Implementation of library for acoustic sound pressure and spanwise correction

Aya Aihara

Division of Electricity,
Uppsala University,
Sweden

2018-11-28

Outline

Outline

Introduction

Computational aeroacoustics for noise emission

Problem

High cost of simulating the whole computational domain to obtain the sound pressure generated from long-span body

Aim

Implementation of library to calculate the total sound pressure using the flow field data of the body section in the computational domain

Outline

Sound propagation

- Acoustic wave equation
  - Describes the propagation of acoustic pressure in a medium
- Curle's acoustic analogy
  - Considers the influence of solid boundaries upon the flow field
  - Solution of the Curle's equation

$$\rho(\boldsymbol{x}, t) - \rho_0 = \frac{1}{4\pi c_0^2} \frac{\partial^2}{\partial x_i \partial x_j} \int_V \frac{T_{ij}}{r} dV(\boldsymbol{y}) - \frac{1}{4\pi c_0^2} \frac{\partial}{\partial x_i} \int_S \frac{n_j}{r} (p\delta_{ij} - \tau_{ij}) dS(\boldsymbol{y})$$

Sound propagation

- Modified Curle's equation

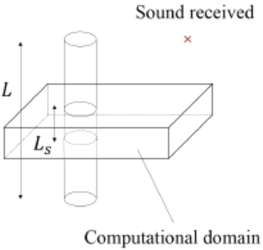
$$p(\boldsymbol{x},t)-p_0=\frac{1}{4\pi}\int_V\left(\frac{l_il_j}{c_0^2r}\ddot{T}_{ij}+\frac{3l_il_j-\delta_{ij}}{c_0r^2}\dot{T}_{ij}+\frac{3l_il_j-\delta_{ij}}{r^3}T_{ij}\right)dV(\boldsymbol{y})$$
$$+\frac{1}{4\pi}\int_S l_in_j\left(\frac{\dot{p}\delta_{ij}-\dot{\tau}_{ij}}{c_0r}+\frac{p\delta_{ij}-\tau_{ij}}{r^2}\right)dS(\boldsymbol{y})$$

This equation is implemented in the library.

Spanwise correction

Correction method proposed by Kato *et al.* [1]

- p_{corr} : Sound pressure generated from the entire body (L)
- p : Sound pressure generated from the section of the computational domain (L_s)



Corrected pressure $p_{corr} = r_{corr}(f)\,p$ where

$$r_{corr}(f) = \begin{cases} L/L_s & (L \leq L_c(f)) \quad (1) \\ \sqrt{LL_c}/L_s & (L_s \leq L_c(f) \leq L) \quad (2) \\ \sqrt{L/L_s} & (L_c(f) \leq L_s) \quad (3) \end{cases}$$

$L_c(f)$: Spanwise coherence length

[1] C. Kato *et al.*. Numerical prediction of aerodynamic noise radiated from low mach number turbulent wake

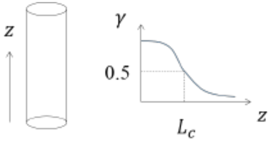
Spanwise correction

- L_c is the length where the coherence is 0.5
- Coherence function $\gamma(f, z)$ between the surface pressure at $z = x$, y

$$\gamma(f, z) = \frac{|W_{xy}(f)|^2}{W_{xx}(f) \cdot W_{yy}(f)}$$

where

$W_{xy}(f)$: Cross power spectral density between x and y
$W_{xx}(f), W_{yy}(f)$: Power spectral density



Outline

AcousticAnalogy library

- Developed by M. Heinrich and uploaded in GitHub
  https://github.com/Kiiree/curleAnalogy
- Calculates the sound pressure p based on Curle's analogy
- Top-level directory structure

```
acousticFunctionObject
 ├─ Curle
 │   ├─ Curle.H
 │   ├─ Curle.C
 │   ├─ CurleFunctionObject.H
 │   └─ CurleFunctionObject.C
 ├─ Make
 │   ├─ files
 │   └─ options
 ├─ soundObserver.H
 └─ soundObserver.C
```

Modification of AcousticAnalogy library

In order to obtain the corrected sound pressure $p_{corr} = r_{corr}\, p$, the code additionally needs to

1. Sample pressure on the body surface
2. Determine the coherence $\gamma(f, z)$ from the sampled pressure
3. Calculate the spectrum of the sound pressure p
4. Find r_{corr} and calculate the spectrum of the corrected sound pressure p_{corr}

Preparation

- Go to `$WM_PROJECT_USER_DIR/src` and place the directory of the AcousticAnalogy library
- Create a new directory
  ```
  mkdir CurleCorr
  ```
- Copy the files from the AcousticAnalogy library
  ```
  cp -r acousticFunctionObject/* CurleCorr/
  ```
- Go to the directory
  ```
  cd CurleCorr
  ```
- Rename the files
  ```
  mv Curle/Curle.H Curle/CurleCorr.H
  mv Curle/Curle.C Curle/CurleCorr.C
  mv Curle/CurleFunctionObject.H Curle/CurleCorrFunctionObject.H
  mv Curle/CurleFunctionObject.C Curle/CurleCorrFunctionObject.C
  ```
- Replace the word `Curle` to `CurleCorr`
  ```
  sed -i s/Curle/CurleCorr/g Curle/*
  ```

Make directory

- In files

```
Curle/CurleCorr.C
Curle/CurleCorrFunctionObject.C
soundObserver.C

LIB = \$(FOAM_USER_LIBBIN)/libAcousticAnalogyCorr
```

- In options

```
EXE_INC = \
    -I\$(LIB_SRC)/finiteVolume/lnInclude \
    -I\$(LIB_SRC)/meshTools/lnInclude \
    -I\$(LIB_SRC)/fileFormats/lnInclude \
    -I\$(LIB_SRC)/sampling/lnInclude \
    -I\$(LIB_SRC)/randomProcesses/lnInclude

LIB_LIBS = \
    -lspecie \
    -lfiniteVolume \
    -lmeshTools \
    -lfileFormats \
    -lsampling \
    -lrandomProcesses
```

CurleCorr.H

- Add header files

```
#include "probes.H"
#include "complexFields.H"
```

- Top of the CurleCorr class should be

```
class CurleCorr
:
    public functionObjectFile,
    public probes
```

- Add protected member data

```
const fvMesh& mesh_;
bool loadFromFiles_;
 ⋮
```

CurleCorr.H

- Add public member functions

```
virtual void storeSampledPressure();
virtual void calculateSpectrum();
virtual void calculateCoherence();
virtual void calculateCorrection();
virtual complexField calcFFT(const scalarList&);
```

Function	Description
`storeSampledPressure`	Sample surface pressure
`calculateSpectrum`	Calculate the spectrum of p
`calculateCoherence`	Determine the coherence $\gamma(f, z)$
`calculateCorrection`	Find r_{corr} and calculate the spectrum of p_{corr}
`calcFFT`	Compute the Fourier transform

CurleCorr.C

- Add header files

```
#include "fft.H"
```

- Add in the initialise function

```
countFFT_ += Nstart_;
  ⋮
```

- Add lines for initialization in contructor

```
probes(name, obr, dict, loadFromFiles),
mesh_(refCast<const fvMesh>(obr)),
  ⋮
```

- Add in the read function as

```
L_ = readScalar(dict.lookup("L"));
Ls_ = readScalar(dict.lookup("Ls"));
  ⋮
```

CurleCorr.C

- Add in the `write` function

```
probes::write();
storeSampledPressure();
if(countStep_==freqSample_*(pow(2,countFFT_)+pow(2,Nstart_-1)))
{
    calculateSpectrum();
    calculateCoherence();
    calculateCorrection();
    countFFT_ += 1;
}
countStep_ += 1;
```

CurleCorr.C

- Add the definition of `storeSampledPressure` function

```
void Foam::CurleCorr::storeSampledPressure()
{
    const volScalarField& p =
                 obr_.lookupObject<volScalarField>(pName_);
    const scalarField p_sample = probes::sample( p );
    forAll(p_sample,i)
    {
        pList_[i].append(p_sample[i]);
    }
}
```

CurleCorr.C

- Add the definition of `calculateSpectrum` function

```
void Foam::CurleCorr::calculateSpectrum()
{
  ⋮
}
```

- Add the definition of `calculateCoherence` function

```
void Foam::CurleCorr::calculateCoherence()
{
  ⋮
}
```

- Add the definition of `calculateCorrection` function

```
void Foam::CurleCorr::calculateCorrection()
{
  ⋮
}
```

CurleCorr.C

- Add the definition of `calcFFT` function as

```
Foam::complexField Foam::CurleCorr::calcFFT
(
  const scalarList& tfield
)
{
  complexField tfftField = ReComplexField(tfield);
  labelList fftList ( 1, tfield.size() );
  complexField Cofft=fft::reverseTransform(tfftField,fftList);
  Cofft *= 2.0/pow(tfield.size(),0.5);
  Cofft[0] /= 2.0;
  Cofft.last() /= 2.0;
  return Cofft;
}
```

soundObserver.H/soundObserver.C

- In `soundObserver.H` add a private member data

  ```
  List<scalar> pPrimeAll_;
  ```

  and two public member functions

  ```
  const List<scalar>& pPrimeAll() const
  {
      return pPrimeAll_;
  }
  void storepPrime(scalar pPrime);
  ```

- In `soundObserver.C` add a line for initialization in constructor

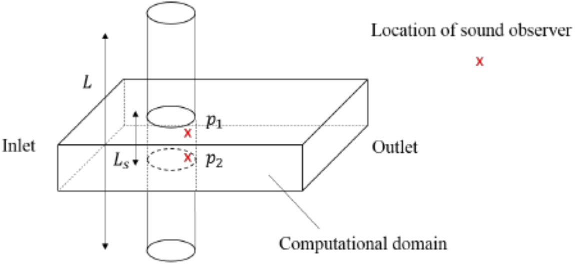
  ```
  pPrimeAll_(0)
  ```

  and the definition of `storepPrime` function

  ```
  void Foam::SoundObserver::storepPrime(scalar pPrime)
  {
      pPrimeAll_.append(pPrime);
  }
  ```

Outline

Test case description



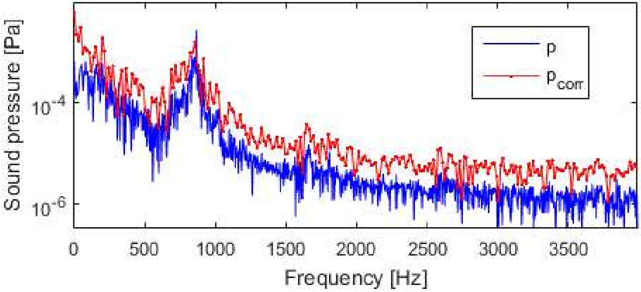
- Inlet velocity : 70.2 m/s
- Density of air : 1.20 kg/m^3
- Diameter : 19.0 mm
- L : 0.50 m
- L_s : 0.05 m
- Surface pressure sampled at p_1 and p_2
- Observer at 2.4 m from the center of the cylinder

Input entries

- In functions in controlDict

```
functionObjectLibs ( "libAcousticAnalogyCorr.so" );
type CurleCorr;
patchName ( cylinder );
probeLocations
(
    (0.0095057 0 -0.02)
    (0.0095057 0 0.02)
)
observers
{
    micro1 { position (0 -2.4335 0); }
}
L               0.5;
Ls              0.05;
freqSample      1024;
Nstart          3;
Naverage        4;
Naverage        4;
   ⋮
```

Result



p : Sound pressure generated from the section of the computational domain

p_{corr} : Sound pressure generated from the entire cylinder

Thank you for your attention