel runs, the mode information must be carefully distributed to each process. This is managed by the boundary condition, which includes tables and functions that maps serial point labels to parallel counterparts and vice versa.

To enable the construction of the label maps, the pre-processing step must include generation of a file, containg a list of the serial case patch point labels. An application is written to create this file.

## 3.2 Usage of the modalDisplacement boundary condition

First of all, a solver that implements a `dynamicFvMesh` is needed. E.g. pimpleDyMFoam or interDyMFoam. The dynamicFvMesh requires two case files:

- A point field, `pointDisplacement`.
- A dictionary, `constant/dynamicMeshDict`, defining the distribution of the point motion.

Both of the above are commonly found in the tutorials and are not specific to the modalDisplacement boundary condition.

Secondly, and unique to the present implementation, is the dicitonaries and boundary condition definition needed for the modalDisplacement boundary condition. The dictionaries are all stored in the directory `constant/structuralModes/`, and consist of the following:

- `fluidProperties`: Defines fluid density, since the solver might be incompressible.
- `modeData_<patchName>`: Contains definition of each structural mode.
- `labels_<patchName>`: A list of the serial point labels for the patch. This file is generated by the `writePatchPointLabels` utility.
- `monitorPoints_<patchName>`: An optional list of point coordinates, which displacements are reported on standard output.

The `modeData_...` dictionary consists of a list of sub-dictionaries. Each describing one structural mode. For a simulation including two modes only, it has the following form (excluding the header):

```
...
2
(
    first
    {
        generateMode no;
        frequency     10;
```

```
        scalingFactor 1;
        modeDisplacement
        #include "firstModeDisplacements.dat"
    }

    second
    {
        generateMode yes;
        generatedMode
        {
            origin  (0 0 0);
            axis    (0 0 1);
            waveLength   0.5;
            amplitude (0.05 0 0);
        }

        frequency     25;
        scalingFactor 1;
        modeDisplacement 0();
    }
)
```

The first mode shape is based on existing data residing in the file `firstModeDisplacements.dat`. This data file contains a vectorField with displacements corresponding to the patch points. The second mode examplifies the posibility to let the boundary condition generate a mode. Presently this mode is a simple trigonometric function, suitable for beams.

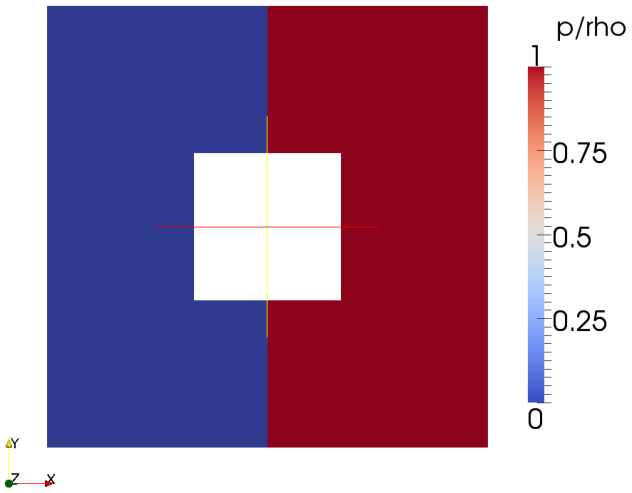
# 4 Test cases

The following is a tentative list of test cases for the algorithm.

1. A moving box in a fixed pressure field. No FSI, but testing of several components in the program.
2. A beam mounted perpendicular to the flow. The beam is fixed to the floor at one end and free to move at the other.
3. An elastic hydrofoil with varying angle of attack, [2].
4. A propeller blade in a varying inflow.
5. A hull structure excited by non-steady propulsion force.

The beam test case is fundamental for the verification of the algorithm and the implementation, and several tests should be performed for the beam. Apart from cross-flow, the beam should be tested in a still fluid, to check the change in oscillation frequency due to the added mass of the surrounding fluid. When

Figure 2: The pressure distribution for the moving box test case, on the plane $z = 0$.




started in cross-flow, the beam should first bend in the flow direction and then, when vortex shedding starts, it should go over to oscillations in the transversal direction.

## 4.1 The moving box

This test case does not contain fluid-structure interaction but is designed to only check the following parts of the implementation.

- Input and output concerning the structural mode description.
- Construction of the right hand side in the ODE for the mode coefficients.
- Solution of the ODE for the mode coefficients.

The geometry is a rectangular box inside a cube. The sides of the boxes are aligned with the coordinate directions. The outer box is stationary and the inner box can move as a rigid body along the coordinate axes. The geometrical parameters of the problem are given in table 1. The motion of the body is driven by a prescribed pressure field which is contant in time and is given by the expression,

$$p = \begin{cases} \Delta p & \text{for} \quad x > 0.5 \\ 0 & \text{for} \quad x \leq 0.5 \end{cases}$$

The pressure jump, $\Delta p$, and other physical parameters are given in table 2, and the pressure distribution is illustrated in figure 2.

The modal motion of the box is represented by three modes with the frequencies,

$$f_1 = 1\,\text{Hz}, \qquad f_2 = 2\,\text{Hz}, \qquad f_3 = 3\,\text{Hz},$$

Table 1: The dimensions of the outer and inner boxes. All quantities are given in meters, square meters or cubic meters as appropriate.

| | Outer box | Inner box |
|---|---|---|
| $x_{min}$ | 0 | $1/3 \approx 0.3333$ |
| $x_{max}$ | 1 | $2/3 \approx 0.6667$ |
| $y_{min}$ | 0 | $1/3 \approx 0.3333$ |
| $y_{max}$ | 1 | $2/3 \approx 0.6667$ |
| $z_{min}$ | 0 | $3/7 \approx 0.4286$ |
| $z_{max}$ | 1 | $4/7 \approx 0.5714$ |
| $V$ | 1 | $1/63 \approx 0.0159$ |
| $A_x = A_y$ | 1 | $1/21 \approx 0.0476$ |
| $A_z$ | 1 | $1/9 \approx 0.1111$ |

Table 2: Physical parameters. Note that the solid density can be calculated from the prescribed modal functions, as described in the text.

| Quantity | Notation | Value | Unit |
|---|---|---|---|
| Fluid density | $\rho_f$ | 1000 | $\mathrm{kg/m^3}$ |
| Pressure jump | $\Delta p$ | 1000 | Pa |

and mode shapes corresponding to rigid body motion in the three coordinate directions,

$$\boldsymbol{\phi}_1(\mathbf{x}) = a\mathbf{e}_x \qquad \boldsymbol{\phi}_2(\mathbf{x}) = a\mathbf{e}_y \qquad \boldsymbol{\phi}_3(\mathbf{x}) = a\mathbf{e}_z,$$

where, $a = 0.1\,\mathrm{m}$.

The only way that the solid density enters the algorithm is through the normalization of the modes. Given the information above, we can thus calculate which structural density, $\rho_s$, it corresponds to.

$$1 = \int_{V_s} \rho_s |\boldsymbol{\phi}(\mathbf{x})|^2 \,\mathrm{d}V = \rho_s V_i \cdot a^2$$

Using this expression and the inner box volume given in table 1, we obtain, $\rho_s = 6300\,\mathrm{kg/m^3}$.

For this case, the forcing integrals can be evaluated analytically. For the first mode, we have,

$$Q_1 = -\int_S p\mathbf{n} \cdot \boldsymbol{\phi}_i \,\mathrm{d}S = -\Delta p A_x a \approx -4.76\mathrm{Nm}.$$

For the second and third modes, the forcing is zero, $Q_2 = Q_3 = 0$.

### 4.1.1 Analytical solution of first mode motion

The motion of the block can thus be calculated by the solution of the ODE for the first mode coefficient,

$$\frac{\mathrm{d}^2\alpha_1}{\mathrm{d}t^2} + \omega_1^2\alpha_1 = Q_1,$$

with inital data,

$$\alpha_1(0) = 0, \qquad \dot{\alpha}(0) = 0.$$

The solution is given by the following convolution expression, for a general time dependent forcing $Q_1(t)$, which then can be simplified since in the present case, the forcing is constant.

$$\alpha_1(t) = \frac{1}{\omega_1}\int_0^t \sin\left[\omega_1(t-\tau)\right] Q_1(\tau)\,\mathrm{d}\tau = \frac{Q_1}{\omega_1^2}(1-\cos\omega_1 t)$$

### 4.1.2 Computed motion of the first mode

The problem is solved by a program which involves all the functionality related to the structural motion, but without the fluid solver. As described above, only a pressure field (constant in time) is provided, which drives the structural motion. Now we describe how input is provided to the program.

1. The modes are described in the file:
   `constant/structuralModes/modeData_movingBlock`

Here is a segment of this input file relating to the first mode:

```
first
{
    generateMode no;
    generatedMode
    {
        origin  (0 0 0);
        axis    (0 0 1);
        waveLength   1;
        amplitude (0.1 0 0);
    }

    frequency   1;
    scalingFactor 0.1;
    modeDisplacement 170
    (
        (1 0 0)
        (1 0 0)
        (1 0 0)
        (1 0 0)
        (1 0 0)
```

2. The input to the mesh deformation is given in the file:
   `constant/dynamicMeshDict`

This file contains the following information:

```
dynamicFvMesh      dynamicMotionSolverFvMesh;

motionSolverLibs ("libfvMotionSolvers.so");

solver            displacementLaplacian;

diffusivity       inverseDistance (movingBlock);
```

3. The remaining input, initial grid, pressure field, controls for time stepping etc, are given in the same way as for a standard simulation case. The case is simulated on the time interval, $0 < t < 10\,\mathrm{s}$, with a time step, $\Delta t = 0.01\,\mathrm{s}$.

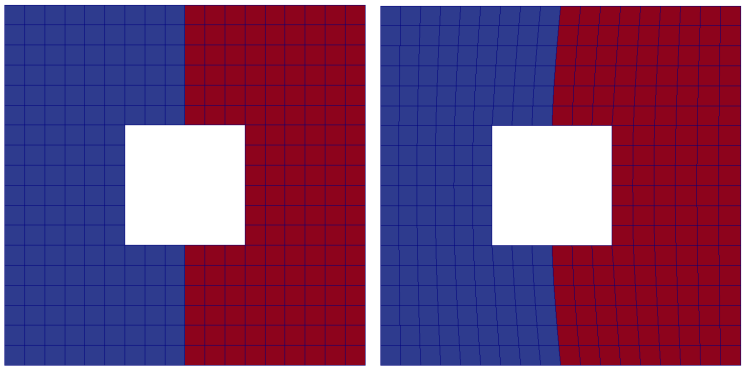


Figure 3: Initial (undeformed) stat to the left, and maximal displacement, at $t = 0.5\,\mathrm{s}$, to the right.

In figure 3, we illustrate the grid deformation in the most deformed state, at $t = 0.5\,\mathrm{s}$, and compare it to the initial grid.

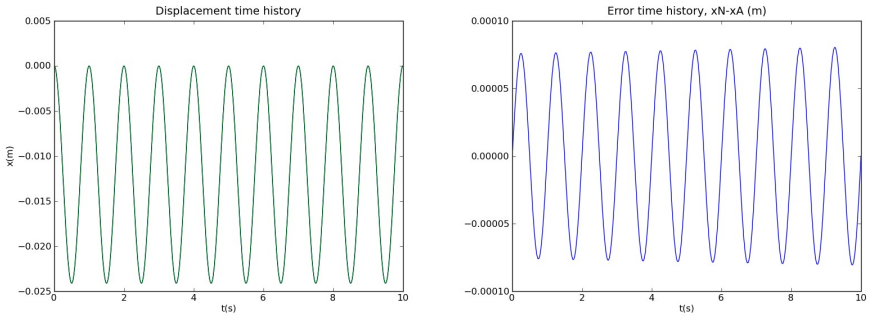



Figure 4: Box displacement (in $x$-direction) as function of time, $x_N(t)$, to the left, and error, $x_N(t) - x_A(t)$, to the right.

In order to check the computed motion, we post-process the computed displacement for a point on the box. The displacement is only in the $x$-direction,

and we denote it by $x_N(t)$, where the sub-script indicates "numerical". We also compare this to the exact analytical solution for the displacement, which we denote $x_A(t)$. The displacement and the error are shown in figure 4. We note that the error is less than 0.34% of the oscillation amplitude over the whole simulation interval.

## 4.2 The beam

Cantilever beam = fixed at one end ...

The material parameters seem to correspond to something like rubber ...

Table 3: Material and geometrical parameters of the cantilever beam with quadratic cross section and cross-sectional area $D^2$.

| Quantity | Notation | Value | Unit |
|---|---|---|---|
| Modulus of elasticiy | $E$ | 0.03 | GPa |
| Poisson's ratio | $\nu_s$ | 0.3 | - - - |
| Density | $\rho_s$ | 5000 | kg/m$^3$ |
| Length | $L$ | 0.1 | m |
| Width | $D$ | 0.02 | m |

### 4.2.1 Mode generation, interpolation to CFD and scaling

Two modal analyses are performed in two softwares. Comsol Multiphysics and Ansys. Both softwares delivered the same mode shapes, but the mode normalisation was only successfully made in Ansys. The latter has as default normalisation option to scale the modes with the *mass matrix*, which turns out to comply with the above equation 1, as discussed below.

Mode data are exported from the structural solver as displacement vectors at each solid mesh point, in ascii column data file format, where each row contains point coordinates and corresponding displacement. This data then be interpolated to the CFD boundary using the implemented OpenFOAM utility `interpolateModeData`. In short, this utility reads a set of "mode files" exported from the structural solver, and writes their interpolated counterparts to another set of files. The latter is then included in the dictionary defining the modes to be included in the OpenFOAM FSI.

To be able to assert that each mode fulfills equation 1, another utility is written: `calculateModeScaleFactor`. This utility requires the same dictionary input as the FSI case, but must be run on the solid itself. That is, a mesh in the solid region needs to be generated. In the case of the cantilever beam, this is trivial.

When mode data for the first five modes from Ansys are interpolated to the OpenFOAM case consisting of the solid beam, `calculateModeScaleFactor` is run on each of the modes and the result is assuring: For all bending modes the scale factor calculated by the OpenFOAM utility is very close to unity. The

exception is the third torsion mode, for which the scale factor seem to be more difficult to calculate based on displacements in a cartesian system.

To conclude: Modes exported from a structural solver can be interpolated to the OpenFOAM case and in case the normalisation of these modes is believed to disagree with equation 1, the scale factor can be calculated using an OpenFOAM case consisting of the solid only.

# References


[1] E. H. Dowell and K. C. Hall. Modeling of fluid-structure interaction. *Annu. Rev. Fluid Mech.*, 33:445–490, 2001.

[2] A. Ducoin and Y. L. Young. Hydroelastic response and stability of a hydrofoil in viscous flow. In *2nd Int. Symp. on Marine Propulsors*, Hamburg, Germany, June 2011.