Implementation of Simple FSI Model with `functionObject`

Matvey Kraposhin, **Ilia Marchevsky**

Institute for System Programming of RAS
Bauman Moscow State Technical University

Guimarães
June 28th, 2016

Outline

1. **Training course materials**
2. **What is FSI**
   - Examples of FSI in nature and engineering practice
   - Different approaches for solving FSI problems
   - Coupling strategies for partitioned approach
   - FSI-simulation applications architectures
3. **FSI model problem**
   - FSI example: circular cylinder wind resonance
   - Chosen solution approaches
4. **How to implement extensions for OpenFOAM**
   - Different strategies to extend OpenFOAM
   - fvOption facility
   - functionObject facility
5. **How to implement FSI with functionObject**
   - FSI example: circular cylinder wind resonance
   - "Hello, World" functionObject
   - Simplest coupling strategy implementation
   - Restart implementation
6. **Numerical example**
   - Validation example for laminar flow
   - Turbulent flow example

1 Training course materials

Training course materials

- Location of the course:
  `https://github.com/unicfdlab/TrainingTracks/`
- Folder `simpleFsi-OF3.0.0` for `OpenFOAM 3.0.0` version of this course

No.	Name	Description
1	cases	Cases that will be used to demonstrate `functionObject`'s created during the track
2	geometry	Contains geometry and mesh files created with SALOME platform, version 7.3.0
3	papers	Papers that were used in this course. If paper is open-access, then the PDF is placed, otherwise only the reference
4	src	Source code of `functionObject` classes considered in this track
5	materials	This presentation and other materials that were used in this course

2 What is FSI

- Examples of FSI in nature and engineering practice
- Different approaches for solving FSI problems
- Coupling strategies for partitioned approach
- FSI-simulation applications architectures

Examples of FSI in nature and engineering practice

What is FSI?

- **Fluid-Structure-Interaction**
- **Describes interaction between fluid (liquid or gas) and solid body (structure) in a system**
  - fluid interacts with a solid structure, exerting pressure that may cause deformation or displacement in the structure and, thus, alter the flow of the fluid itself
- **Typically connected with "bad" things**
  - fluttering of airplanes
  - deformations
  - vibrations
  - collapse of constructions
- **Interesting for many researchers in physics, mathematics and computer science**

Tacoma Narrows Bridge Collapse (USA, 1940)

Source: http://www.youtube.com/watch?v=nFzu6CNtqec

Volgograd 'Dancing' Bridge (Russia, 2010)

Source: http://www.youtube.com/watch?v=GORcnngwJ_Q

VIVACE Energy Generator

Source: http://www.youtube.com/watch?v=IcR8HszacOE

Flow simulation around movable structures (1)

Lagrangian description

- fluid particles carry their own properties (density, momentum, *etc.*)
- $\rho(p,\, t)$, $V(p,\, t)$, $P(p,\, t)$
- low numerical viscosity
- arbitrary body motion & deformation
- may be computationally expensive
- SPH, PFEM, Vortex Methods, *etc*

SPH-method

http://youtube.com/watch?v=EcAZv5xcvn8

Viscous Vortex Domains method (VVD)

http://youtube.com/watch?v=H-snLmMQKOY

Flow simulation around movable structures (2)

Eulerian description

- flow properties at every point in space
- $\rho(x,\,t)$, $V(x,\,t)$, $P(x,\,t)$
- not very large displacement & rotation
- requires mesh deformation/reconstruction
- 'body fitted' mesh methods

ALE description

- Arbitrary Lagrangian-Eulerian approach
- Overset meshes (Chimera, *etc*)
- Immersed boundary (IB) methods

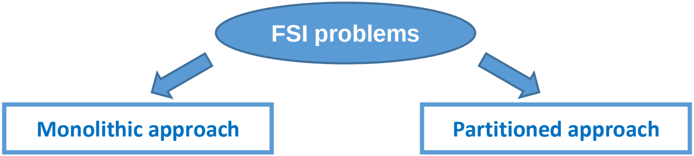
Body-fitted mesh

http://youtube.com/watch?v=mt2wv5P5zaY

LS-STAG immersed boundary method

http://youtube.com/watch?v=H-snLmMQK0Y

Different approaches for solving FSI problems

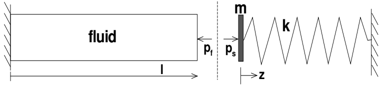


Monolithic approach

- Treats coupled fluid and structure equations simultaneously
- System is in general nonlinear, solution involves Newton's method
- **Advantages:**
  - high accuracy & stability
- **Disadvantages:**
  - expensive computation of derivatives (Jacobian matrix)
  - loss of software modularity due to the simultaneous solution of fluid and structure

Partitioned approach

Example: The piston problem
(Interface region expanded for clarity).



Basic ideas

- Systems spatially decomposed into partitions
- Solution is separately advanced in time over each partition
- Partitions interact on their interface
- Interaction by transmission and synchronization of coupled state variables

Advantages & Disadvantages

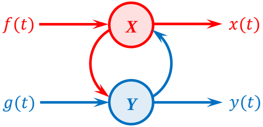
- **Advantages:**
  - customization
  - independent modeling
  - software reuse
  - modularity
- **Disadvantages:**
  - requires careful formulation and implementation to avoid serious degradation in stability and accuracy
  - parallel implementations are error-prone

Michler C., Hulshoff S.J., van Brummelen E.H., de Borst R. A monolithic approach to fluid-structure interaction // *Computers & Fluids*. 2004. Vol. 33, Is. 5–6. P. 839–848

Example: Monolithic approach

Governing equations:

$$\begin{cases} 3\dot{x} + 4x - y = f(t), \\ \dot{y} + 6y - 2x = g(t) \end{cases}$$



Backward Euler scheme:

$$\begin{aligned} x^{n+1} &= x^n + \dot{x}^{n+1}\Delta t, \\ y^{n+1} &= y^n + \dot{y}^{n+1}\Delta t \end{aligned}$$

Monolithic coupling scheme

Purely implicit discretization scheme leads to common linear system for new state (x^{n+1}, y^{n+1}) of all coupled subsystems:

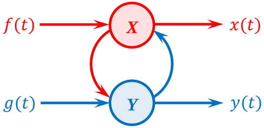
$$\begin{pmatrix} 3 + 4\Delta t & -\Delta t \\ -2\Delta t & 1 + 6\Delta t \end{pmatrix} \begin{pmatrix} x^{n+1} \\ y^{n+1} \end{pmatrix} = \begin{pmatrix} f^{n+1}\Delta t + 3x^n \\ g^{n+1}\Delta t + y^n \end{pmatrix}$$

Felippa C.A., Park K.C., Farhat C. Partitioned analysis of coupled mechanical systems // *Department of Aerospace Engineering Sciences and Center for Aerospace Structures University of Colorado at Boulder Boulder*. 1999. Report No. CU-CAS-99-06. 28 p.

Example: Partitioned approach

Governing equations:

$$\begin{cases} 3\dot{x} + 4x - y = f(t), \\ \dot{y} + 6y - 2x = g(t) \end{cases}$$

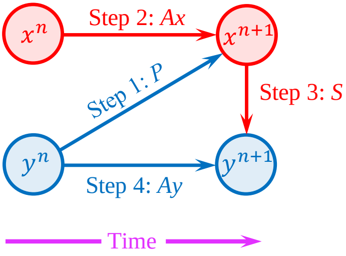


Backward Euler scheme:

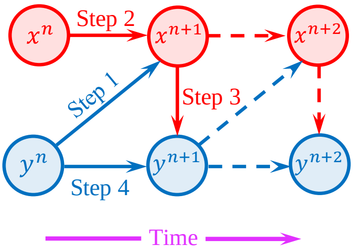
$$x^{n+1} = x^n + \dot{x}^{n+1}\Delta t,$$
$$y^{n+1} = y^n + \dot{y}^{n+1}\Delta t$$

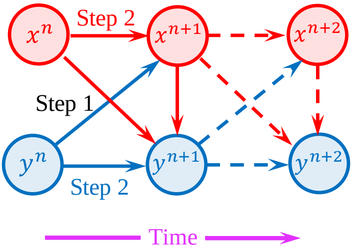
Simple partitioned scheme (weakly coupled scheme)

1. Predict: $y_*^{n+1} = y^n + \dot{y}^n \Delta t$
2. Advance x: $x^{n+1} = \dfrac{f^{n+1}\Delta t + 3x^n + y_*^{n+1}}{3 + 4\Delta t}$
3. Substitute: $x_*^{n+1} = x^{n+1}$
4. Advance y: $y^{n+1} = \dfrac{g^{n+1}\Delta t + y^n + 2x_*^{n+1}}{1 + 6\Delta t}$



Different coupling strategies





- Suppose two communicating programs ("staggered" solution procedure)
- One predictor (y)

- With two predictors (both x and y) both programs advance concurrently
- Better for parallelization

Weak & strong coupling

Weakly coupled strategies

- single (one for the fluid part and one for the structure) solution per time step
- easy to implement
- loss of conservation properties of the continuum fluid-structure system (energy increasing, unstable)
- time step is usually small
- improvements by predictors (accuracy and stability)

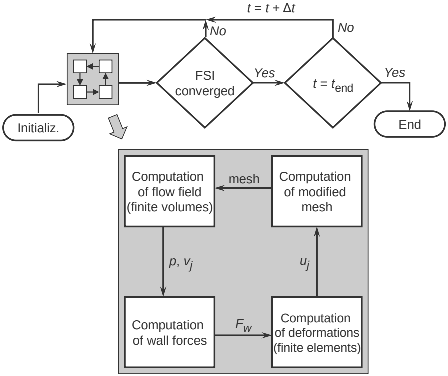
Strongly coupled strategies

- alternate fluid and structure solutions within a time step until convergence
- treat the interaction between the fluid and the structure synchronously
- maintain conservation properties
- greater computational cost per time step
- algorithmic improvements possible

Algorithmical improvements of the partitioned approach

Subiteration in detail

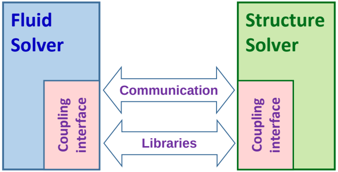
1. **Kinematic condition:**
   fluid velocity = structure velocity
   Constitutes a boundary condition for the initial-boundary-value problem of the fluid
2. **Solve the fluid:**
   the result is the flow velocity and pressure fields
3. **Dynamic condition:**
   the result is the fluid pressure (the forces) acting on the structure surface
4. **Solve the structure:**
   the result is the displacement of every point on the structure



FSI-simulation applications architectures

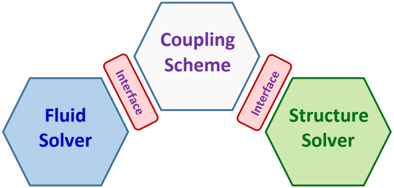
Direct communication

- coupling scheme inside the programs
- application calls the other for new boundary conditions



Client-server communication

- applications as servers
- requests from client
- concept fulfills the two requirements



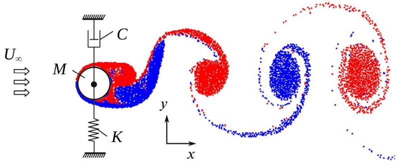
3 FSI model problem

- FSI example: circular cylinder wind resonance
- Chosen solution approaches

FSI example

Governing Equations

$$\frac{\partial \vec{U}}{\partial t} + (\vec{U} \cdot \nabla)\vec{U} = \nu \Delta \vec{U} - \frac{\nabla p}{\rho}$$
$$\nabla \cdot \vec{U} = 0$$
$$M\ddot{y} + C\dot{y} + Ky = F_y(t)$$



Dimensionless parameters

$\mathrm{St} = \dfrac{f \cdot D}{U_\infty}$ – Strouhal number

$\mathrm{Re} = \dfrac{U_\infty \cdot D}{\nu}$ – Reynolds number

$U_r = \dfrac{U_\infty}{f_n \cdot D}$ – reduced velocity

$m^* = \dfrac{4M}{\rho_f \pi D^2 L}$ – mass ratio

$\zeta = \dfrac{C}{2\sqrt{KM}}$ – damping ratio

Notation

$y(t)$, F_y – cylinder vertical displacement and lift force (m, N)

M, C, K – system mass, damping coefficient and rigidity (kg, N s/m, N/m)

D, L – cylinder diameter and length (m)

U_∞, ρ, ν – flow velocity, density and kinematic viscosity (m/s, kg/m^3, m^2/s)

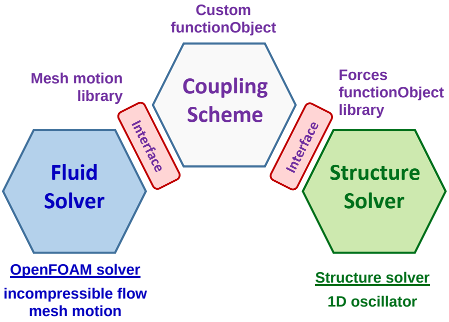
f – lift force frequency (Hz)

f_n – eigenfrequency, $f_n = \frac{1}{2\pi}\sqrt{\frac{K}{M}}$

Chosen solution approaches

- **Flow simulation:**
  - FVM — Finite volume method
  - ALE — Arbitrary Lagrangian-Eulerian
- **Structure simulation:**
  - Dynamic model with 1 degree of freedom
  - RK — Runge-Kutta 2nd order scheme
- **Coupling strategy:**
  - Partitioned approach
  - Weak coupling without predictor

Client-server architecture



4 How to implement extensions for OpenFOAM

- Different strategies to extend OpenFOAM
- fvOption facility
- functionObject facility

Different strategies to extend OpenFOAM

- **Develop new solver** Difficult for further extension
- **Develop new library:**
  - user-defined boundary condition → breaks client-server architecture
  - user-defined `fvOption` → assumes direct matrix modification
  - user-defined `functionObject` → primarily designed for postprocessing
- **Use run-time compiled input data:**
  - coded boundary condition
  - coded `fvOption`
  - coded `functionObject`

  }
  - needs special permissions for execution
  - difficult to debug

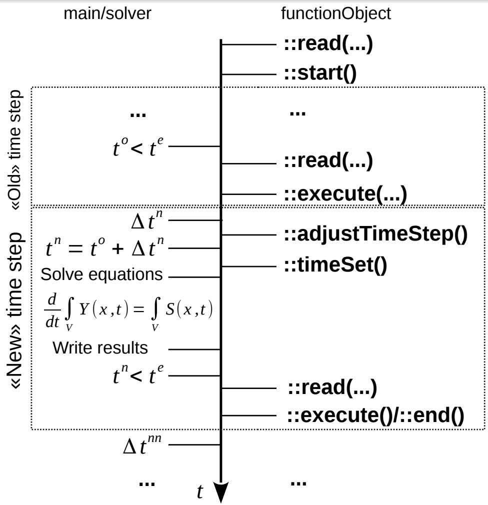
fvOption facility

Execution order diagram

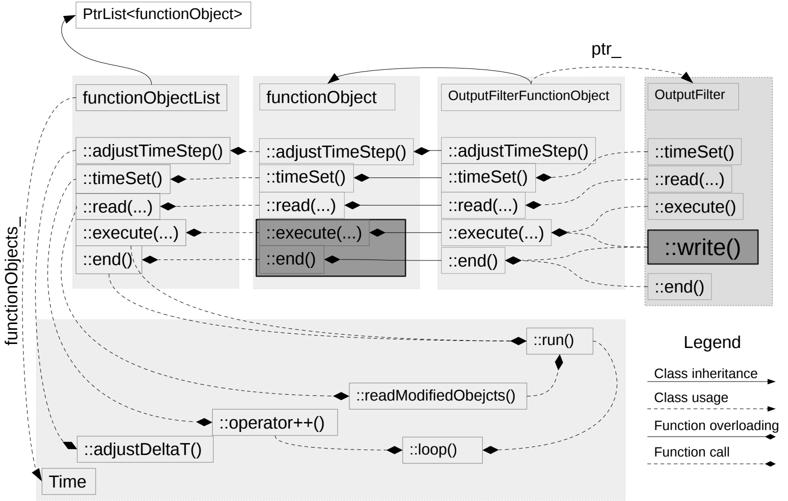
Equition to be solved: $\frac{d}{dt}\int\limits_V Y(x,\,t) = \int\limits_V S(x,\,t)$

Solver operations	fvOption operations
Formulation of discrete equation in solver $\frac{V^n\rho^nY^n - V^o\rho^oY^o}{\Delta t} + \sum\limits_f \phi_f Y_f^n = S^n$	
	Adding "sources" from `fvOption` to solver matrix A and r.h.s. b **::addSup(...)**
$AY^n = b$	
	Manipulation with matrix A from solver in `fvOption` **::constrain(...)**
$Y^n = A^{-1}b$	
	Manipulation with new solution Y^n in `fvOption` **::correct(...)**

functionObject facility — execution order diagram



functionObject facility — call order diagram



5 How to implement FSI with functionObject

- FSI example: circular cylinder wind resonance
- "Hello, World" functionObject
- Simplest coupling strategy implementation
- Restart implementation

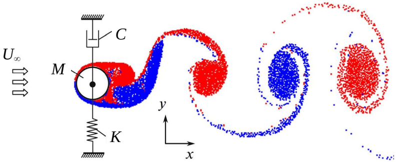
FSI example

Governing Equations

$$\frac{\partial \vec{U}}{\partial t} + (\vec{U}\cdot\nabla)\vec{U} = \nu\Delta\vec{U} - \frac{\nabla p}{\rho}$$

$$\nabla\cdot\vec{U} = 0$$

$$M\ddot{y} + C\dot{y} + Ky = F_y(t)$$



Dimensionless parameters

$\mathrm{St} = \dfrac{f\cdot D}{U_\infty}$ – Strouhal number

$\mathrm{Re} = \dfrac{U_\infty\cdot D}{\nu}$ – Reynolds number

$U_r = \dfrac{U_\infty}{f_n\cdot D}$ – reduced velocity

$m^* = \dfrac{4M}{\rho_f \pi D^2 L}$ – mass ratio

$\zeta = \dfrac{C}{2\sqrt{KM}}$ – damping ratio

Notation

$y(t),\ F_y$ – cylinder vertical displacement and lift force (m, N)

$M,\ C,\ K$ – system mass, damping coefficient and rigidity (kg, N/(m s), N/m)

$D,\ L$ – cylinder diameter and length (m)

$U_\infty,\ \rho,\ \nu$ – flow velocity, density and kinematic viscosity (m/s, kg/m^3, m^2/s)

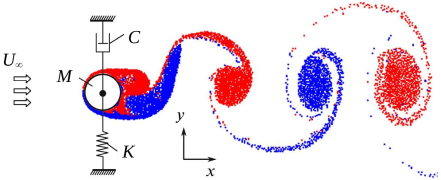
f – lift force frequency (Hz)

f_n – eigenfrequency, $f_n = \frac{1}{2\pi}\sqrt{\frac{K}{M}}$

FSI Coupling Strategy

Forces computation

- It's necessary to compute forces acting the cylinder at every time step
- How to calculate forces: use `libforces` library



Time step advancement algorithm

0. $t := t_0 + \Delta t;$
1. Move cylinder surface (mesh motion)
2. Move fluid
3. Forces computation & cylinder motion
4. Advance in time

“Hello, World” functionObject

How to create `functionObject`

- Create derived (inheriting) class
  - `helloWorld.H`
  - `helloWorld.C`
  - `helloWorldFunctionObject.H`
  - `helloWorldFunctionObject.C`
- Define overloaded functions
  - `::read(...)` — reads necessary data from dictionary for `libforces`
  - `::write()` — writes “Hello, World” and forces for cylinder
- Set `wmake` settings & Compile `libhelloWorldFunctionObject`
  - `Make/files`
  - `Make/options`
- Update `controlDict`
- Run

helloWorld.H

```
class helloWorld
:
    public forces
{
public:
    // Runtime type information
    TypeName("helloWorld");
    // Constructors
        //- Construct for given objectRegistry and dictionary.
        //  Allow the possibility to load fields from files
        helloWorld
        (
            const word& name,
            const objectRegistry&,
            const dictionary&,
            const bool loadFromFiles = false,
            const bool readFields = true
        );
    // Destructor
        virtual ~helloWorld();
    // Read the helloWorld data
        virtual void read(const dictionary&);
    // Write the helloWorld
        virtual void write();
};
```

helloWorld.C (1)

```
#include "helloWorld.H"
#include "dictionary.H"
// * * * * * * * * * Static Data Members * * * * * * * * * //
namespace Foam
{
    defineTypeNameAndDebug(helloWorld, 0);
}
// * * * * * * * * * Constructors  * * * * * * * * * //
Foam::helloWorld::helloWorld
(
    const word& name,
    const objectRegistry& obr,
    const dictionary& dict,
    const bool loadFromFiles,
    const bool readFields
)
:
    forces(name, obr, dict, loadFromFiles, readFields)
{
    this->read(dict);
}
// * * * * * * * * * Destructor  * * * * * * * * * //
Foam::helloWorld::~helloWorld()
{}
```

helloWorld.C (2)

```
// * * * Member Functions  * * * //

void Foam::helloWorld::read(const dictionary& dict)
{
    forces::read(dict);
}

void Foam::helloWorld::write()
{
    if (!active_)
    {
        return;
    }
    forces::write();
    Info << "Hello, World! Total force = " << forceEff() << endl;
}

// ******************************************************* //
```

helloWorldFunctionObject.H & .C

helloWorldFunctionObject.H

```
#include "helloWorld.H"
#include "OutputFilterFunctionObject.H"
namespace Foam
{
    typedef OutputFilterFunctionObject<helloWorld>
        helloWorldFunctionObject;
}
```

helloWorldFunctionObject.C

```
#include "helloWorldFunctionObject.H"
namespace Foam
{
    defineNamedTemplateTypeNameAndDebug(helloWorldFunctionObject, 0);
    addToRunTimeSelectionTable
    (
        functionObject,
        helloWorldFunctionObject,
        dictionary
    );
}
```

wmake settings

Make/files

```
helloWorld.C
helloWorldFunctionObject.C

LIB = $(FOAM_USER_LIBBIN)/libhelloWorldFunctionObject
```

Make/options

```
EXE_INC = \
    -I$(LIB_SRC)/fileFormats/lnInclude \
    -I$(LIB_SRC)/transportModels \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/incompressible/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/compressible/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/postProcessing/functionObjects/forces/lnInclude

LIB_LIBS = -lcompressibleTransportModels  -lturbulenceModels \
    -lincompressibleTurbulenceModels  -lcompressibleTurbulenceModels \
    -lincompressibleTransportModels  -lfluidThermophysicalModels \
    -lspecie  -lfileFormats  -lfiniteVolume  -lmeshTools  -lforces
```

Complilation & running

Compile

```
$ wmake libso
```

Add to controlDict

```
functions
{
    #include "helloWorld"
}
```

Run

```
$ pimpleDyMFoam | tee -a log
```

helloWorld part of controlDict

```
helloWorld1
{
    type          helloWorld;

    functionObjectLibs
    ( "libhelloWorldFunctionObject.so" );
    outputControl    timeStep;
    timeInterval     1;  //must be 1
    log           yes;

    //from libforces
    patches       ( cylinder );

    // Indicates incompressible
    rhoName       rhoInf;

    // Redundant for incompressible
    rhoInf        1000;

    // Reference point for torque computation
    CofR          (0 0 0);
}
```

PrintScreen

Compilation

```
wmakeLnInclude: linking include files to ./lnInclude
Making dependency list for source file helloWorldFunctionObject.C
Making dependency list for source file helloWorld.C
g++ -m64 -Dlinux64 -DWM_ARCH_OPTION=64 -DWM_DP -DWM_LABEL_SIZE=32 -Wall -Wextra -Wold-style-cast -Wnon-virtual-dtor -Wno-unused-parameter -Wno-invalid-offset
of -O3  -DNoRepository -ftemplate-depth-100 -I/unicluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-3.0.0/src/fileFormats/lnInclude -I/unicluster/bl460Cluster/o
pt/fvm/OpenFOAM/OpenFOAM-3.0.0/src/transportModels -I/unicluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-3.0.0/src/transportModels/compressible/lnInclude -I/u
nicluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-3.0.0/src/TurbulenceModels/turbulenceModels/lnInclude -I/unicluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-3
.0.0/src/TurbulenceModels/incompressible/lnInclude -I/unicluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-3.0.0/src/TurbulenceModels/compressible/lnInclude -I/
unicluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-3.0.0/src/thermophysicalModels/basic/lnInclude -I/unicluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-3.0.0/s
rc/finiteVolume/lnInclude -I/unicluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-3.0.0/src/meshTools/lnInclude -I/unicluster/bl460Cluster/opt/fvm/OpenFOAM/Open
FOAM-3.0.0/src/postProcessing/functionObjects/forces/lnInclude -IlnInclude -I. -I/unicluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-3.0.0/src/OpenFOAM/lnIncl
ude -I/unicluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-3.0.0/src/OSspecific/POSIX/lnInclude   -fPIC -c helloWorld.C -o Make/linux64GccDPInt32Opt/helloWorld
```

• • •

```
           -lcompressibleTransportModels -lturbulenceModels -lincompressibleTurbulenceModels -lcompressibleTurbulenceModels -lincompressibleTransportModels
-lfluidThermophysicalModels -lspecie -lfileFormats -lfiniteVolume -lmeshTools -lforces  -o /unicluster/home/matvey.kraposhin/OpenFOAM/matvey.kraposhin-3.0.0/
platforms/linux64GccDPInt32Opt/lib/libhelloWorldFunctionObject.so
'/unicluster/home/matvey.kraposhin/OpenFOAM/matvey.kraposhin-3.0.0/platforms/linux64GccDPInt32Opt/lib/libhelloWorldFunctionObject.so' is up to date.
```

Running

```
GAMGPCG:  Solving for p, Initial residual = 6.508932e-05, Final residual = 9.7587e-08, No Iterations 9
time step continuity errors : sum local = 1.71463e-14, global = 1.433886e-15, cumulative = -2.848206e-11
ExecutionTime = 17.31 s  ClockTime = 17 s

helloWorld helloWorld1 output:
    sum of forces:
        pressure : (0.2572604 0.0001039098 5.908528e-21)
        viscous  : (0.237799 3.783711e-06 -2.461399e-21)
        porous   : (0 0 0)
    sum of moments:
        pressure : (-2.078197e-07 0.0005145209 -8.395356e-14)
        viscous  : (-7.567422e-09 0.000475598 2.797701e-10)
        porous   : (0 0 0)

Hello, World! Total force = (0.4950594 0.0001076935 3.447129e-21)
```

Simplest coupling strategy implementation

How to create basicFsi `functionObject`

- Copy **helloWorld** `functionObject` and rename
  - `basicFsi.H`, `basicFsi.C`
  - `basicFsiFunctionObject.H`, `basicFsiFunctionObject.C`
- Add additional `#include-s`
- Modify functions
  - `::basicFsi(...)` — constructor
  - `::read(...)` — reads necessary data from dictionary for `libforces` and dynamic properties of the structure
  - `::write()` — simulates cylinder-spring dynamics
- Define function
  - `::setDisplacements(...)` — sets displacement at fluid-structure interface in the fluid domain
- Compile `libbasicFsiFunctionObject`
- Update `controlDict` & Run

Runge — Kutta 2nd order method

Cylinder dynamics equation

$$M\ddot{y} + C\dot{y} + Ky = F_y \quad \Leftrightarrow \quad \begin{cases} \dot{y} = V_y, \\ \dot{V}_y = \dfrac{F_y - CV_y - Ky}{M}. \end{cases}$$

Runge — Kutta 2nd order explicit method

0. For $t = t_n$ values $y^n = y(t_n)$, $V_y^n = V_y(t_n)$ are known. Hydrodynamic force F_y assumed to be constant during time step.
1. For $t_* = t_n + \frac{\Delta t}{2}$:

$$y^* = y^n + V_y^n \frac{\Delta t}{2}, \quad V_y^* = V_y^n + \frac{F_y - CV_y^n - Ky^n}{M} \frac{\Delta t}{2}.$$

2. For $t_{n+1} = t_n + \Delta t$:

$$y^{n+1} = y^n + V_y^* \Delta t, \quad V_y^{n+1} = V_y^n + \frac{F_y - CV_y^* - Ky^*}{M} \Delta t.$$

Additional `#include-s`

Additional `#include-s`

For `basicFsi.H`:

```
#include "volFieldsFwd.H"
#include "Tuple2.H"
#include "OFstream.H"
```

For `basicFsi.C`:

```
#include "volFields.H"
#include "Time.H"
#include "IFstream.H"
```

Additions to basicFsi.H

```
class basicFsi
:
    public forces
{
protected:
        scalar M_;        // cylinder mass
        scalar C_;        // damping coefficient
        scalar K_;        // rigidity coefficient
        scalar R_;        // ratio of cyl. length to domain depth
        scalar Ymax_;     // maximum amplitude of displacement
        Pair<scalar> Y_;         // current state of system (y,Vy)
        Pair<scalar> Yold_;      // old state of system (y, Vy)
        autoPtr<OFstream> out_; // pointer to output stream
public:
    //- Runtime type information
    TypeName("basicFsi");

    // Member Functions
        //- Distributes displacements between slave processes
        // and sets cellDisplacement field Y component on patch
        void setDisplacements(volVectorField& yDispl);
};
```

New constructor in basicFsi.C

```
Foam::basicFsi::basicFsi
(
    const word& name,
    const objectRegistry& obr,
    const dictionary& dict,
    const bool loadFromFiles,
    const bool readFields
)
:
    forces(name, obr, dict, loadFromFiles, readFields),
        M_(0.0), C_(0.0), K_(0.0), R_(0.0),
        Ymax_(0.0), Y_ (0.0, 0.0), Yold_(0.0, 0.0), out_(NULL)
{
    this->read(dict);
    if (Pstream::master())
    {
        out_.reset
        (
            new OFstream( dict.lookup("results") )
        );
        out_() << "Time;Y;Vy;Fy" << endl;
    }
}
```

read & setDisplacement functions in basicFsi.C

```
void Foam::basicFsi::read(const dictionary& dict)
{
    forces::read(dict);
    dict.lookup("M") >> M_;
    dict.lookup("C") >> C_;
    dict.lookup("K") >> K_;
    dict.lookup("R") >> R_;
    dict.lookup("Ymax") >> Ymax_;
}

void Foam::basicFsi::setDisplacements(volVectorField& yDispl)
{
    if (Pstream::parRun())
        Pstream::scatter<scalar>(Y_.first());
    vector YPatch (0.0, Y_.first(), 0.0);
    forAllConstIter(labelHashSet, patchSet_, iter)
    {
        label patchId = iter.key();
            forAll(yDispl.boundaryField()[patchId], faceI)
                yDispl.boundaryField()[patchId][faceI] = YPatch;
    }
}
```

write function in basicFsi.C

```
void Foam::basicFsi::write()
{
    if (!active_) return;

    forces::write();
    volVectorField& yDispl =
        const_cast<volVectorField&>
       ( obr_.lookupObject<volVectorField>("cellDisplacement") );

        if (Pstream::master())
    {
        scalar dt = yDispl.mesh().time().deltaT().value();
        scalar ct = yDispl.mesh().time().value();
        vector force = forceEff();
        scalar yForce = force.y();

        Pair<scalar> Ymid;  //For Runge-Kutta 2-nd order method
        ...
        Y_.first() = ...;    Y_.second()= ...;    Yold_ = Y_;

        if (log_) Info << "yForce = ..." << endl;
        out_() << ct << ";" << Y_.first() << ... << endl;
    }
    setDisplacements(yDispl);
}
```

Complilation & running

Compile

```
$ wmake libso
```

Add to controlDict

```
functions
{
    #include "basicFsi"
}
```

Run

```
$ pimpleDyMFoam | tee -a log
```

basicFsi part of controlDict

```
basicFsi1
{
    type          basicFsi;

    functionObjectLibs
    ( "libbasicFsiFunctionObject.so" );

    ... // The same as in "helloWorld"

    //FSI
    M             7.144575;
    K             639.032;
    C             0.94597;
    R             282;
    results       "yD.csv";
    Ymax          1.0; //Almost unbounded
}
```

Restart implementation

How to create weaklyCoupledFsi `functionObject`

- Copy **basicFsi** `functionObject` and rename
  - `weaklyCoupledFsi.H`, `weaklyCoupledFsi.C`
  - `weaklyCoupledFsiFunctionObject.H`, `weaklyCoupledFsiFunctionObject.C`
- Modify functions
  - `::weaklyCoupledFsi(...)` — constructor
  - `::read(...)` — reads data from dictionary for `libforces`, dynamic properties of the structure and restores previous state
  - `::write()` — simulates cylinder-spring dynamics and writes current state
- Compile `libweaklyCoupledFsiFunctionObject`
- Update `controlDict`
- Run:
  - run in serial mode
  - run in parallel mode

Modifications in `weaklyCoupledFsi.H`

```
class weaklyCoupledFsi
:
    public forces
{
protected:
        ...
        //- true if after restart data should be appended to log
        //  false if log should be overwritten
        bool append_;
        ...
public:
    ...
    //- Runtime type information
    TypeName("weaklyCoupledFsi");
    ...
};
```

Modified constructor in `weaklyCoupledFsi.C`

```
Foam::weaklyCoupledFsi::weaklyCoupledFsi(...)
: forces(...), ..., append_(false)
{
    this->read(dict);
    if (Pstream::master())
    {
        List<word> oldFileLines(0);
        if (append_)
        {
            IFstream outOld( dict.lookup("results") );
            while(!outOld.eof() && outOld.opened())
            {
                word str(word::null);
                outOld.getLine(str);
                if (!str.empty())
                    oldFileLines.append(str);
            }
        }
        out_.reset( new OFstream( dict.lookup("results") ) );
        if (append_ && oldFileLines.size())
            forAll(oldFileLines, iLine)
                out_() << oldFileLines[iLine] << endl;
        else
            out_() << "Time;Y;Vy;Fy" << endl;
    }
}
```

read(...) function in weaklyCoupledFsi.C (1)

```
void Foam::weaklyCoupledFsi::read(const dictionary& dict)
{
    forces::read(dict);
    dict.lookup("M") >> M_;
    dict.lookup("C") >> C_;
    dict.lookup("K") >> K_;
    dict.lookup("R") >> R_;
    dict.lookup("Ymax") >> Ymax_;
    dict.lookup("append") >> append_;

    Info << "Reading old state" << endl;

        autoPtr<IOdictionary> weaklyCoupledFsiDictPtr;
    //try to read weaklyCoupledFsi object properties
    {
        volVectorField& yDispl =
            const_cast<volVectorField&>
            (
                obr_.lookupObject<volVectorField>("cellDisplacement")
            );

<to be continued!>
```

read(...) function in weaklyCoupledFsi.C (2)

```
...
        //read weaklyCoupledFsiDict header
        IOobject weaklyCoupledFsiHeader
        (
            "weaklyCoupledFsiDict",
            yDispl.mesh().time().timeName(),
            "uniform",
            yDispl.mesh(),
            IOobject::MUST_READ,
            IOobject::NO_WRITE,
            false
        );

        if (weaklyCoupledFsiHeader.headerOk())
        {
            weaklyCoupledFsiDictPtr.reset
            (
                new IOdictionary( weaklyCoupledFsiHeader )
            );
            weaklyCoupledFsiDictPtr().lookup("YOld") >> Y_;
            Yold_ = Y_;
        }
        setDisplacements(yDispl);
    }
}
```

Addition to `write` function in `weaklyCoupledFsi.C`

```
void Foam::weaklyCoupledFsi::write()
{
    ...
    if (Pstream::master())
    {
        ...
        //write data to file if time is equal to output time
        if (yDispl.mesh().time().outputTime())
        {
            IOdictionary weaklyCoupledFsiDict
            (
                IOobject
                ( "weaklyCoupledFsiDict",
                  yDispl.mesh().time().timeName(), "uniform",
                  yDispl.mesh(), IOobject::NO_READ, IOobject::
                      NO_WRITE, false)
            );
            weaklyCoupledFsiDict.set<Pair<scalar> > ( "YOld", Yold_ );
            weaklyCoupledFsiDict.regIOobject::write();
        }
    }
    setDisplacements(yDispl);
}
```

Complilation & running

Compile

```
$ wmake libso
```

Modification of controlDict

```
...
startFrom        latestTime;
...
functions
{
   #include "weaklyCoupledFsi"
}
```

basicFsi part of controlDict

```
weaklyCoupledFsi1
{
   type         weaklyCoupledFsi;

   functionObjectLibs
   ( "libweaklyCoupledFsiFunctionObject.so" );

   ... // The same as in "basicFsiWorld"

   //FSI
   ... // The same as in "basicFsiWorld"
   append       true;
}
```

Run

- **in sequential mode:**

  `$ pimpleDyMFoam | tee -a log`

- **in parallel mode:**

  `$ mpirun -np 6 pimpleDyMFoam -parallel | tee -a log`

6 Numerical example

- Validation example for laminar flow
- Turbulent flow example

Validation example for laminar flow ($\mathrm{Re} = 150$)

Dimensionless parameters

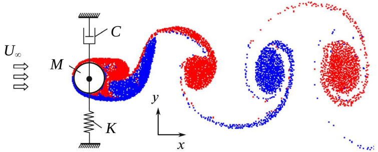
$$\mathrm{Re} = 150, \quad U_r = 5,$$
$$m^* = 2, \quad \zeta = 0.007$$

Geom. & physical parameters

$$\rho_f = 1000\ \mathrm{kg/m^3}, \quad D = 0.0635\ \mathrm{m},$$
$$U_\infty = 0.4779\ \mathrm{m/s}, \quad L = 1.128\ \mathrm{m}$$

Derived parameters

$$\nu = 0.000202311\ \mathrm{m^2/s}, \quad f_n = 1.5052\ \mathrm{Hz},$$
$$M = 7.144575\ \mathrm{kg}, \quad K = 639.032\ \mathrm{N/m},$$
$$C = 0.94597\ \mathrm{N\,s/m}$$



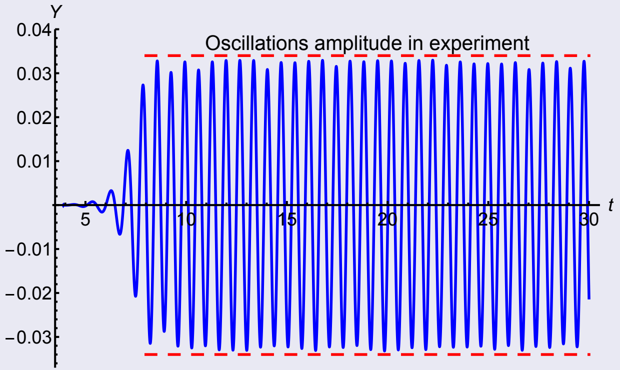
Direct numerical simulation (using `laminar` turbulence model)

Folder with this case: [validation-laminar-cont]

Results: vorticity & velocity ($\mathrm{Re} = 150$)

On youtube.com: http://youtube.com/watch?v=s3IM-g6tPK8

Results: cylinder displacement $(\mathrm{Re} = 150)$



Experiment:

$$\frac{Y_{Ampl}}{D} \approx 0.55$$

Absolute value:

$$Y_{Ampl} \approx 0.035 \text{ m}$$

Carmo B.S., Sherwin S.J., Bearman P.W., Willden R.H.J. Flow-induced vibration of a circular cylinder subjected to wake interference at low Reynolds number // *Journal of Fluids and Structures*. ????????

Example for turbulent flow ($\mathrm{Re} = 30\,000$)

Dimensionless parameters

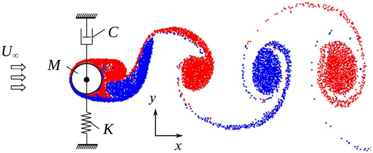
$$\mathrm{Re} = 30\,000, \quad U_r = 6.2,$$
$$\frac{M}{\rho D^2 L} = \frac{\pi}{4} m^* = 5.02, \quad \zeta = 0.02$$

Geom. & physical parameters

$$\rho_f = 1000\ \mathrm{kg/m^3}, \quad D = 0.0635\ \mathrm{m},$$
$$U_\infty = 0.4779\ \mathrm{m/s}, \quad L = 1.128\ \mathrm{m}$$

Derived parameters

$$\nu = 10^{-6}\ \mathrm{m^2/s}, \quad f_n = 1.2\ \mathrm{Hz},$$
$$M = 22.832\ \mathrm{kg}, \quad K = 1297.97\ \mathrm{N/m},$$
$$C = 6.89\ \mathrm{N\,s/m}$$



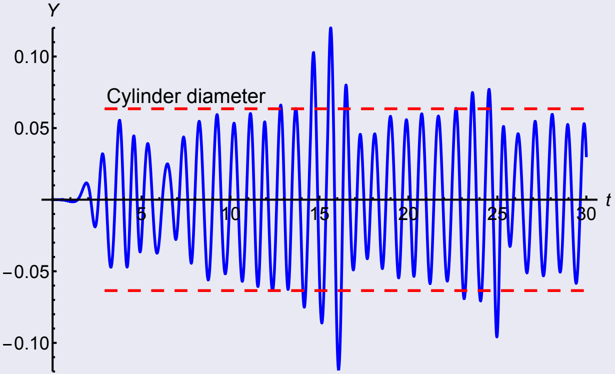
Turbulence simulation (using LES-approach with `dynamicKEqn` model)

Folder with this case: main-les-long

Results: vorticity & velocity ($\mathrm{Re} = 30\,000$)

On youtube.com: http://youtube.com/watch?v=tosM8sNfkho

Results: cylinder displacement ($\mathrm{Re} = 30\,000$)



Experiment:

$$\frac{Y_{Ampl}}{D} \approx 1.02$$

Absolute value:

$$Y_{Ampl} \approx 0.065 \text{ m}$$

Blevins R.D., Coughran C.S. Experimental Investigation of Vortex-Induced Vibration in One and Two Dimensions With Variable Mass, Damping, and Reynolds Number // *Journal of Fluids Engineering*, 2009. Vol. 131, No. 10. P. 101202 (7 pages). DOI:10.1115/1.3222904

Thank you for your attention!

Questions?