流固耦合声学有限元方法理论

目录

简介

本文档描述了用于解决Y型管道中流固耦合声学问题的有限元方法理论基础。该方法模拟了声波在流体域中的传播以及与弹性壁面的相互作用。

计算域包括:

- 流体域（空气，密度 ρ_f = 1.225 kg/m³）
- 固体域（管壁）
- 流固界面（流体与固体的耦合表面）

系统使用谐波激励（频率 ω），所有场变量都以复数形式表示，时间依赖项为 e^(jωt)。

控制方程

流体声学方程

声学流体域由Helmholtz方程控制：

$$\nabla^2 p + k^2 p = 0$$

其中：

- p 是声压
- $k = \omega / c_f$ 是波数
- c_f 是声波在流体中的传播速度（本例中空气声速为343 m/s）
- $\omega = 2\pi f$ 是角频率
- f 是激励频率（Hz）

固体弹性方程

弹性固体域由弹性动力学方程控制：

$$\rho_s \frac{\partial^2 \mathbf{u}}{\partial t^2} - \nabla \cdot \boldsymbol{\sigma} = \mathbf{0}$$

对于简谐激励，这变为：

$$-\omega^2 \rho_s \mathbf{u} - \nabla \cdot \boldsymbol{\sigma} = \mathbf{0}$$

其中：

- $\mathbf{u}$ 是位移向量
- ρ_s 是固体密度
- $\boldsymbol{\sigma}$ 是应力张量

应力和应变的关系通过线性弹性本构方程给出：

$$\boldsymbol{\sigma} = \mathbf{D} \cdot \boldsymbol{\varepsilon}$$

其中：

- $\mathbf{D}$ 是弹性刚度矩阵
- $\boldsymbol{\varepsilon}$ 是应变张量，由位移梯度定义：$\boldsymbol{\varepsilon} = \frac{1}{2}[\nabla \mathbf{u} + (\nabla \mathbf{u})^T]$

流固耦合条件

在流固界面上，以下耦合条件必须满足：

1. **动量守恒**：流体对固体的法向压力等于固体表面的法向应力：

   $$\mathbf{n} \cdot \boldsymbol{\sigma} = -p\mathbf{n}$$

   其中 $\mathbf{n}$ 是从流体指向固体的界面法向量。

2. **连续性条件**：流体和固体的法向加速度相等：

   $$\mathbf{n} \cdot \frac{\partial^2 \mathbf{u}}{\partial t^2} = \mathbf{n} \cdot \frac{1}{\rho_f} \nabla p$$

   对于谐波激励，这简化为：

   $$-\omega^2 \mathbf{n} \cdot \mathbf{u} = \frac{1}{\rho_f} \frac{\partial p}{\partial n}$$

离散化与有限元公式

流体域四面体元

形函数

我们使用线性四面体单元，其形函数为：

$$N_i(\mathbf{x}) = a_i + b_i x + c_i y + d_i z, \quad i = 1, 2, 3, 4$$

其中系数通过求解线性系统确定：

$$\begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{bmatrix} \begin{bmatrix} a_i \\ b_i \\ c_i \\ d_i \end{bmatrix} = \begin{bmatrix} \delta_{i1} \\ \delta_{i2} \\ \delta_{i3} \\ \delta_{i4} \end{bmatrix}$$

形函数的梯度为：

$$\nabla N_i = \begin{bmatrix} b_i \\ c_i \\ d_i \end{bmatrix}$$

四面体体积计算

四面体的体积 V 计算为：

$$V = \frac{|\det(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)|}{6}$$

其中 $\mathbf{v}_i$ 是从节点1到节点$i+1$的向量。

流体单元矩阵

刚度矩阵：

$$K_{ij}^e = \int_{\Omega_e} \nabla N_i \cdot \nabla N_j \, d\Omega \approx V(\nabla N_i \cdot \nabla N_j)$$

质量矩阵：

$$M_{ij}^e = \int_{\Omega_e} N_i N_j \, d\Omega \approx \frac{V}{20}(1 + \delta_{ij})$$

流体域的单元矩阵为：

$$A_f^e = K^e - k^2 M^e$$

其中 $k = \omega / c_f$ 是波数。

固体域四面体元

在固体域，我们处理的是矢量问题，每个节点有3个自由度（位移的x、y、z分量）。

应变-位移矩阵

B矩阵定义了应变和节点位移之间的关系：

$$\boldsymbol{\varepsilon} = \mathbf{B}\mathbf{u}_e$$

对于线性四面体单元：

$$\mathbf{B} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \dots \\ 0 & \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & \dots \\ 0 & 0 & \frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & \dots \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & 0 & \dots \\ 0 & \frac{\partial N_1}{\partial z} & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial z} & \frac{\partial N_2}{\partial y} & \dots \\ \frac{\partial N_1}{\partial z} & 0 & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial z} & 0 & \frac{\partial N_2}{\partial x} & \dots \end{bmatrix}$$

其中我们使用Voigt记号表示应变：$\boldsymbol{\varepsilon} = [\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{zz}, \gamma_{xy}, \gamma_{yz}, \gamma_{zx}]^T$。

弹性矩阵

对于各向同性线性弹性材料，D矩阵为：

$$\mathbf{D} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix}$$

其中：

- E 是杨氏模量
- ν 是泊松比

固体单元矩阵

刚度矩阵：

$$K_s^e = \int_{\Omega_e} \mathbf{B}^T \mathbf{D} \mathbf{B} \, d\Omega \approx V \mathbf{B}^T \mathbf{D} \mathbf{B}$$

质量矩阵：

$$M_s^e = \rho_s \frac{V}{4} \begin{bmatrix} \mathbf{I}_3 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_3 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}_3 \end{bmatrix}$$

其中 $\mathbf{I}_3$ 是3×3单位矩阵，ρ_s 是固体密度。

固体域的单元矩阵为：

$$A_s^e = K_s^e - \omega^2 M_s^e$$

耦合矩阵

耦合矩阵通过计算流固界面上的积分来离散化前面描述的耦合条件。对于界面三角形面片：

流体到固体耦合矩阵：

$$C_{sf}[i,j] = -\int_{\Gamma_{fs}} N_j^f \mathbf{n} N_i^s \, d\Gamma \approx -\frac{S}{3}\mathbf{n}$$

其中：

- S 是界面三角形的面积
- N_j^f 是流体单元的形函数
- N_i^s 是固体单元的形函数

固体到流体耦合矩阵：

$$C_{fs}[i,j] = \rho_f \omega^2 \int_{\Gamma_{fs}} N_i^f \mathbf{n} N_j^s \, d\Gamma \approx \rho_f \omega^2 \frac{S}{3} \mathbf{n}$$

全局系统装配

总体矩阵结构

完整的全局系统具有以下结构：

$$\begin{bmatrix} \mathbf{A}_f & \mathbf{C}_{fs} \\ \mathbf{C}_{sf} & \mathbf{A}_s \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \mathbf{u} \end{bmatrix} = \begin{bmatrix} \mathbf{F}_f \\ \mathbf{F}_s \end{bmatrix}$$

其中：

- $\mathbf{A}_f$ 是流体域的系统矩阵
- $\mathbf{A}_s$ 是固体域的系统矩阵
- $\mathbf{C}_{fs}$ 是固体到流体的耦合矩阵
- $\mathbf{C}_{sf}$ 是流体到固体的耦合矩阵
- $\mathbf{p}$ 是流体节点处的声压未知量
- $\mathbf{u}$ 是固体节点处的位移未知量
- $\mathbf{F}_f$ 和 $\mathbf{F}_s$ 是相应的载荷向量

边界条件

在系统中引入了以下边界条件：

1. **流体入口**：在x≈0处施加声压（Dirichlet条件）：$p = p_{source}$
2. **流体出口**：在x≈length_main和分支末端施加零梯度条件（Neumann条件）：$\frac{\partial p}{\partial n} = 0$
    - 这是一个无反射边界条件，允许声波离开计算域而不产生反射
    - 此条件为FEM的自然边界条件，在组装系统时无需额外修改边界矩阵项
3. **固定边界**：在x=0和r=r_outer处固定固体节点（Dirichlet条件）：$\mathbf{u} = \mathbf{0}$

Dirichlet条件使用惩罚法实现：

$$A_{ii} = \beta, \quad F_i = \beta \cdot p_{prescribed}$$

其中 β 是一个较大的惩罚系数。

实现细节

流体部分

1. 组装流体系统矩阵 $\mathbf{A}_f = \mathbf{K}_f - k^2 \mathbf{M}_f$：

```
def assemble_global_fluid_system(self):
    # 使用预先计算的大小和映射
    n_fluid_local_dof = self.N_fluid_unique # 基于唯一流体节点的大小
    fluid_mapping = self.fluid_mapping

    K_f = torch.zeros((n_fluid_local_dof, n_fluid_local_dof), dtype=torch.float32, dev
    M_f = torch.zeros((n_fluid_local_dof, n_fluid_local_dof), dtype=torch.float32, dev
    F_f = torch.zeros(n_fluid_local_dof, dtype=torch.float32, device=device)

    # 装配K_f和M_f
    for elem in self.fluid_elements:
        coords = self.nodes[elem]
        K_e, M_e = element_matrices_fluid(coords)
        # 将全局元素索引映射到局部流体索引
        local_indices = [fluid_mapping[glob_idx.item()] for glob_idx in elem]

        # 使用局部索引进行装配
        for r_local_map in range(4): # 局部索引中的索引(0-3)
             row_idx = local_indices[r_local_map]
             for c_local_map in range(4):
                  col_idx = local_indices[c_local_map]
                  K_f[row_idx, col_idx] += K_e[r_local_map, c_local_map]
                  M_f[row_idx, col_idx] += M_e[r_local_map, c_local_map]

    k_sq = (self.omega / self.c_f)**2
    A_f = K_f - k_sq * M_f
    return A_f, F_f
```

固体部分

2. 组装固体系统矩阵 $\mathbf{A}_s = \mathbf{K}_s - \omega^2 \mathbf{M}_s$:

```python
def assemble_global_solid_system(self, E, nu, rho_s):
    # 使用预先计算的大小和映射
    n_solid_dof = self.n_solid_dof
    solid_mapping = self.solid_mapping

    K_s = torch.zeros((n_solid_dof, n_solid_dof), dtype=torch.float32, device=device)
    M_s = torch.zeros((n_solid_dof, n_solid_dof), dtype=torch.float32, device=device)
    F_s = torch.zeros(n_solid_dof, dtype=torch.float32, device=device)

    # 装配K_s和M_s
    for elem in self.solid_elements:
        coords = self.nodes[elem]
        K_e, M_e = element_matrices_solid(coords, E, nu, rho_s)
        local_solid_indices = [solid_mapping[glob_idx.item()] for glob_idx in elem]

        for r_local_map in range(4): # 元素节点索引(0-3)
             solid_idx_r = local_solid_indices[r_local_map] # 局部固体索引
             for c_local_map in range(4):
                  solid_idx_c = local_solid_indices[c_local_map]
                  # 从K_e和M_e获取3x3块
                  K_block = K_e[r_local_map*3:(r_local_map+1)*3, c_local_map*3:(c_loca
                  M_block = M_e[r_local_map*3:(r_local_map+1)*3, c_local_map*3:(c_loca
                  # 添加到全局固体矩阵
                  K_s[solid_idx_r*3:(solid_idx_r+1)*3, solid_idx_c*3:(solid_idx_c+1)*3
                  M_s[solid_idx_r*3:(solid_idx_r+1)*3, solid_idx_c*3:(solid_idx_c+1)*3

    # 计算A_s = K_s - omega^2 * M_s
    A_s = K_s - (self.omega**2) * M_s
    return A_s, F_s
```

耦合矩阵

3. 组装耦合矩阵 $\mathbf{C}_{sf}$ 和 $\mathbf{C}_{fs}$:

```
        # 装配耦合矩阵（使用映射）
        C_sf = torch.zeros((n_solid_dof, N_fluid_unique), dtype=torch.float32, device=device)
        C_fs = torch.zeros((N_fluid_unique, n_solid_dof), dtype=torch.float32, device=device)

        if self.interface_idx.numel() > 0:
            interface_node_set = set(self.interface_idx.cpu().numpy())
            # 跟踪对应于self.interface_idx的界面法线
            interface_normals_map = {global_idx.item(): normal_vec for global_idx, normal_vec

            # 遍历流体单元查找界面面
            for elem_nodes in self.fluid_elements:
                # 先检查元素的任何节点是否为界面节点（优化）
                if not any(node.item() in interface_node_set for node in elem_nodes):
                    continue

                # 检查每个面
                local_faces = [(0, 1, 2), (0, 3, 1), (0, 2, 3), (1, 3, 2)]
                nodes_coords_elem = self.nodes[elem_nodes]
                for i_face, local_face in enumerate(local_faces):
                    global_node_indices_face = elem_nodes[torch.tensor(local_face, device=elem_

                    # 确保面的所有节点都在流体和固体映射中且在界面上
                    is_mappable_interface_face = True
                    local_fluid_indices_face = []
                    local_solid_indices_face = []

                    for node_idx_tensor in global_node_indices_face:
                        node_idx = node_idx_tensor.item()
                        if node_idx in interface_node_set and node_idx in fluid_mapping and no
                            local_fluid_indices_face.append(fluid_mapping[node_idx])
                            local_solid_indices_face.append(solid_mapping[node_idx])
                        else:
                            is_mappable_interface_face = False
                            break

                    if is_mappable_interface_face:
                        # 计算法向量和面积
                        p0_idx, p1_idx, p2_idx = global_node_indices_face
                        p0, p1, p2 = self.nodes[p0_idx], self.nodes[p1_idx], self.nodes[p2_idx
                        v1, v2 = p1 - p0, p2 - p0
                        normal_vec_cross = torch.cross(v1.float(), v2.float())
                        face_area = torch.norm(normal_vec_cross) / 2.0

                        if face_area > 1e-12:
                            normal_vec = normal_vec_cross / (2.0 * face_area)
                            # 确保法向量指向流体外部
                            local_idx_p3 = list(set(range(4)) - set(local_face))[0]
                            p3 = nodes_coords_elem[local_idx_p3]
                            face_centroid = (p0 + p1 + p2) / 3.0
                            vec_to_p3 = p3 - face_centroid
                            if torch.dot(normal_vec, vec_to_p3.float()) > 0:
                                normal_vec = -normal_vec

                            # 装配C_sf（作用在固体上的力是-p*n）
                            force_contrib = -(face_area / 3.0) * normal_vec
                            # 装配C_fs（流体方程中的项rho*omega^2*u_n）
                            motion_contrib = rho_f * (self.omega**2) * (face_area / 3.0) * norm

                            # 使用局部索引添加贡献
                            for i_node_face in range(3): # 遍历面节点0,1,2
                                fluid_local_idx = local_fluid_indices_face[i_node_face]
```

```
                        solid_local_idx = local_solid_indices_face[i_node_face]

                        C_sf[solid_local_idx*3:solid_local_idx*3+3, fluid_local_idx] +
                        C_fs[fluid_local_idx, solid_local_idx*3:solid_local_idx*3+3] +
```

全局系统

4. 组装全局矩阵和应用边界条件:

```
# 构造全局块矩阵
global_dim = N_fluid_unique + n_solid_dof
A_global = torch.zeros((global_dim, global_dim), dtype=torch.float32, device=device)
A_global[0:N_fluid_unique, 0:N_fluid_unique] = A_f
A_global[0:N_fluid_unique, N_fluid_unique:] = C_fs
A_global[N_fluid_unique:, 0:N_fluid_unique] = C_sf
A_global[N_fluid_unique:, N_fluid_unique:] = A_s

# 构造全局载荷向量
F_global = torch.cat((F_f, F_s), dim=0)

# 应用边界条件
penalty = self.bcpenalty

# 应用流体入口BC (p = source_value)
for global_idx_tensor in self.near_fluid_idx:
    global_idx = global_idx_tensor.item()
    if global_idx in fluid_mapping:
        local_idx = fluid_mapping[global_idx]
        A_global[local_idx, :] = 0.0
        A_global[:, local_idx] = 0.0
        A_global[local_idx, local_idx] = penalty
        F_global[local_idx] = penalty * source_value

# 应用流体出口BC (p = 0)
for global_idx_tensor in self.outlet_fluid_idx:
    global_idx = global_idx_tensor.item()
    if global_idx in fluid_mapping:
        local_idx = fluid_mapping[global_idx]
        A_global[local_idx, :] = 0.0
        A_global[:, local_idx] = 0.0
        A_global[local_idx, local_idx] = penalty
        F_global[local_idx] = 0.0

# 应用固体固定BC (u = 0)
for global_node_idx_tensor in self.fixed_solid_nodes_idx:
    global_node_idx = global_node_idx_tensor.item()
    if global_node_idx in solid_mapping:
        solid_local_idx = solid_mapping[global_node_idx]
        # 计算全局DOF索引
        global_dof_indices = [N_fluid_unique + solid_local_idx*3 + i for i in range(3)

        for dof_idx in global_dof_indices:
            A_global[dof_idx, :] = 0.0
            A_global[:, dof_idx] = 0.0
            A_global[dof_idx, dof_idx] = penalty
            F_global[dof_idx] = 0.0
```

求解

5. 求解全局系统：

```
def solve(self, E, nu, rho_s):
    """
    给定材料参数，组装全局系统并求解，返回：
      - 预测远端麦克风处流体声压（取 fluid 域 x≈1.0 点平均）
      - 全局解向量 u（其中 u[0:n_nodes] 为 fluid 声压）
    """
    A_global, F_global, N_fluid_unique, n_solid_dof_actual = self.assemble_global_syst
    print("[info] 开始求解 (mapped system)")
    try:
        u = torch.linalg.solve(A_global.float(), F_global.float())
    except torch._C._LinAlgError as e:
        print(f"Solver Error: {e}")
        print("Matrix might still be singular or ill-conditioned.")
        torch.save(A_global, 'A_global_error.pt')
        torch.save(F_global, 'F_global_error.pt')
        raise
    print("[info] 求解完成")

    # 提取麦克风处的声压
    p_mic = torch.tensor(0.0, device=device, dtype=u.dtype)
    if self.mic_node_idx is not None:
        global_mic_idx = self.mic_node_idx.item()
        if global_mic_idx in self.fluid_mapping:
            local_mic_idx = self.fluid_mapping[global_mic_idx]
            if local_mic_idx < N_fluid_unique:
                p_mic = u[local_mic_idx]
            else:
                print(f"[warning] Mapped mic index {local_mic_idx} out of bounds for f
        else:
            print(f"[warning] Global mic node index {global_mic_idx} not found in flui
    else:
        print("[warning] Mic node index not defined.")

    return p_mic.squeeze(), u
```

求解过程

1. **网格读取与处理**：从gmsh文件读取四面体网格，识别流体、固体和界面部分
2. **模型初始化**：识别边界条件，创建节点映射，计算界面法向量
3. **系统组装**：装配流体和固体系统矩阵，计算耦合矩阵，构建全局矩阵系统
4. **边界条件应用**：添加入口声源、出口零声压和固体固定约束
5. **求解**：求解线性系统得到声压和固体位移
6. **后处理**：提取特定位置（例如麦克风位置）的解，可视化结果

参考文献

1. Zienkiewicz, O. C., & Taylor, R. L. (2000). The Finite Element Method: Solid Mechanics (5th ed., Vol. 2). Butterworth-Heinemann.
2. Petyt, M. (2010). Introduction to Finite Element Vibration Analysis (2nd ed.). Cambridge University Press.
3. Sandberg, G., & Wernberg, P. A. (1995). A symmetric finite element formulation for acoustic fluid-structure interaction analysis. Journal of Sound and Vibration, 123(3), 507-515.

4. Everstine, G. C. (1981). A symmetric potential formulation for fluid-structure interaction. Journal of Sound and Vibration, 79(1), 157-160.