Implementation of acoustic analogy

CAA in OpenFOAM

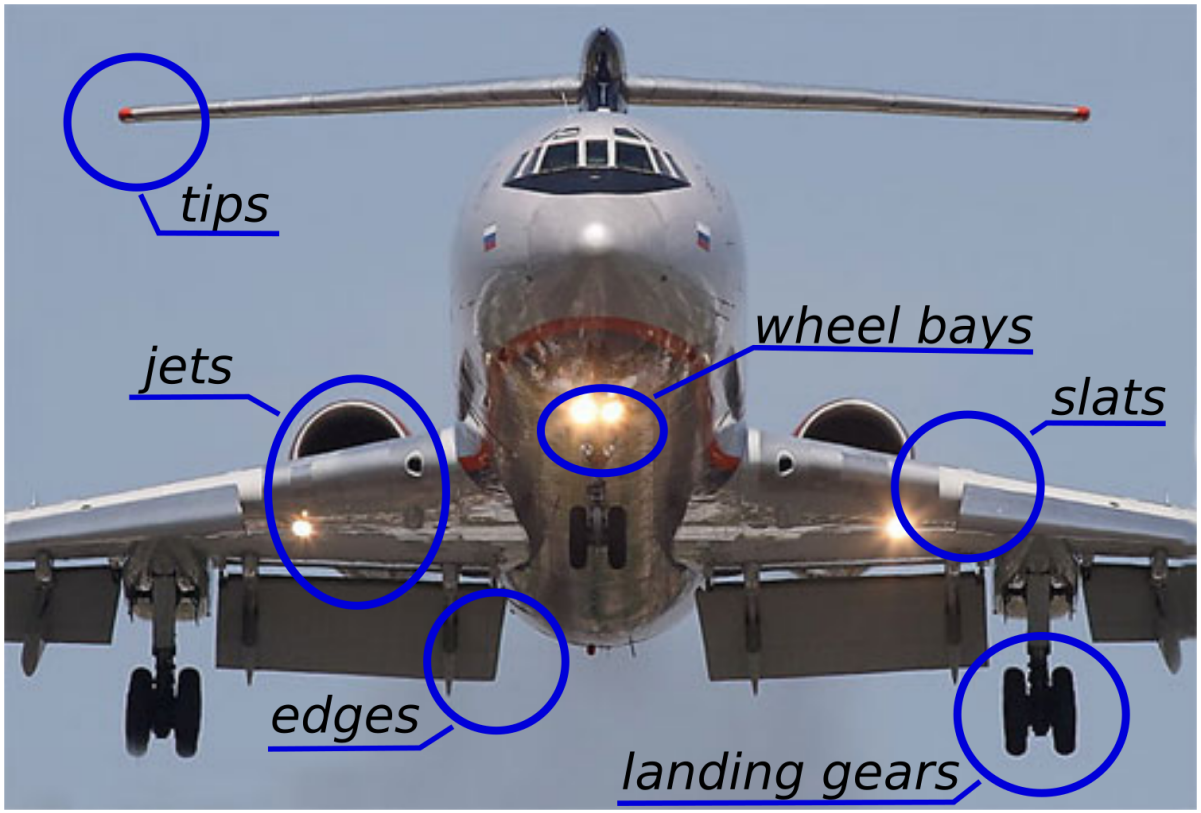
Ilya Evdokimov, M. Kraposhin, S. Strizhak

11th OpenFOAM Workshop, Guimarães, Portugal

26th – 30th of June, 2016

Content

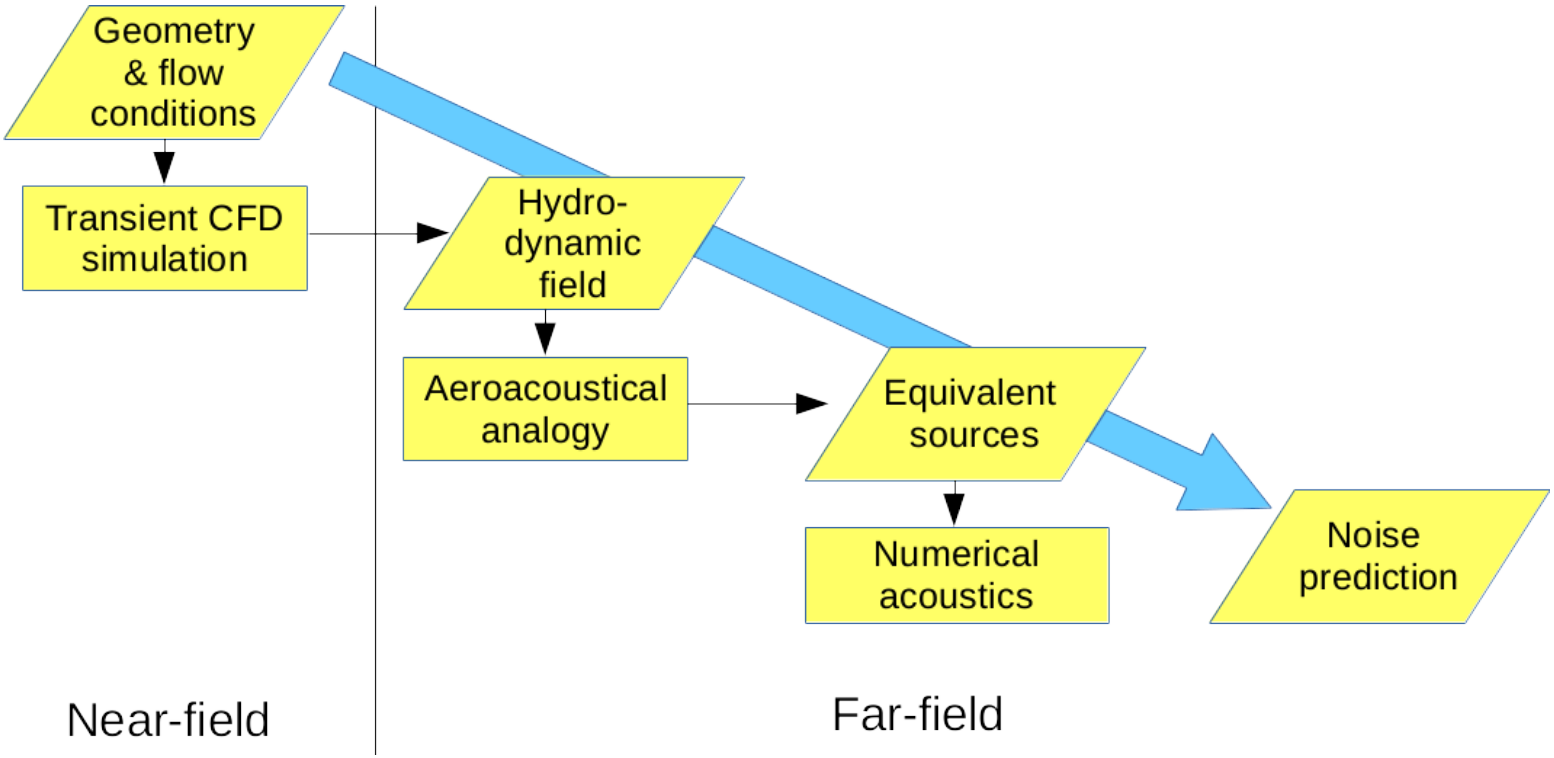
Airplane Noise Sources



Roland Ewert, Aircraft noise simulation at DLR, CEAA, Svetlogorsk, Russia. 2012.

Place of acoustic analogies in CFD

ISP RAS



Important issues

ISP RAS

- Boundary Conditions
- Numerical Schemes
- Hybrid URANS/LES, LES models
- Number of cells per wave
- Sound pressure level correction in Z direction in 2D simulations
- FFTW library (Fourier Transform)

Governing equations

ISP RAS

Source:

Curle, N. The influence of solid boundaries upon aerodynamic sound. Proc. Royal Soc. 1955. 231A. Pp. 505–514.

Just for the record, we are starting from Lighthill's equation:

$$\frac{\partial^2 \rho'}{\partial t^2} - c_0^2 \frac{\partial^2 \rho'}{\partial x_i^2} = \frac{\partial^2 T_{ij}}{\partial x_i \partial x_i} \tag{1}$$

where

$$T_{ij} = \rho v_i v_j + p_{ij} - c_0^2 \rho \delta_{ij} \tag{2}$$

$\rho =$ density, $p_{ij} =$ compressive stress tensor, $c_0 =$ speed of sound, $v_i =$ component velocity in direction $x_i (i = 1, 2, 3)$,

Governing equations

Curle's fundamental result:

$$\rho - \rho_0 = \frac{1}{4\pi c_0^2} \frac{\partial^2}{\partial x_i \partial x_j} \int_V \frac{T_{ij}(\mathbf{y}, t - r/c_0)}{r} d\mathbf{y} + \frac{1}{4\pi c_0^2} \frac{\partial^2}{\partial x_i} \int_S \frac{P_i(\mathbf{y}, t - r/c_0)}{r} d\mathbf{y} \tag{3}$$

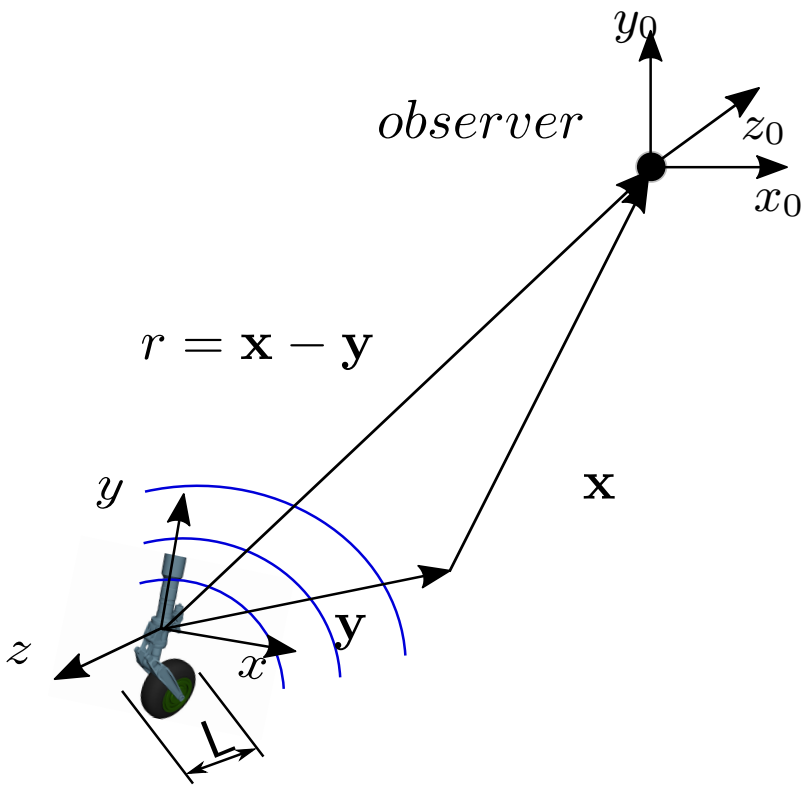
where $P_i = -l_j p_{ij}$, l_j – direction cosines of the outward normal of the fluid, $(l_1, l_2, l_3) = \mathbf{n}$.

(!)

We need to do some transformations and assumptions to get computational form of the equation 3

Assumptions & Simplifications

- Volumetric sources are negligible (no Lighthill tensor)
- Flow is Isentropic in the region of interest ($c_0^2 = \gamma p/\rho$)
- Compressibility effects are negligible
- The body is compact ($L << \lambda$), the retarded time neglected



In the end we simplify equation 3 and obtain

$$\rho - \rho_0 = \frac{1}{4\pi c_0^3} \frac{x_i}{x^2} \frac{\partial}{\partial t} F_i(t) \tag{4}$$

or assuming that flow is isentropic and $x \approx r$

$$p' = \frac{1}{4\pi c_0} \frac{x_i}{r^2} \frac{\partial}{\partial t} F_i(t) \tag{5}$$

$F_i(t) = \int_S P_i(\mathbf{y}, t) dS(\mathbf{y})$ is the total resultant force upon the fluid by the solid body and x_i is the direction vector.

The equation 5 represents Curle's analogy (far-field interpretation) in various literature.

Limitations

- Low-speed incompressible flow
- The surface assuming non-deforming and stationary
- The non-linear effects are excluded

About Fourier Transform

ISP RAS

Let us assume that we have function x(n). We can approximate it with the next expansion:

$$x(n) = \sum_{m=0}^{N-1} F(m) \exp\left(\frac{i2\pi mn}{N}\right) \tag{6}$$

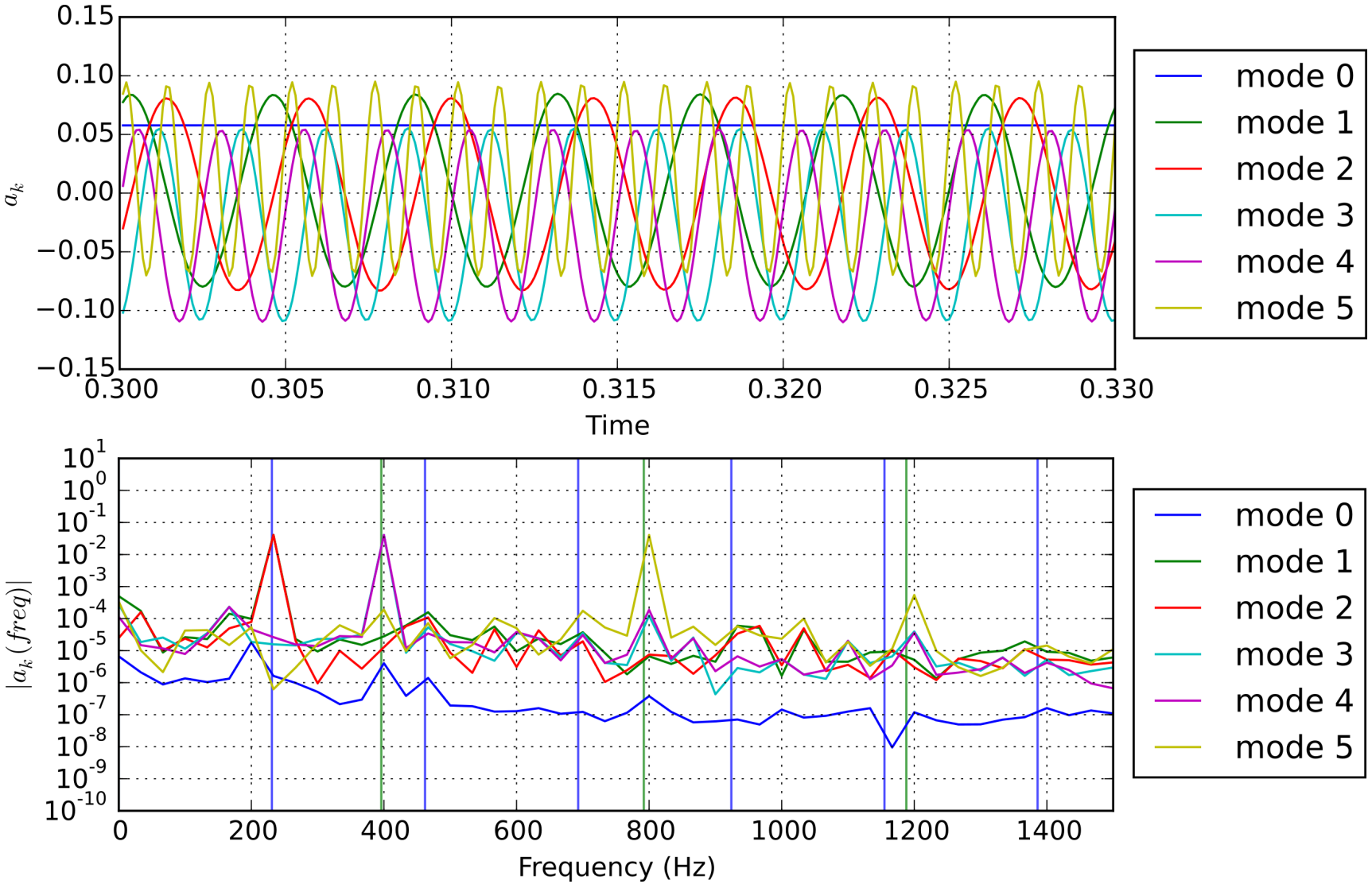
where

- $x(n)$ – function value at grid point n
- $F(m)$ – spectral coefficient for the m^{th} wave component
- $\exp$ could be replaced on $\sin$ or $\cos$ or it's sum
- The *power spectrum* – a plot of the $|F(m)|^2$ vs. frequency

The Fast Fourier Transform algorithms allows us to find $F(m)$, m

Find more about FFT with Python hands-on programming: Home page for ESCI 386 - Scientific Programming, Analysis and Visualization with Python

FFT instance: signal and amplitudes



Selecting a tool: FFTW vs noise

ISP RAS

FFTW

+ Was implemented in `libAcoustics` before `noise`
– Works fast, although there was not special research
– Is an additional library that user should properly compile

`noiseFFT`

+ Native and implemented since 2.2 (or earlier?)
+ User do not need to provide any additional stuff
+ It seems that it's not slower than FFTW

OpenFOAM FFT functions

ISP RAS

noise utility

Code to perform noise analysis of pressure data using the noiseFFT library.

noiseFFT.C

Provides main functions for noise.C:

- evaluate FFT-coefficients,
- RMS-values,
- saves graphs

noiseFFT.C suggest using of forwardTransform and reverseTransform, but we do not need more than half of this code.
Where: src/randomProcesses/noise

fft.C

Contains all Fast-Fourier Transform Mechanics.
Where: src/randomProcesses/fft/

OpenFOAM FFT functions

ISP RAS

1. Obtain FFT-coefficients

```
tmp<scalarField> tPn2
( mag
    ( fft::reverseTransform
        ( ReComplexField(p_), labelList(1, p_.size() )
)));
```

2. Save them in RAM

```
tmp<scalarField> tPn
( new scalarField
    ( scalarField::subField(tPn2(), tPn2().size()/2)
));
```

OpenFOAM FFT functions

ISP RAS

3. Process raw FFT coefficients

```
scalarField& Pn = tPn();
Pn *= 2.0/sqrt(scalar(tPn2().size()));
Pn[0] /= 2.0;
```

...in according with

$$F(m) = \frac{1}{N} \sum_{n=0}^{N-1} f(n) \exp(-i2\pi mn/N) \tag{7}$$

using `fft` we only get $\sum_{n=0}^{N-1} f(n) \exp(-i2\pi mn/N)$,
then the $1/N$ coefficient is implemented separetely as $1/\sqrt{N/2}$ in `fft.C` and then as $2/\sqrt{N/2}$ in the in `noiseFFT.C` when calculating `scalarField& Pn`.

4. Obtain frequencies

```
scalar N = p_.size();
scalarField f(N/2);
scalar deltaf = 1.0/(N*tau);
forAll(f, i)
        { f[i] = i*deltaf;}
```

... in according with

$$\nu_m = \frac{m}{\tau_s N};\ m = 0, 1, 2...N/2 \tag{8}$$

The purpose of the functionObject API

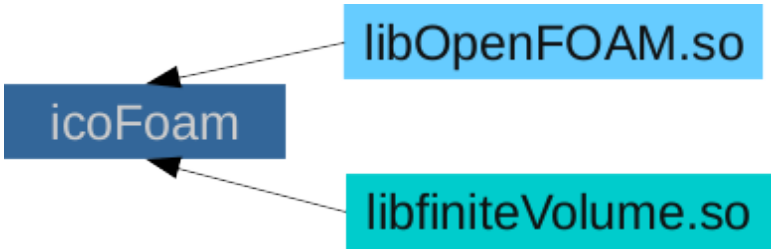
ISP RAS

- functionObject is the C++ interface for user post-processing, i.e. user subroutines implemented in the object-oriented way
- functionObject API uses dynamic library loading mechanism (thus, you must be familiar with the creation of libraries in OpenFOAM)
- functionObject is independent from the main solver application and can be reused with different application
- All data which comes from solver to functionObject is read-only by default

Dynamic libraries in OpenFOAM

ISP RAS

- Dynamic libraries are the mechanism for the execution in the main executable procedures, which are stored in the separate files



- Some libraries can be linked directly with executable, some — can be invoked during execution (functionObject)
- Dynamic libraries are widely used for run-time selection of different models (turbulence models, transport models, e.t.c.)
- There are special macro definitions to help in creation of dynamic libraries in OpenFOAM
- Dynamic libraries use OpenFOAM software pointers — autoPtr<> to create new instances by default

How to create library

ISP RAS

Example: SimpleFoamLibrary

- simpleFoamLibrary.H, simpleFoamLibrary.C — definition of library class
- newSimpleFoamLibrary.C — definition of the ::New(...) procedure, used to create and instance of the library
- Make/files: contains list of source files for compilation and name of the library

```
simpleFoamLibrary.C
newSimpleFoamLibrary.C
altSimpleFoamLibrary/altSimpleFoamLibrary.C
LIB = $(FOAM_USER_LIBBIN)/libsimpleFoamLibrary
```

- Make/options: compilation options

```
EXE_INC = \
-I$(LIB_SRC)/finiteVolume/lnInclude
LIB_LIBS = \
-lfiniteVolume
```

- Run «wmake libso» to compile library

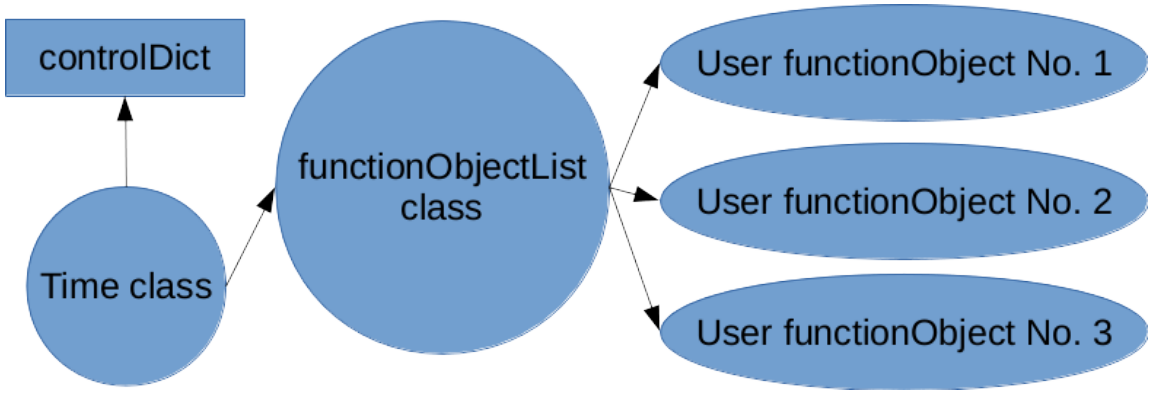
How it works

ISP RAS

The key idea – the C++ mechanism of inheritance:

1. There is a prototype of a class that provides data processing
2. This prototype contains virtual methods for data processing
3. User must provide the new class, inherited from this prototype. This class must contain implementation of virtual methods for post-processing
4. This implementation of the user class must be placed in the separate dynamic library

How it works



1. List of user-defined functionObjects is located in controlDict file
2. A list of implementations of functionObject is stored in the special array (class functionObjectList)
3. Array of different functionObjects is stored in the object of class Time. Class Time is responsible for the execution of the functionObject at the different moments of simulation

Instances

Where:

→ `$FOAM_SRC/postProcessing/functionObjects`

Some examples:

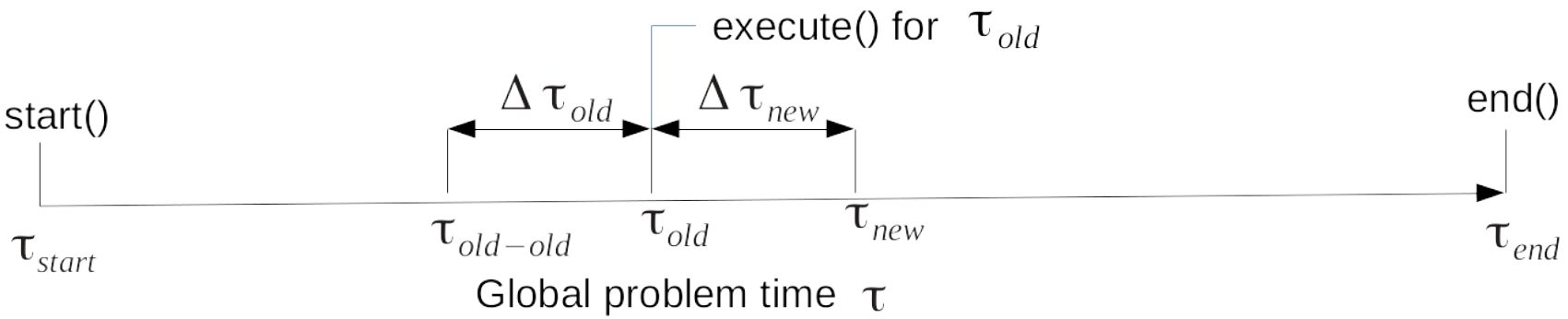
1. forces
2. forceCoeffs
3. fieldMinMax

Call to `functionObject`

`functionObjectList` contains reference to Time, i.e. - root objectRegistry
`Time::run()` - returns true if the current time is less than the final time
Different functions are called at three time points:

OpenFOAM iteration stages:

- `FunctionObjectList::end()`
- `FunctionObjectList::start()`
- `FunctionObjectList::execute()`
- `FunctionObjectList::write()`



functionObjectList class methods

ISP RAS

::start()

executes all functionObjects before the time loop

::end()

executes all functionObjects after the time loop

::execute()

executes all functionObjects after the end of each time step (before new time step)

::read()

reads «function» entry of the «controlDict» file and creates (or updates) functionObjects. Each functionObject is created using functionObject::New(Time&) static method

::write()

saves results on hard disk

functionObject class methods

ISP RAS

::start(), ::end(), ::execute()

inherited from above

::read(const dictionary&)

abstract method, read and set function object

::functionObject(const word&, const Time&, const dictionary&)

must be implemented in child classes

::New(const word&, const Time&, const dictionary&)

performs look-up in dynamic libraries (see previous slide) table. If the given type of object is found, returns pointer of the given type to the object.

::iNew sub-class

used to create functionObject from Istream (given entry of dictionary)

`functionObject` class methods

ISP RAS

These methods should be implemented

- start()
- execute(...)
- functionObject(...)
- read(...)
- write(...)

How to implement new `functionObject`

ISP RAS

Example: SimpleFoamFunctionObject

1. Create new library (directory for library, Make/files, Make/options)
2. Copy files functionObject.H and functionObject.C from source code of OpenFOAM
3. Replace functionObject class name with desired name in this source code files
4. Rename source code files according to the given class name
5. Edit files Make/files and Make/options (as it was done in the dynamic library example)
6. Place your own code to you class (next slide) and compile with `wmake libso`

How to implement new `functionObject`

ISP RAS

Example: SimpleFoamFunctionObject

1. New class must be derived from functionObject class:

```
class SimpleFoamFunctionObject : public functionObject
```

2. New class should contain declaration and implementation (your post-processing code) of the listed above four member functions:

```
SimpleFoamFunctionObject virtual bool start();
(virtual bool execute(bool);
        const word& name,
        virtual bool read(const dictionary&);
        const Time&,
        const dictionary&
);
```

3. New class should not contain functions ::New(...), sub-class ::iNew, macro-definitions declareRunTimeSelectionTable (in .H) and defineRunTimeSelectionTable (in .C)
4. Define type name of your class and put it in the run-time selection table (in .C file):

```
namespace Foam
{
        defineTypeNameAndDebug(SimpleFoamFunctionObject, 0);
        addToRunTimeSelectionTable(functionObject, SimpleFoamFunctionObject, dictionary);
}
```

Connection to the case

ISP RAS

Example: testSimpleFoamFunctionObject

1. Open file `controlDict`
2. Create (if it is not created) new sub-dictionary «functions»:

```
functions
{
}
```

3. Place named description of you functionObject:

```
HelloFunction1  //title of functionObject
{
        type SimpleFoamFunctionObject;
        //type (class name) of functionObject
        functionObjectLibs ("libSimpleFoamFunctionObject.so");
        //where to find this class
        helloAddress "World";
        //parameters of functionObject which are read by function //read(...)
}
```

OutputFilterFunctionObject

ISP RAS

- Used for libraries like forces, forceCoeffs and other
- Implements time-stepping control (user don't need to care about it)
- Templated child of the functionObject class
- Stores reference to the Time object and passes reference of the mesh object (or mesh region object) to the template object
- `::start()` allocates memory for the template object
- `::execute(bool)` allocates memory for the template object (if it was not allocated earlier) and runs execute() method of this object
- `::end()` executes end() method of the template object (same as execute(...) method)
- `::read(const dictionary&)` read dictionary and run start()
- *An option*. There are special functions `forceEff()` and `momentEff` in the OpenFOAM 3.0 that allow us to calculate forces directly without special procedures.

How to use it

ISP RAS

Example: forces in OpenFOAM source code

1. Create your own library
2. Create post-processing class which must contain at least new constructor, and member functions:

```
Foam::forces::forces
( const word& name, const objectRegistry& obr, const dictionary& dict, const bool loadFromFiles, const bool reac
void execute(); void write(); void end();
```

3. Create new type with template OutputFilterFunctionObject

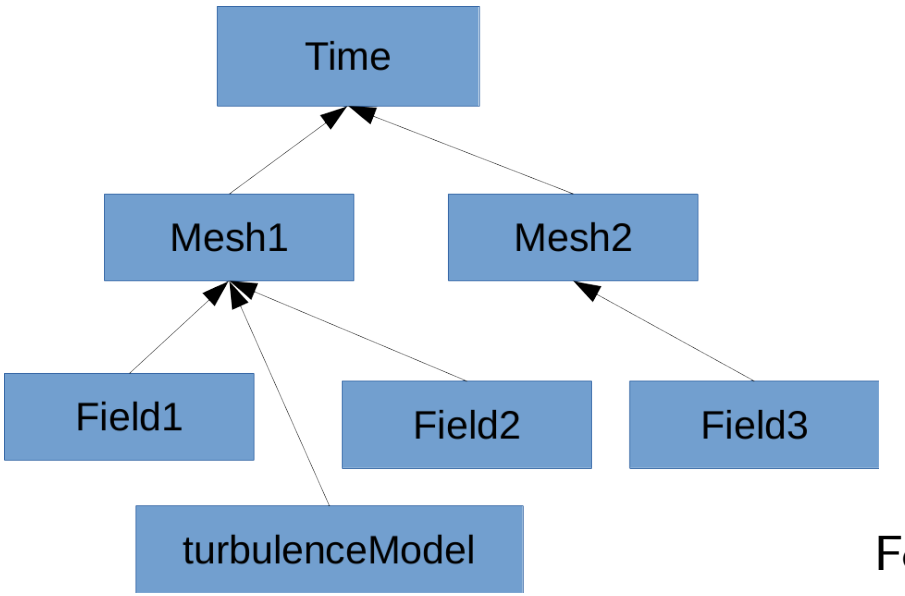
```
typedef OutputFilterFunctionObject<forces> forcesFunctionObject;
```

4. Register this new type in the library

```
defineTypeNameAndDebug(Foam::forces, 0);
defineNamedTemplateTypeNameAndDebug(forcesFunctionObject, 0);
addToRunTimeSelectionTable
(
        functionObject,
        forcesFunctionObject,
        dictionary
);
```

objectRegistry

Class `objectRegistry` — is a special hash table which is used to store named references to solver variables (time, mesh, fields and others). This table is organized in the hierarchical tree with the root object of class Time. The main branches in this tree are time and mesh. Main leaves — fields, models and others



Main access functions:

- `const T& lookupObject<T>(word name)` Returns object with name «name» and type T
- `bool foundObject<T>(word name)` Returns «true» if object with name, «name» and type T is found

For example:

```
const volScalarField& p =
mesh.lookupObject<volScalarField>(«p»
```

How Solver Data is Accessed

All objects in the user-define functionObject class can be accessed from objectRegistry reference, passed to the constructor as the second argument (see `forces` example code). Of course, if you are creating your own class, you must store this reference in the class variable (obr_ in example). By default this objectRegistry refers to the mesh (fvMesh type) object:

```
onst fvMesh& mesh = refCast<const fvMesh>(obr_); //refCast - type conversion
```

Now you can import any object from the solver

```
const Time& time = mesh.time(); //reference to problem time object
const volScalarField& p = mesh.lookupObject<volScalarField>(pName_);
//reference to pressure (field with name "p")
```

As you can see, all references are constant — you can't modify them. But you can always (if you are sure what you are doing) cancel constant specification:

```
const volScalarField& p = mesh.lookupObject<volScalarField>(pName_); \\ pName_ = "p"
volScalarField& nonConstP = const_cast<volScalarField&>(p); \\disable const for nonConstP
```

What we need to know

ISP RAS

1. How to provide user's input data
2. How to estimate integral body force acting on the fluid
3. How to calculate time derivative of this force
4. How to estimate acoustic pressure, SPL and their spectrum (fft)
5. How to save calculated data to hard disk
6. How to organize calculations in case of parallel execution

Implemented library (libAcoustics) contains next files: Curle.C CurleFunctionObject.C CurleFunctionObject.H Curle.H SoundObserver.C SoundObserver.H

Reading `controlDict`

Read of user supplied data input data implemented in the function `Curle::read(const dictionary&)`. Next data must be provided by the user:

- Probe frequency
- Names of patches to integrate over
- Start and end time for probing
- Sonic speed
- Names of pressure and density fields (or reference density for incompressible case)
- Reference length for 2D case
- Observers information: position, reference pressure, fft analysis frequency

Reading controlDict

ISP RAS

```
const dictionary& obsDict = dict.subDict("observers");
wordList obsNames = obsDict.toc();
forAll (obsNames, obsI)
{
    word oname = obsNames[obsI];
    vector opos (vector::zero);
    obsDict.subDict(oname).lookup("position") >> opos;
    scalar pref = 1.0e-5;
    obsDict.subDict(oname).lookup("pRef") >> pref;
    label fftFreq = 1024;
    obsDict.subDict(oname).lookup("fftFreq") >> fftFreq;
    observers_.append
    (
            SoundObserver
            (
                    oname,      opos,   pref,   fftFreq
            )
    ); }
```

Body force estimation

ISP RAS

To estimate body force we need to integrate pressure field (flow is turbulent and viscous stresses are neglected) over all `surfaces(Curle::correct())`:

```
const fvMesh& mesh = refCast<const fvMesh>(obr_); // take referenc
//sign '-' needed to calculate force, which exerts fluid by solid
vector F        (0.0, 0.0, 0.0);
vector dFdT (0.0, 0.0, 0.0);
scalar deltaT = mesh.time().deltaT().value();
forAll(patchNames_, iPatch)
{
    word patchName = patchNames_[iPatch];
    label patchId = mesh.boundary().findPatchID(patchName);
    scalarField pp = normalStress(patchName);
    F -= gSum (pp * mesh.Sf().boundaryField()[patchId]);
}
```

Calculating acoustic pressure

Time derivative of the force could be calculated using Euler and backward differencing schemes:

```
dFdT = (3.0*F - 4.0*FOldPtr_() + FOldOldPtr_()) / 2.0 /
deltaT;
```

Acoustic pressure is estimated for each sound observer (class Sound Observer) using expression, listed above :

```
scalar coeff1 = 1. / 4. / Foam::constant::mathematical::pi / c0_;
forAll (observers_, iObs)
{
    SoundObserver& obs = observers_[iObs];
    vector l = obs.position() - c_;
    scalar r = mag(l);
    scalar oap = l & (dFdT + c0_ * F / r) * coeff1 / r / r;
    if (dRef_ > 0.0)
    {
        oap /= dRef_;
    }
    obs.apressure(oap);
}
```

Fast-Fourier Transform

PSD(SPL) levels can be obtained only for Fourier expansion of the acoustic pressure, because acoustic pressure can be negative. To obtain Fourier expansion we use functuions of the class `fft`:

```
tmp<scalarField> tPn2
(
    mag
        ( fft::reverseTransform (
            ReComplexField(p_),
                labelList(1, p_.size()) )
        )
);
```

After performing Fourier transform it returns list of frequencies. Amplitudes are calculated separately. After that SPL spectrum is calculated.

Saving data to disk

Data is saved on the disk using OFstream OpenFOAM class:

```
// time history output
CurleFilePtr_() << (cTime - timeStart_) << " ";
forAll(observers_, iObserver)
{
    const SoundObserver& obs = observers_[iObserver];
    CurleFilePtr_() << obs.apressure() << " ";
}
```

There are two files for each observer: one for acoustic pressure time history and another for Fourier analysis results

Parallel execution

In case of parallel execution, we need to account for the next:

1. Specified patches can be located on different processors — we need to gather data:

```
gSum (scalar), reduce(scalar, sumOp<scalar>());
```

2. `fft` and write operations must be performed only on one processor (for example, master) – Pstream class:
   `Pstream::master()` – true if the process is master
   `Pstream::parRun()` – true if program is launched in parallel

and
`wmake wclean` to compile library!

Test case description

Airframe and Automobile Noise

ISP RAS

Task: Calculate Aeolian tones generated by cylinder at various Reynolds numbers

Sources:

- NASA NTRS, Second Computational Aeroacoustics Workshop on Benchmark Problems
- Brentner et all, COMPUTATION OF SOUND GENERATED BY FLOW OVER A CIRCULAR CYLINDER: AN ACOUSTIC ANALOGY APPROACH
- Revell et all, Experimental Study of Aerodynamic Noise vs Drag Relationships for Circular Cylinders

Test case description

Aeolian tones of cylinder

ISP RAS

Experimental setup data

- $Re = 90000$
- $D = 0.019$ m
- $M = 0.1, 0.2, 0.3, 0.35, 0.4, 0.45, 0.5$
- $Tu = 0.2 - 0.4\%$

For our Curle analogy we can choose $M = 0.1 - 0.2$. Having in the mind Brentner work, let's choose $M = 0.2$

Test case description

Aeolian tones of cylinder

ISP RAS

- `pisoFoam` – incompressible flows solver
- $k - \omega SST$ – turbulence model
- `blockMesh` – less than 100 000 cells
- Boundary conditions

Patch	Velocity, U	Pressure, p
inlet	fixedValue	zeroGradient
outlet	zeroGradient	totalPressure

Test case description

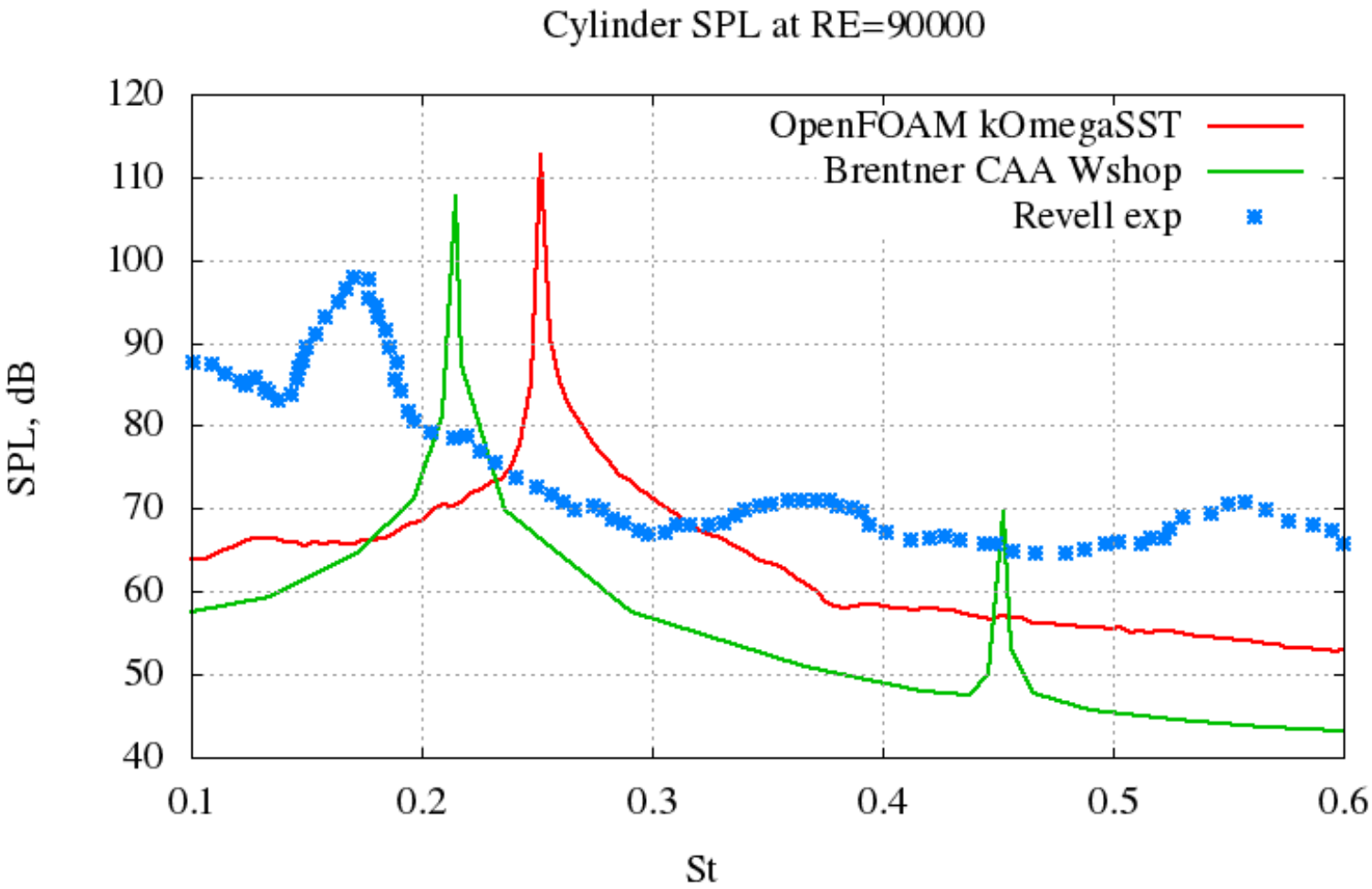
Aeolian tones of cylinder

Simulation steps:

- OFF turbulence, get laminar model results
- ON turbulence and set `libAcoustics` dictionary
- `gnuPlot` if you wish

Test case description

Aeolian tones of cylinder



Thank you for your attention. Questions?