Implementation of library for acoustic sound pressure and spanwise correction

Aya Aihara

Division of Electricity, Uppsala University, Sweden

2018-11-28
Outline

1 Introduction

2 Theory

3 Implementation of library

4 Test case
Outline

1 Introduction

2 Theory

3 Implementation of library

4 Test case
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

1. Introduction
2. Theory
3. Implementation of library
4. Test case
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

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

- Modified Curle’s equation

\[ p(x, t) - p_0 = \frac{1}{4\pi} \int_V \left( \frac{l_{ij}}{c_0^2 r} \dddot{T}_{ij} + \frac{3l_{ij} - \delta_{ij}}{c_0 r^2} \ddot{T}_{ij} + \frac{3l_{ij} - \delta_{ij}}{r^3} \dot{T}_{ij} \right) dV(y) \]

\[ + \frac{1}{4\pi} \int_S l_i n_j \left( \frac{\dot{p}\delta_{ij} - \tau_{ij}}{c_0 r} + \frac{p\delta_{ij} - \tau_{ij}}{r^2} \right) dS(y) \]

This equation is implemented in the library.
Spanwise correction

Correction method proposed by Kato et al. [1]

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

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

\[
 r_{\text{corr}}(f) = \begin{cases} 
  \frac{L}{L_s} & (L \leq L_c(f)) \\
  \sqrt{\frac{L L_c}{L_s}} & (L_s \leq L_c(f) \leq L) \\
  \sqrt{\frac{L}{L_s}} & (L_c(f) \leq L_s)
\end{cases}
\]

$L_c(f)$: 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

1. Introduction
2. Theory
3. Implementation of library
4. Test case
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
  ```bash
  mkdir CurleCorr
  ```
- Copy the files from the AcousticAnalogy library
  ```bash
  cp -r acousticFunctionObject/* CurleCorr/
  ```
- Go to the directory
  ```bash
  cd CurleCorr
  ```
- Rename the files
  ```bash
  ```
- Replace the word Curle to CurleCorr
  ```bash
  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
  
  ```cpp
  virtual void storeSampledPressure();
  virtual void calculateSpectrum();
  virtual void calculateCoherence();
  virtual void calculateCorrection();
  virtual complexField calcFFT(const scalarList&);
  ```

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>storeSampledPressure</td>
<td>Sample surface pressure</td>
</tr>
<tr>
<td>calculateSpectrum</td>
<td>Calculate the spectrum of $p$</td>
</tr>
<tr>
<td>calculateCoherence</td>
<td>Determine the coherence $\gamma(f, z)$</td>
</tr>
<tr>
<td>calculateCorrection</td>
<td>Find $r_{corr}$ and calculate the spectrum of $p_{corr}$</td>
</tr>
<tr>
<td>calcFFT</td>
<td>Compute the Fourier transform</td>
</tr>
</tbody>
</table>
CurleCorr.C

- Add header files
  ```
  #include "fft.H"
  ```

- Add in the initialise function
  ```
  countFFT_ += Nstart_;  
  ```

- Add lines for initialization in constructor
  ```
  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

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

```cpp
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
  ```cpp
  void Foam::CurleCorr::calculateSpectrum()
  {
  ...
  }
  ```

- Add the definition of calculateCoherence function
  ```cpp
  void Foam::CurleCorr::calculateCoherence()
  {
  ...
  }
  ```

- Add the definition of calculateCorrection function
  ```cpp
  void Foam::CurleCorr::calculateCorrection()
  {
  ...
  }
  ```
Add the definition of calcFFT function as

```c++
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;
}
```
In `soundObserver.H` add a private member data

```cpp
List<scalar> pPrimeAll_;  
```
and two public member functions

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

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

```cpp
pPrimeAll_(0) 
```
and the definition of `storepPrime` function

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

1. Introduction
2. Theory
3. Implementation of library
4. Test case
Test case description

- Inlet velocity: 70.2 m/s
- Density of air: 1.20 kg/m³
- 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

```plaintext
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

\[ p : \text{Sound pressure generated from the section of the computational domain} \]
\[ p_{\text{corr}} : \text{Sound pressure generated from the entire cylinder} \]
Thank you for your attention