# Implementation of Simple FSI Model with `functionObject`

**Matvey Kraposhin, Ilia Marchevsky**

ISP RAS

Institute for System Programming of RAS
Bauman Moscow State Technical University

Exeter
July 24th, 2017

# 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
   - 3 DoFs 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-OF4.1` for `OpenFOAM 4.1` 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=IcR8HszacQE

# 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

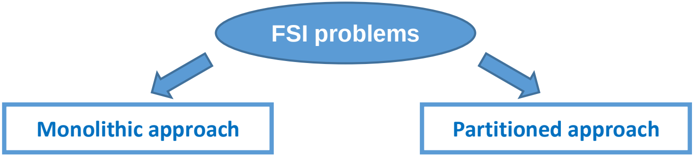
## Body-fitted mesh

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

## LS-STAG immersed boundary method

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

# Different approaches for solving FSI problems

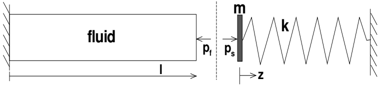



## Monolithic approach

- 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

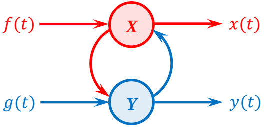
- **Advantages:**
  - customization
  - independent modeling
  - software reuse
  - modularity
- **Disadvantages:**
  - requires careful formulation and implementation to avoid serious degradation in stability and accuracy
  - parallel implementations are error-prone

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:

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

## 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:

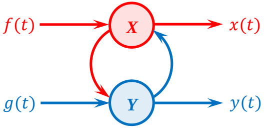
$$\begin{pmatrix} 3 + 4\Delta t & -\Delta t \\ -2\Delta t & 1 + 6\Delta t \end{pmatrix} \begin{pmatrix} x^{n+1} \\ y^{n+1} \end{pmatrix} = \begin{pmatrix} f^{n+1}\Delta t + 3x^n \\ g^{n+1}\Delta t + y^n \end{pmatrix}$$

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:

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

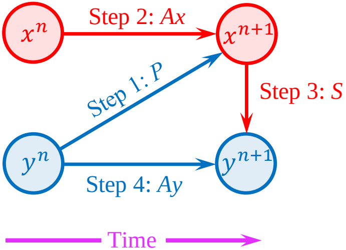



Backward Euler scheme:

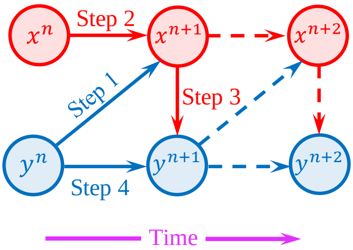
$$x^{n+1} = x^n + \dot{x}^{n+1}\Delta t,$$
$$y^{n+1} = y^n + \dot{y}^{n+1}\Delta t$$

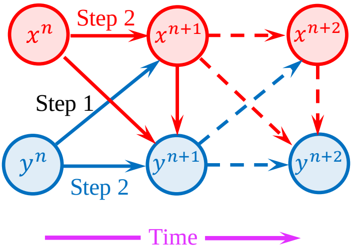
## Simple partitioned scheme (weakly coupled scheme)

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:**

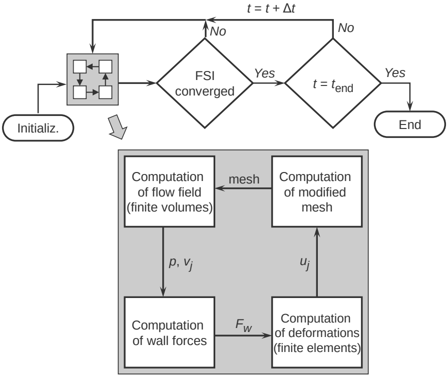
   the result is the flow velocity and pressure fields

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



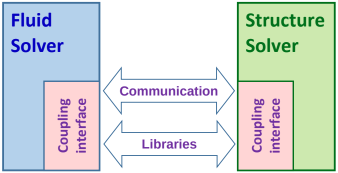
# FSI-simulation applications architectures

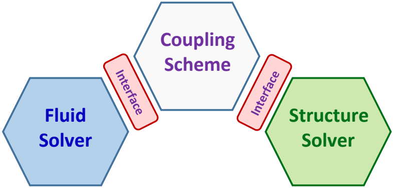
## Direct communication

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

## Client-server communication

- applications as servers
- requests from client



## 3 FSI model problem

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

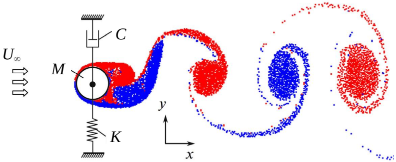
# FSI example

## 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)

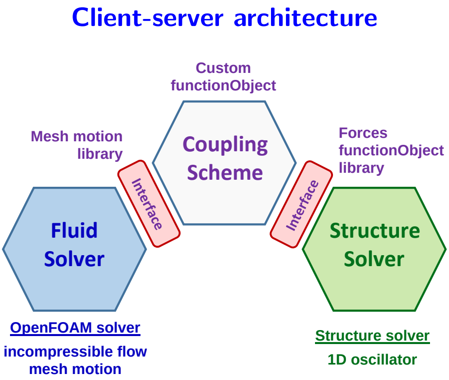
$U_\infty$, $\rho$, $\nu$ – flow velocity, density and kinematic viscosity (m/s, kg/m$^3$, m$^2$/s)

$f$ – lift force frequency (Hz)

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

# 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 2[nd] order scheme
- **Coupling strategy:**
  - Partitioned approach
  - Weak coupling without predictor



## 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

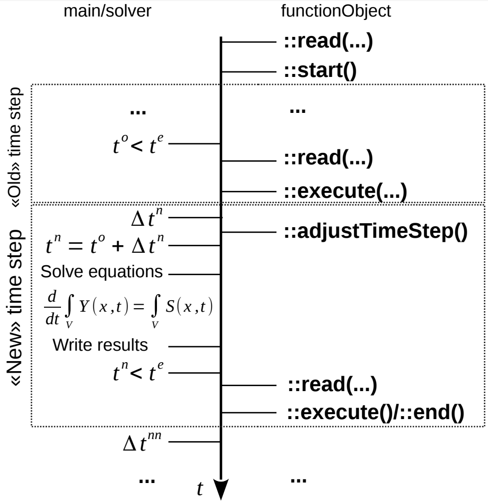
# fvOption facility

## Execution order diagram

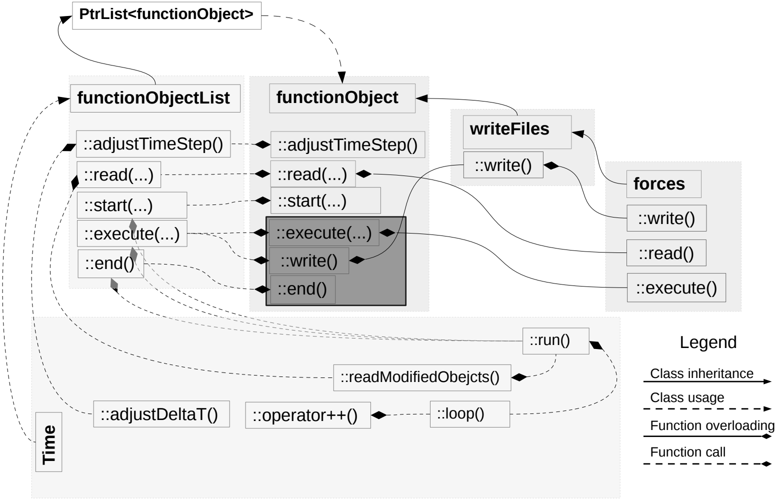
Equition to be solved: $\frac{d}{dt}\int\limits_V Y(x,\,t) = \int\limits_V S(x,\,t)$

| 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
- 3 DoFs implementation

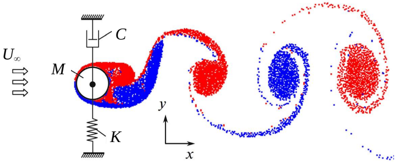
# FSI example

## 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)

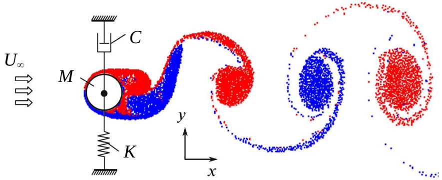
$f$ – lift force frequency (Hz)

$f_n$ – 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`
- Define overloaded functions
  - `::read(...)` — reads necessary data from dictionary for `libforces`
  - `::execute()` — returns true (defined for compatibility)
  - `::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
{
protected:
    helloWorld(const helloWorld&);
    void operator=(const helloWorld&);
public:
    TypeName("helloWorld");
        helloWorld
        (
            const word& name,
            const Time& runTime,
            const dictionary& dict
        );
        helloWorld
        (
            const word& name,
            const objectRegistry& obr,
            const dictionary& dict
        );
        virtual ~helloWorld();
    // Member Functions
        virtual bool read(const dictionary&);
        virtual bool execute();
        virtual bool write(); // Write the helloWorld (write forces
            output to console)
};
```

# helloWorld.C (1)

```
namespace Foam{namespace functionObjects{
    defineTypeNameAndDebug(helloWorld, 0);
    addToRunTimeSelectionTable(functionObject, helloWorld, dictionary)
        ;
}}
// * * * * * * * * * * * * * * Constructors * * * * * * * * * * * * *
    * //
Foam::functionObjects::helloWorld::helloWorld
(const word& name, const Time& runTime, const dictionary& dict)
: forces (name,runTime,dict)
{
    this->read(dict);
}
Foam::functionObjects::helloWorld::helloWorld
(const word& name,const objectRegistry& obr,const dictionary& dict)
: forces(name,obr,dict)
{
    this->read(dict);
}
// * * * * * * * * * * * * * * Destructor * * * * * * * * * * * * * *
    //
Foam::functionObjects::helloWorld::~helloWorld()
{}
```

# helloWorld.C (2)

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

bool Foam::functionObjects::helloWorld::read(const dictionary& dict)
{
    return forces::read(dict);
}


bool Foam::functionObjects::helloWorld::write()
{
    if (!forces::write())
    {
        return false;
    }

    Info << "Hello, World! Total force = " << forceEff() << endl;

    return true;
}

bool Foam::functionObjects::helloWorld::execute()
{
    return true;
}
```

# wmake settings

## Make/files

```
helloWorld.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)/functionObjects/forces/lnInclude

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

# Complilation & running

### Compile

```
$ wmake libso
```

### Add to controlDict

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

### Run

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

### Content of helloWorld file

```
helloWorld1
{
    type          helloWorld;

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

    //from libforces
    patches       ( cylinder );

    // Indicates incompressible
    rho           rhoInf;

    // Redundant for incompressible
    rhoInf        1000;

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

# PrintScreen

## Compilation

```
Making dependency list for source file helloWorld.C
g++ -std=c++0x -m64 -Dlinux64 -DWM_ARCH_OPTION=64 -DWM_DP -DWM_LABEL_SIZE=32 -Wall -Wextra -Wold-style-cast -Wr
 -Wno-unused-parameter -Wno-invalid-offsetof -O3  -DNoRepository -ftemplate-depth-100 -I/unicluster/bl460Cluste
OAM/OpenFOAM-4.1/src/fileFormats/lnInclude -I/unicluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-4.1/src/transpo
cluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-4.1/src/transportModels/compressible/lnInclude -I/unicluster/bl4
vm/OpenFOAM/OpenFOAM-4.1/src/TurbulenceModels/turbulenceModels/lnInclude -I/unicluster/bl460Cluster/opt/fvm/Ope
4.1/src/TurbulenceModels/incompressible/lnInclude -I/unicluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-4.1/src/
s/compressible/lnInclude -I/unicluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-4.1/src/thermophysicalModels/bas
unicluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-4.1/src/finiteVolume/lnInclude -I/unicluster/bl460Cluster/opt
penFOAM-4.1/src/meshTools/lnInclude -I/unicluster/bl460Cluster/opt/fvm/OpenFOAM/OpenFOAM-4.1/src/functionObject

            .  .  .  .  .  .  .  .  .

            -lcompressibleTransportModels -lturbulenceModels -lincompressibleTurbulenceModels -lcompressibleTu
lincompressibleTransportModels -lfluidThermophysicalModels -lspecie -lfileFormats -lfiniteVolume -lmeshTools -
cluster/home/matvey.kraposhin/OpenFOAM/matvey.kraposhin-4.1/platforms/linux64GccDPInt32Opt/lib/libhelloWorldFu
'/unicluster/home/matvey.kraposhin/OpenFOAM/matvey.kraposhin-4.1/platforms/linux64GccDPInt32Opt/lib/libhelloWo
t.so' is up to date.
```

## Running

```
GAMGPCG:  Solving for p, Initial residual = 0.006152924, Final residual = 3.419924e-05, No Iterations 6
GAMGPCG:  Solving for p, Initial residual = 0.002562999, Final residual = 9.874949e-08, No Iterations 19
time step continuity errors : sum local = 1.248614e-14, global = 1.11202e-15, cumulative = -2.790008e-11
ExecutionTime = 8.3 s  ClockTime = 9 s

helloWorld helloWorld1 write:
    sum of forces:
        pressure : (0.6413926 0.001837396 -4.237932e-21)
        viscous  : (0.5799882 2.571528e-06 4.905686e-22)
        porous   : (0 0 0)
    sum of moments:
        pressure : (-3.674792e-06 0.001282785 1.325992e-12)
        viscous  : (-5.143055e-09 0.001159976 1.511074e-09)
        porous   : (0 0 0)

Hello, World! Total force = (1.221381 0.001839968 -3.747364e-21)
```

# Simplest coupling strategy implementation

## How to create basicFsi `functionObject`

- Copy **helloWorld** `functionObject` and rename
  - `basicFsi.H`, `basicFsi.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
  - `::createFsiOutFile(...)` — create file for output of FSI simulation
- 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)
        //create file for FSI simulation output
        void createFsiOutFile(const dictionary& dict);
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::functionObjects::basicFsi::basicFsi
(const word& name, const Time& runTime, const dictionary& dict)
:forces(name,runTime,dict),
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)
{
    this->read(dict);
    this->createFsiOutFile(dict);
}

void Foam::functionObjects::basicFsi::createFsiOutFile
(const dictionary& dict)
{
    if (Pstream::master())
    {
        files().resize(3);
        files().set(2,new OFstream(dict.lookup("results")));
        file(2) << "Time;Y;Vy;Fy" << endl;
    }
}
```

# read & setDisplacement functions in basicFsi.C

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

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

```
bool Foam::basicFsi::write()
{
    if (!forces::write())
        return false;
    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_;

        Log << "yForce = ..." << endl;
        file(2) << ct << ";" << Y_.first() << ... << endl;
    }
    setDisplacements(yDispl);
    return true;
}
```

# Complilation & running

## Compile

```
$ wmake libso
```

## Add to controlDict

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

## Run

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

## Content of basicFsi file

```
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`
- 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`

```
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);
        }
    }
    this->createFsiOutFile(dict);
    if (append_ && oldFileLines.size())
    {
        for(label i=1; i<oldFileLines.size(); i++)
            file(2) << oldFileLines[i] << endl;
    }
}
```

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

```
bool Foam::weaklyCoupledFsi::read(const dictionary& dict)
{
    if (!forces::read(dict))
        return false;
    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);
    }
    return true;
}
```

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

```
bool 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);
    return true;
}
```

# 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 "basicFsi"

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

## Run

- **in sequential mode:**

  `$ pimpleDyMFoam | tee -a log`

- **in parallel mode:**

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

# 3 DoFs implementation

## 3 DoFs can be interpreted as 3 distinct springs

- scalar C_ → vector C_
- scalar K_ → vector K_
- scalar Ymax_ → vector Ymax_
- Pair⟨scalar⟩ Y_ → Pair⟨vector⟩ Y_
- Pair⟨scalar⟩ Yold_ → Pair⟨vector⟩ Yold_
- To specify springs direction, coordinate system transformation is applied:
  autoPtr⟨coordinateSystem⟩ coordSys_;
- Next procedures needs modifications:
  - ::read(...) — to read coordinate system information
  - ::setDisplacements(...) — to apply coordinate transformation before setting displacements
  - ::write() — to solve for motion equation of each spring independently

# ::read(...) and ::setDisplacements(...) modifications

## ::read(...) modifications

```
...
    if (coordSys_.empty())
        coordSys_ = coordinateSystem::New (obr_, dict);
...
```

## ::setDisplacements(...) modifications

```
    ...
    if (Pstream::parRun())
        Pstream::scatter<vector>(Y_.first());

    //displacements are relative to initial position
    vector YPatch (coordSys_().globalVector(Y_.first()));

    forAllConstIter(labelHashSet, patchSet_, iter)
    {
    ...
```

# ::write modifications

```
...
vector force = forceEff();
vector yForce = coordSys_().localVector(force); //convert
    force to local coord system

//Runge-Kutta 2-nd order method
Pair<vector> Ymid;

forAll(Ymid.first(), iCmpt)
{
    Ymid.first()[iCmpt] = Yold_.first()[iCmpt] + 0.5*dt*Yold_.
        second()[iCmpt];
...
forAll(Y_.first(), iCmpt)
{
    file(2)<< ";" << Y_.first()[iCmpt];
}
...
file(2)<< endl;
...
```

## 6 Numerical example

- Validation example for laminar flow
- Turbulent flow example

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

## Dimensionless parameters

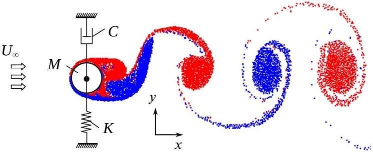
$$\mathrm{Re} = 150, \quad U_r = 5,$$
$$m^* = 2, \quad \zeta = 0.007$$

## 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

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



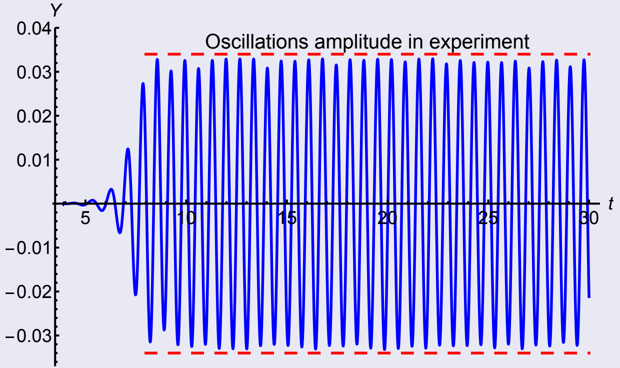

Direct numerical simulation (using `laminar` turbulence model)

Folder with this case: validation-laminar-cont

# Results: vorticity & velocity (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*. 2011. V.27, Is.4. Pp. 503–522

# Example for turbulent flow (Re $= 30\,000$)

## Dimensionless parameters

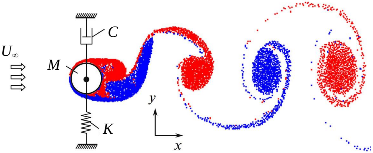
$$\mathrm{Re} = 30\,000, \quad U_r = 6.2,$$
$$\frac{M}{\rho D^2 L} = \frac{\pi}{4} m^* = 5.02, \quad \zeta = 0.02$$

## Geom. & physical parameters

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

## Derived parameters

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



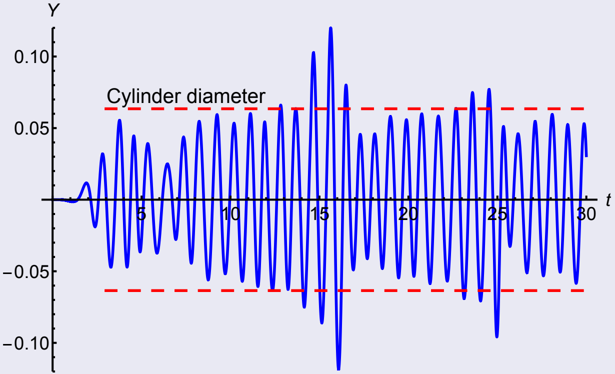

Turbulence simulation (using LES-approach with `dynamicKEqn` model)

Folder with this case: main-les-long

# Results: vorticity & velocity (Re $= 30\,000$)

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

# Results: cylinder displacement (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?