FSI_T1

Fluid-Structure Interaction Practice Homework

by harryzhou2000 @

School of Aerospace Engineering, Tsinghua University

Note:

if you speak Chinese, my report for coursework is recommended

if you desire to go through some analysis of specific cases. If you want to see the latex formulae properly, see readme.pdf

this document is under work, the contents are incomplete.

Introduction

Origin

This project was a coursework for *the Fundamental of Computational Mechanics* of THU in spring 2021. The course contains both CFD and CSD, and lecturers said I could complete a FSI program in place of both final assignments, as a result this program was born. (It was said FSI projects had seldom been seen in the course apart from one, but I didn't actually get an A for this second record ~~lmao~~)

Basic Purpose

FSI_T1 is basically a c++ (with some template) library providing multiple classes, which enables you to assemble from them a custom
Fluid,
Structural
or
FSI computation case. Fluid part currently supports only Euler equation, which represents adiabatic and inviscid compressible ideal gas dynamics. Solid part currently supports linear elastic mechanics, including statics and

modal-truncation method dynamics. FSI part, correspondingly, is basically meant for moderate structural displacement, typically aeroelastic cases.

The library currently does not have a interface supporting script input or interactive case setting, which means one must construct each single case inside a c++ calling of the relevant classes.

About Project Source Code and Dependencies

What did I code?

All the actual source code files are put in **root** of project directory, all as .hpp or .h files. The cpp files are meant for case building, containing various case sets, and they can be viewed as some kind of 'static scripts' for the program.

The draw-back of not having a script interface and using .cpp as 'script' is that each time you have a new case to compute a re-compiling is needed, which could be rather time consuming.

Dependencies

This project depends on Eigen, which completely is a c++ template library, with no binary file linking. For convenience, I just threw the entire Eigen source code in ./include/, so you won't have to install it. This project also needs c++ standard library and STL to work, which can be handled automatically by compilers mostly.

Numerical Methods and Algorithms

Fluid Methods

To comply with significant movement of boundaries caused by structural displacement, the method of arbitrary Euler-Lagrangian (ALE) description of fluid is adopted. The major differences between ALE and Euler descriptions is that ALE adds some items to Euler relating to the mesh speed[1].

Consider Euler equation for gas dynamics in conservative form:

$$\frac{\partial U}{\partial t}+\frac{\partial F}{\partial x}+\frac{\partial G}{\partial y}+\frac{\partial H}{\partial z}=0$$

$$U=\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho w\\ E\end{bmatrix},F=\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ \rho uw\\ u(E+p)\end{bmatrix},G=\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ \rho vw\\ v(E+p)\end{bmatrix},H=\begin{bmatrix}\rho w\\ \rho uw\\ \rho vw\\ \rho w^2+p\\ w(E+p)\end{bmatrix}$$

Thus in a ALE control volume:

$$\frac{\partial}{\partial t}\int_{\Omega(t)}UdV+\int_{\partial\Omega(t)}U(u_{\Omega x}n_x+u_{\Omega y}n_y+u_{\Omega z}n_z)d\Gamma+\int_{\partial\Omega(t)}(Fn_x+Gn_y+Hn_z)d\Gamma=0$$

Where $u_{\Omega i}$ are the speed components of the C.V.'s boundaries. The corresponding bold symbols are for vectors in xyz space.

Annotating:

$$\boldsymbol{u^*}=\boldsymbol{u}-\boldsymbol{u_\Omega}$$

Thus:

$$\frac{\partial}{\partial t}\int_{\Omega(t)}\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho w\\ E\end{bmatrix}dV+\int_{\partial\Omega(t)}\left(\begin{bmatrix}\rho u^*\\ \rho u^*u^*+p\\ \rho u^*v^*\\ \rho u^*w^*\\ u^*(E+p)\end{bmatrix}+\begin{bmatrix}\rho v^*\\ \rho u^*v^*\\ \rho v^*v^*+p\\ \rho v^*w^*\\ v^*(E+p)\end{bmatrix}+\begin{bmatrix}\rho w^*\\ \rho u^*w^*\\ \rho v^*w^*\\ \rho w^*w^*+p\\ w^*(E+p)\end{bmatrix}\right)d\Gamma$$
$$+\int_{\partial\Omega(t)}\rho\left(u_x^*n_x+u_y^*n_y+u_z^*n_z\right)\begin{bmatrix}0\\ u_{\Omega x}\\ u_{\Omega y}\\ u_{\Omega z}\\ 0\end{bmatrix}d\Omega=0$$

Thus if using $\boldsymbol{u^*}$ as speed, the flux could be approximated with a common approximate Riemann solver when the approximate field is not C^0. Other items could be handled rather simply.

Finite volume spacial discretion is adopted(FVM). This program uses Roe's approximate Riemann solver[2], which is modified with entropy fix of Harten-Yee[3] for numerical flux. This program conducts 2nd-order

reconstruction with Barth-Jesperson limiter[4].

The boundary conditions are built basically in the form of virtual cells. Slip-wall boundary and non-reflecting boundary are implemented. A pressure inlet/outlet boundary is also implemented , but its robustness is yet to be improved.

Time discretion in fluid part is a implicit Euler scheme.

The mesh considering FVM is completely composed of tetrahedra.

Structural Methods

Solid part is conducted with classic linear finite element method(FEM). Using symbols form a text book[5], a linear (small displacement) elastic dynamic problem is discretized as:

$$M\ddot{a} + C\dot{a} + Ka = F$$

Where a is the vector of nodal DOFs, defined as nodal displacement components. The matrices above are distributed in the finite elements as below:

$$K^e = \int_{\Omega e} B^{eT} D B^e dV, \quad M^e = \int_{\Omega e} N^{eT} \rho N^e dV,$$
$$F^e = \int_{\Omega e} f N^e dV + \int_{\partial \Omega e} p N^e d\Gamma$$

Where N^e is a row vector of shape functions(interpolating functions)on the nodes in the finite element, and B^e is a matrix representing linear contributions of the nodal DOFs to the strain field(which has 6 rows if using symmetric 3-D strain tensor, and DOF columns corresponding to DOFs). ρ is just density field, and f , p , are external forces of volume and surface.

In this project, currently only 4-node tetrahedra elements are implemented, which means both integrations in faces or volumes only need one interpolation point. If higher order elements are to be used, the interpolation method should be accurate enough.

A static solver can be found in the program, with LDL decomposition method or PCG method. For FSI purpose, a modal-truncation dynamic solver is implemented. Ignoring matrix C at first, then using a eigen problem:

$$K\Phi = M\Phi\Lambda$$

While demanding that:

$$\Phi^T M \Phi = I$$

Therefore:

$$\Phi^T K \Phi = \Lambda$$

Is diagonal.

The dynamic system becomes:

$$\ddot{x} + \Phi^T C \Phi \dot{x} + \Lambda x = G$$

Where:

$$G = \Phi^T F, \quad \Phi^T x = a$$

If the damping part $\Phi^T C \Phi$ is also diagonal, which is assumed in most CSD computations, the problem is decoupled for each DOF in eigenspace as:

$$\ddot{x_k} + d_k \dot{x_k} + \lambda_k x_k = g_k$$

Where d_k is a modal damper in the eigenspace, and $\sqrt{\lambda_i}$ are frequencies of each mode.

Therefore the discretized problem becomes a set of ODEs that can be separately solved. When the major concern is on vibration rather than delicate propagation of elastic waves, modes with higher frequencies contribute little to major features of the system, so instead of obtaining the whole eigentransformation, modes with n-smallest eigenvalues are solved. As a result, numerical methods solving modes with smallest eigenvalues can be applied here. This project uses the simplest one, the inverse power method.

FSI Methods

The FSI in this project is based on a simple explicit coupling method. The external force for structural model is obtained from the flow in the last coupling time step, and the flow obtains the moving mesh and interface from the structural movement in the last couping time step. If a general implicit method is to be expressed, the condition connecting both problems is the coordinated interface movement and passage of force on the interface, which defines where the overall system should converge. As this project only considers transient problem, only using a explicit method is acceptable.

Due to some problems in my meshing technique, the fluid and structural meshes don;t adapt perfectly on the interface, but their mesh densities are similar. So when passing force and displacement, this project conducts simple interpolations between the surfaces(with KNN search). It should be noted that this is a common problem in more complex meshing(such as rotating meshes or overlapping grids), so there must be some better and more robust techniques to perform the interpolation than this project's implementation.

The last problem in FSI is to move the fluid mesh correctly. There are many interpolating methods that moves the mesh well, but this project uses a much simpler but rather time-consuming way. If you view the fluid part as a solid body, and apply displacement boundary conditions on all its outer faces, when the interface moves, the fluid mesh moves correspondingly. When the interface is attached, the movements inside the mesh is also almost continuous. The only problem that arises is that 'stress' distribution in the fluid mesh could be singular around some points of the interface, and is mostly concentrated near the interface. Under the assumption of linear structural response, the elements near the interface are likely to be overlapped or distorted. A simple way of solving this is to set higher value of modulus near the interface, forcing the strain to distribute farther from the interface. Also, setting higher shearing modulus could help abating the distortion.

To set the modulus distribution automatically, a distance field could be applied. However, in this project, as the elements are actually smaller near the interface, the volumes of interface are used to set the modulus.

A Brief Guide To Using This Code

Implementation Framework

Classes

Fluid

Structural

FSI

Input File Convention

Case Building

Compiling

In theory, this project should be included as a header-only c++ library in your project, which means no actual building is needed. However, as the input/output and intermediate files are pretty tricky to produce, so main.cpp is provided for using. mainbkp.cpp provides other useful cases than in main.cpp. To compile, you simply use **g++** to compile the single main.cpp file, or use **make** like instructed below.

If you put all the functions in mainbkp.cpp into main.cpp to compile, when using mingw64 g++ in Windows, the compiler could exit abnormally declaring an error on insufficient resources concerning templates. I can't repeat this problem, please tell me in the issues why this could happen if you know.

The mainSG.cpp is for a 2D structural gird Euler equation gas dynamics solver, sharing some headers. Its case construction appears much more complex.

Make

```
make main.exe #for debug
make mainR.exe #for release
make mainSG.exe #for mainSG debug
make mainSGR.exe #for mainSG release
```

Notes

It was only after I STARTED this document when I find my English being rather amateur, even in my own domains... I even searched for a proper antonym for 'proficient' for use in the previous sentence... So please forgive me if any expression in my text seems strange or erroneous.

References

[1] P, Le, Tallec, et al. Fluid structure interaction with large structural displacements[J]. *Computer Methods in Applied Mechanics and Engineering*, 2001, 190(24-25):3039-3067.

[2] Roe P L . Approximate Riemann solvers, parameter vectors, and difference schemes[J]. *Journal of Computational Physics*, 1981, 43(2):357-372.

[3] Yee H C . Upwind and symmetric shock-capturing schemes. 1987.

[4] Barth T J , Jespersen D C . The design and application of upwind schemes on unstructured meshes[J]. AIAA Aerospace Sciences Meeting, 1989, 0366(13).

[5] 王勖成. 有限单元法[M]. 清华大学出版社, 2003.