jCODE Manual

March 15, 2022

Contents

1 Introduction

`jCODE` is a high-performance, Fortran/MPI-based code that solves the viscous compressible Navier–Stokes equations on a structured curvilinear grid. The underlying numerical discretization was adapted from the code written by Dr. Ramanathan Vishnampet [5]. The code is developed and maintained by the Capecelatro Research Group at the University of Michigan. It features a range of models and methods designed for solving turbulent reacting multiphase flows. Various features include:

- High-order energy-stable discretization based on summation-by-parts finite differences
- Lagrangian particle tracking with two-way coupling for mass, momentum, and heat transfer between phases
- Immersed boundary method for treatment of complex geometries
- Shock capturing based on localized artificial dissipation
- Discrete adjoint method for obtaining precision-limited sensitivity gradients

This manual provides the necessary details for acquiring the code, compilation instructions, and several representative example cases. The following section will provide a brief description of the underlying physical models and numerical discretization. Further information can be found in the papers referenced throughout the document.

2 Physical model

2.1 Non-dimensionalization

The code uses the following variables for non-dimensionalization: ρ^*_∞ (density), c^*_∞ (the speed of sound), μ^*_∞ (viscosity), $C^\star_{p,\infty}$ (the heat capacity), and L^*, a length scale. For multi-component flows, the molecular weight $W^\star_\infty$ is also used. The superscript $\star$ denotes

a dimensional quantity and subscript ∞ represents a reference quantity. With this, the non-dimensional variables are:

$$\begin{aligned} x_i &= \frac{x_i^\star}{L^\star}, & u_i &= \frac{u_i^\star}{c_\infty^\star}, & t &= \frac{t^\star}{L^\star/c_\infty^\star}, \\ \rho &= \frac{\rho^\star}{\rho_\infty^\star}, & p &= \frac{p^\star}{\rho_\infty^\star {c_\infty^\star}^2}, & E &= \frac{E^\star}{{c_\infty^\star}^2} \\ C_{p,k} &= \frac{C_{p,k}^\star}{C_{p,\infty}^\star}, & W_k &= \frac{W_k^\star}{W_\infty^\star}, & \mu &= \frac{\mu^\star}{\mu_\infty^\star} \\ T &= \frac{T^\star}{{c_\infty^\star}^2/C_{p,\infty}^\star}, & T &= \frac{T^\star}{(\gamma-1)T_\infty^\star}, & \lambda &= \frac{\lambda^\star}{\mu_\infty^\star} \end{aligned}$$

where x_i and u_i are the location and velocity in the i-th direction, respectively, ρ is the density, p is the pressure, E is the total energy per unit mass, γ is the specific heat ratio, $C_{p,k}$ is the specific heat at constant pressure for species k, W_k is the molecular weight of species k, and μ and λ are the first and second viscosity coefficients, respectively.

The relevant non-dimensional numbers defined in `jCODE` are

$$\mathrm{Re}_c = \frac{\rho_\infty^\star c_\infty^\star L^\star}{\mu_\infty^\star}, \quad \mathrm{Pr} = \frac{C_{p,\infty}^\star \mu^\star}{k^\star}, \quad \mathrm{Sc}_k = \frac{\mu^\star}{\rho^\star D_k^\star}, \quad \mathrm{Fr}_c = \frac{{c_\infty^\star}^2}{g^\star L^\star}, \tag{1}$$

where $g^\star$ is the magnitude of gravity and $D_k^\star$ denotes the mass diffusivity of species k.

2.2 Governing equations and discretization

The code solves the viscous compressible Navier–Stokes equations with options to include multiple species, Lagrangian particles, and/or immersed boundaries. When two-way coupling between the fluid and particles is enabled, the code solves the volume-filtered equations with appropriate source terms [11]. Details on the fluid-phase equations are given in Sec. 2.2.1, the particle equations of motion are provided in Sec. 2.2.2, and details on two-way coupling can be found in Sec. 2.2.3.

2.2.1 Gas-phase description

The conserved variables are density ρ, momentum in the i-th direction ρu_i, total energy ρE, and mass fraction for species k, ρY_k for $k = 1, \ldots, N$ with N denoting the total number of species. The governing equations for the gas-phase can be written compactly as

$$\frac{\partial \boldsymbol{Q}}{\partial t} + \frac{\partial \boldsymbol{F}_i}{\partial x_i} = \boldsymbol{S}, \tag{2}$$

where $\boldsymbol{Q} = [\rho,\ \rho u_1,\ \rho u_2,\ \rho u_3,\ \rho E,\ \rho Y_1,\ \cdots,\ \rho Y_{N-1}]^\mathsf{T}$ is a vector containing the conservative variables, $\boldsymbol{F}_i = \boldsymbol{F}_i^I - \mathbf{F}_i^V$ is the flux vector containing inviscid $\boldsymbol{F}_i^I$ and viscous $\boldsymbol{F}_i^V$

effects, and $\boldsymbol{S}$ is a vector of volumetric source terms (e.g., due to gravity, two-way coupling with particles, or combustion). The inviscid and viscous fluxes in direction i are

$$\boldsymbol{F}_i^I = \begin{bmatrix} \rho u_i \\ \rho u_1 u_i + p\delta_{i1} \\ \rho u_2 u_i + p\delta_{i2} \\ \rho u_3 u_i + p\delta_{i3} \\ u_i(\rho E + p) \\ \rho Y_1 u_i \\ \vdots \\ \rho Y_{N-1} u_i \end{bmatrix}, \quad \boldsymbol{F}_i^V = \begin{bmatrix} 0 \\ \tau_{1i} \\ \tau_{2i} \\ \tau_{3i} \\ u_j \tau_{ij} - q_i \\ f_1^V \\ \vdots \\ f_{N-1}^V \end{bmatrix}. \tag{3}$$

where τ_{ij} denotes non-dimensional viscous stress tensor, given by

$$\tau_{ij} = \frac{\mu}{\mathrm{Re}_c}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) + \frac{\lambda}{\mathrm{Re}_c}\frac{\partial u_k}{\partial x_k}\delta_{ij}. \tag{4}$$

The non-dimensional heat flux is given by

$$q_i = -\frac{\mu}{\mathrm{Re}_c \,\mathrm{Pr}}\frac{\partial T}{\partial x_i} - \frac{\mu T}{\mathrm{Re}_c}\sum_{k=1}^{N}\frac{1}{W_k \mathrm{Sc}_k}\frac{\partial Y_k}{\partial x_i}, \tag{5}$$

and the non-dimensional mass flux is given by

$$f_k^V = \frac{\mu}{\mathrm{Re}_c \mathrm{Sc}_k}\frac{\partial Y_k}{\partial x_i}, \quad \text{for } k = 1, \ldots, N-1. \tag{6}$$

Pressure and temperature are dependent variables that are determined according to thermodynamic relations and an equation of state, given by

$$p = (\gamma - 1)\left(\rho E - \frac{1}{2}\rho u_i u_i\right) \quad \text{and} \quad T = \frac{\gamma}{\gamma - 1}\frac{pW}{\rho}, \tag{7}$$

where W is the mixture molecular weight. For a single component fluid, $W = 1$ and for a mixture, it is defined as

$$\frac{1}{W} = \sum_{k=1}^{N}\frac{Y_k}{W_k}. \tag{8}$$

Finally, μ varies with temperature according to $\mu = [(\gamma - 1)T]^n$, and $\lambda = \mu_B - 2\mu/3$ where $\mu_B = 0.6\mu$ is the bulk viscosity.

The governing equations are discretized in generalized curvilinear coordinates. Physical coordinates $\boldsymbol{x} = (x_1, x_2, x_3)$ are transformed to computational coordinates $\boldsymbol{\xi} = (\xi_1, \xi_2, \xi_3)$ via

$$\boldsymbol{x} = X(\boldsymbol{\xi}) \quad \text{with inverse} \quad \boldsymbol{\xi} = \Xi_i(\boldsymbol{x}), \tag{9}$$

so that $X^{-1} = \Xi$. The transformation Jacobian $J = \det\left(\frac{\partial \xi_i}{\partial x_j}\right)$ is positive definite, and metrics are defined $\mathrm{M}_{ij} = J^{-1}\frac{\partial \xi_i}{\partial x_j}$ with $\frac{\partial \mathrm{M}_{ij}}{\partial \xi_i} = 0$, so the transformed flow equations are

$$\frac{\partial \boldsymbol{Q}}{\partial t} + J\frac{\partial}{\partial \xi_i}\left[\mathrm{M}_{ij}\left(\boldsymbol{F}_j^I - \boldsymbol{F}_j^V\right)\right] = \boldsymbol{S}. \tag{10}$$

Derivatives $\partial/\partial\xi_i$ are approximated by a narrow-stencil finite-difference operator that satisfies the summation-by-part (SBP) property [6]. This leads to $2s$-order centered-difference stencils at interior points and s-order accurate biased stencils near boundaries, with $s+1$ global accuracy. To evaluate second and mixed derivatives, first derivative operators are applied consecutively, necessitating the use of artificial dissipation to damp the highest wavenumber components supported by the grid. High-order accurate SBP dissipation operators may be used to provide artificial dissipation with a diffusion coefficient that is a function of the local grid resolution [7, 5]. The boundary conditions are imposed weakly to ensure provable stability by employing the simultaneous-approximation-term (SAT) boundary treatment [8, 5]. A list of available boundary conditions is provided in Sec. 4.2.

Kinetic energy preservation is achieved using a skew-symmetric-type splitting of the inviscid flux in generalized curvilinear coordinates [9], which provides nonlinear stability at low Mach number. The convective flux appearing in (10) is expressed in split form as

$$\frac{\partial \rho\hat{u}_i\varphi}{\partial \xi_i} = \frac{1}{2}\frac{\partial \rho\hat{u}_i\varphi}{\partial \xi_i} + \frac{1}{2}\varphi\frac{\partial \rho\hat{u}_i}{\partial \xi_i} + \frac{1}{2}\rho\hat{u}_i\frac{\partial \varphi}{\partial \xi_i}, \tag{11}$$

where $\hat{u}_i = \mathrm{M}_{ij}u_j$ is the contravariant velocity and φ is a generic transported scalar, being unity for the continuity equation, u_i for the momentum equation, and $H = E + p/\rho$ for the total energy equation. The pressure gradient term in the momentum equation is split in a similar fashion according to

$$\frac{\partial p\mathrm{M}_{ij}}{\partial \xi_i} = \frac{1}{2}\frac{\partial p\mathrm{M}_{ij}}{\partial \xi_i} + \frac{1}{2}p\frac{\partial \mathrm{M}_{ij}}{\partial \xi_i} + \frac{1}{2}\mathrm{M}_{ij}\frac{\partial p}{\partial \xi_i}. \tag{12}$$

2.2.2 Particle-phase description

Particles are modeled as discrete entities that are tracked in a Lagrangian manner. The displacement and velocity of an individual particle i are given by Newton's second law according to

$$\frac{\mathrm{d}\boldsymbol{x}_p^{(i)}}{\mathrm{d}t} = \boldsymbol{v}_p^{(i)} \quad \text{and} \quad m_p\frac{\mathrm{d}\boldsymbol{v}_p^{(i)}}{\mathrm{d}t} = \boldsymbol{f}_{\text{inter}}^{(i)} + \boldsymbol{f}_{\text{col}}^{(i)}, \tag{13}$$

where $m_p = \pi\rho_p d_p^3/6$ is the particle mass with ρ_p the particle density, $\boldsymbol{f}_{\text{inter}}^{(i)}$ accounts for fluid stresses at the particle surface, and $\boldsymbol{f}_{\text{col}}^{(i)}$ accounts for particle-particle interactions (e.g., collisions). A soft-sphere collision model is used to handle particle-particle and particle-wall contact. Particles are treated as elastic and frictional with a coefficient of restitution and friction coefficient provided by the user. Collisions and handled efficiently using a nearest-neighbor detection algorithm [10]. Momentum exchange between the phases is decomposed into resolved and unresolved contributions, expressed as

$$\boldsymbol{f}_{\text{inter}}^{(i)} = \mathcal{V}_p\nabla\cdot(-p\mathbb{I} + \boldsymbol{\tau}) + \boldsymbol{f}_{\text{drag}}^{(i)}, \tag{14}$$

where $\mathcal{V}_p$ is the particle volume, and $\boldsymbol{f}_{\text{drag}}^{(i)}$ can be handled using a variety of drag laws available in the code. A list of available drag models is provided in Sec. 4.5. Further details are provided in Appendix A.

The evolution of particle temperature is expressed as

$$m_p C_{p,p} \frac{\mathrm{d}T_p^{(i)}}{\mathrm{d}t} = \mathcal{V}_p \nabla \cdot \left(\frac{\mu}{\mathrm{Re}_c \, \mathrm{Pr}} \nabla T \right) + \boldsymbol{q}_{\text{inter}}^{(i)}, \tag{15}$$

where $T_p^{(i)}$ is the temperature of the i-th particle, $C_{p,p}$ is the ratio of particle-to-fluid heat capacity, and the sub-filtered heat flux, $\boldsymbol{q}_{\text{inter}}^{(i)}$ is modeled using Nusselt number correlations.

2.2.3 Two-way coupling

When two-way coupling is enabled between the fluid and particle phases, the governing equations are modified to account for the fluid volume fraction, $0 \leq \alpha \leq 1$, and interphase exchange of mass, momentum, and energy. The fluid-phase conserved variables are modified as $\boldsymbol{Q} = [\alpha\rho,\ \alpha\rho u_1,\ \alpha\rho u_2,\ \alpha\rho u_3,\ \alpha\rho E,\ \alpha\rho Y_1,\ \cdots,\ \alpha\rho Y_{N-1}]^{\mathsf{T}}$. Lagrangian particle data is projected to the mesh using a filter kernel $\mathcal{G}$ with a chosen size δ_f. Interphase exchange terms are given by

$$\text{Volume fraction:} \quad \alpha = 1 - \sum_{i=1}^{N_p} \mathcal{G}\big(|\boldsymbol{x} - \boldsymbol{x}_p^{(i)}|\big)\, \mathcal{V}_p, \tag{16}$$

$$\text{Momentum exchange:} \quad \boldsymbol{\mathcal{F}} = -\sum_{i=1}^{N_p} \mathcal{G}\big(|\boldsymbol{x} - \boldsymbol{x}_p^{(i)}|\big)\, \boldsymbol{f}_{\text{inter}}^{(i)}, \tag{17}$$

$$\text{Work due to momentum exchange:} \quad \boldsymbol{u}_p \cdot \boldsymbol{\mathcal{F}} = -\sum_{i=1}^{N_p} \mathcal{G}\big(|\boldsymbol{x} - \boldsymbol{x}_p^{(i)}|\big)\, \boldsymbol{v}_p^{(i)} \cdot \boldsymbol{f}_{\text{inter}}^{(i)}, \tag{18}$$

$$\text{Heat exchange:} \quad \mathcal{Q} = -\sum_{i=1}^{N_p} \mathcal{G}\big(|\boldsymbol{x} - \boldsymbol{x}_p^{(i)}|\big)\, \boldsymbol{q}_{\text{inter}}^{(i)}. \tag{19}$$

A two step filtering approach [10] is employed that decouples the mesh size from particle diameter ratio in an efficient manner. First, particle data is sent to neighboring grid points via trilinear extrapolation. The solution is then diffused such that the filtered data is projected with characteristic size of δ_f. To avoid restrictive time step constraints in the diffusion process, the latter step is solved implicitly via approximate factorization with a second order alternating direction implicit (ADI) scheme.

2.3 Time integration

A family of explicit Runge–Kutta (RK) time integration schemes are available in the code. This includes a first-order RK scheme (Euler method); second-order RK (predictor-corrector scheme); an optimal third-order total variation diminishing (TVD) RK method [12];

and a standard fourth-order scheme (RK4). All modules employ the same time marching scheme (e.g., gas-phase equations, particles, immersed boundaries). However, at present, the adjoint solver is only implemented using the RK4 scheme.

2.4 Adjoint method

The adjoint of the perturbed and linearized governing equations can be solved in continuous or discrete form. The adjoint provides the sensitivity gradient of a quantity of interest $\mathcal{J}$ (referred to as the cost functional) to a chosen set of parameters $\boldsymbol{\theta}$ (also referred to as the control parameters in `jCODE`). A list of available cost functionals and control parameters is provided in Sec. 4.7. When the discrete adjoint method is used, the code returns an exact (to machine precision) sensitivity gradient consistent with the SBP-SAT discretization described in the previous section.

In general, we write the cost functional $\mathcal{J}$ as

$$\mathcal{J}(\boldsymbol{Q},\boldsymbol{\theta}) = \int_{t_1}^{t_2}\int_{\Omega} j(\boldsymbol{Q},\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}t, \tag{20}$$

where $j(\boldsymbol{Q},\boldsymbol{\theta})$ is the the cost functional integrated in a target region Ω over duration t_1 to t_2. The change, or first variation, in $\delta\mathcal{J}$ is written as

$$\delta\mathcal{J} = \int_{t_1}^{t_2}\int_{\Omega}\left\{\left(\frac{\partial j}{\partial \boldsymbol{Q}}\right)^{\mathsf{T}}\delta\boldsymbol{Q} + \left(\frac{\partial j}{\partial \boldsymbol{\theta}}\right)^{\mathsf{T}}\delta\boldsymbol{\theta}\right\}\mathrm{d}\boldsymbol{x}\,\mathrm{d}t. \tag{21}$$

Here, $\delta\boldsymbol{Q}$ represents linear perturbations to the flow solution due to variations in the parameters $\delta\boldsymbol{\theta}$ (which in general would require repeated simulations). The adjoint equations are formulated to eliminate the $\delta\boldsymbol{Q}$ dependence in (21). To do this, we introduce the adjoint variable $\boldsymbol{Q}^{\dagger}$ as a Lagrange multiplier to enforce adherence to the governing equations (10) as a constraint, written as

$$\mathcal{N}[\boldsymbol{Q};\boldsymbol{\theta}] = 0 \quad \text{where} \quad \mathcal{N}[\boldsymbol{Q};\boldsymbol{\theta}] = \frac{\partial \boldsymbol{Q}}{\partial t} + J\frac{\partial}{\partial \xi_i}\left[\mathrm{M}_{ij}\boldsymbol{F}_j(\boldsymbol{Q})\right] - \boldsymbol{S}(\boldsymbol{Q},\boldsymbol{\theta}). \tag{22}$$

Variations with respect to the flow solution $\boldsymbol{Q}$ result in a set of partial differential equations that describe the evolution of the adjoint variables

$$\mathcal{N}^{\dagger}[\boldsymbol{Q}^{\dagger};\boldsymbol{Q}] = -\frac{\partial \boldsymbol{Q}^{\dagger}}{\partial t} - J\left[\mathcal{A}_i^{\mathsf{T}}\frac{\partial \boldsymbol{Q}^{\dagger}}{\partial \xi_i} + \mathcal{D}^{\mathsf{T}}\frac{\partial}{\partial \xi_j}\left(\mathcal{B}_{ij}^{\mathsf{T}}\frac{\partial \boldsymbol{Q}^{\dagger}}{\partial \xi_i}\right)\right] + \mathcal{D}^{\mathsf{T}}\mathcal{C}^{\mathsf{T}}\boldsymbol{Q}^{\dagger} = 0. \tag{23}$$

Due to the the negative sign that appears in front of the time derivative, these equations have to be solved backward in time. The coefficient matrices $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$ can be found in [13]. Because the flow variables appear in the adjoint equations as coefficients, they must be available. A standard checkpointing scheme is utilized to do this efficiently.

Once the adjoint variables are solved for, the corresponding sensitivity gradient can be determined according to

$$\frac{\delta\mathcal{J}}{\delta\boldsymbol{\theta}} = \int_{t_1}^{t_2}\int_{\Omega}\left(\frac{\partial j}{\partial \boldsymbol{\theta}} - \boldsymbol{Q}^{\dagger\mathsf{T}}\frac{\partial \boldsymbol{S}}{\partial \boldsymbol{\theta}}\right)\mathrm{d}\boldsymbol{x}\,\mathrm{d}t, \tag{24}$$

with $\delta\mathcal{J}$ now directly related to $\delta\boldsymbol{\theta}$, independent of $\delta\boldsymbol{Q}$. Several optimization routines exist in the code that make use of this sensitivity gradient.

3 Source code and compilation

3.1 Obtaining the source code

The code is currently managed via a Git repository stored on `bitbucket.org`. To obtain the source code, you must first become a member of Bitbucket and request access to the `jCODE` repository by sending an e-mail to `jcaps@umich.edu`. In the email provide your name, institution, and general purpose for using the code. Once you have access, you should create a branch by clicking '*create branch*' found at the top-left corner of this page. Follow the prompts to create a branch stemming from 'main' and choose an appropriate name.

Next, clone your branch to your local machine. This requires that you download Git. From the terminal window create a new directory called `jcode` then change into this directory and type:

```
git clone https://yourBitbucketName@bitbucket.org/jcaps/jcode.git
-b <branch name>
```

Note: Mac users have an alternative to using Git from the command line. There exists free software called SourceTree, a graphical user interface (GUI) integrated with Bitbucket. For users opting to use SourceTree, click '*clone*' at the top-left of the main menu in SourceTree.

Once you have access to the source code, you will find that `jCODE` contains several directories. Their contents and a brief description are outlined in Fig. 1.

Figure 1: Summary of the jCODE file structure

3.2 Compilation details

In order to install jCODE you must have Fortran 90 compilers and MPI installed. Once these are installed, you may install the CFD code itself. This section describes how to complete the download, configuration, and installation.

The source code of jCODE is available in the src directory. The code requires two files to compile: Makefile and Makefile.in. The file named Makefile is generic and architecture/compiler independent. It does not require modification. The file named Makefile.in contains architecture/compiler information that will need to be written for your particular hardware. There is no default Makefile.in file when you download the code for the first time. However, you can find templates for different architectures and compilers in the Makefiles directory. Chose an appropriate template from the Makefiles directory and copy it into src. Ensure that Makefile.in is adapted to your architecture and set of compilers. Modify the following section of Makefile.in to match your local installation locations.

```
CC =<path to MPICH>/bin/mpicc
CXX=<path to MPICH>/bin/mpicxx
F90=<path to MPICH>/bin/mpif90
F77=<path to MPICH>/bin/mpif77
LD =<path to MPICH>/bin/mpif90
```

Choosing the type of compilation

The following compilation commands are executed from the command line while you are in the same folder as Makefile.in.

make opt: This target is generally for production runs, enables all optimization flags and the code will run the fastest. It is the most general way of compiling the code.

make debug: Use this target to compile the code with all debug flags enabled (additional flags can be added in Makefile.in if desired). Depending on the set of compilers, these flags might include: detection of unused variables, uninitialized variables, out-of bounds arrays, unassociated pointers, etc. If you suspect a bug in the code this is a good starting place for finding it. Note, the code will run $\sim 4\times$ slower compared to optimized mode.

make clean: Use this target if you want to remove all modules (.mod in mod), all object files (.o in obj), and all libraries (.a in lib) generated during compilation of the main source code. If the code is already compiled, you must run make clean before compiling in a different mode.

make distclean: This will remove all temporary files associated with the utils directory.

It is good practice to start a new build with the following commands:

```
make clean
make distclean
make debug -OR- make opt
```

Note that compilation might take up to 10 minutes on certain machines. After the code is compiled, a number of executable files can be found in the `bin` directory.

4 Input deck

`input` is a text file that specifies the parameters used during initialization and run time. Comments begin with either # or !. Empty lines are permitted. The 'initialization parameters' are only read when `init_flow` is executed. All other parameters are read during the execution of `jCODE` itself. The following convention is used in this section:

`PARAMETER : DBLE`	The value of `PARAMETER` is a double precision number
`PARAMETER : INT`	The value of `PARAMETER` is an integer
`PARAMETER : STRING`	The value of `PARAMETER` is a character string
`PARAMETER : LOGICAL`	The value of `PARAMETER` is a Fortran logical, `.true.` or `.false.`
`PARAMETER : A B C`	The value of `PARAMETER` is an array and can be of mixed data types

4.1 Initialization parameters

```
# Initialization parameters ##############################################
```

```
simulation name : STRING
```

Determine the type of simulation. Examples will be given in the following section.

```
! Grid size
nx : INT
ny : INT
nz : INT
```

Specify the number of grid points in each direction.

```
! Domain size
Lx : INT
Ly : INT
Lz : INT
```

Specify the domain size in each direction.

```
curvilinear domain : LOGICAL
```

Determine whether curvilinear coordinates should be used.

```
periodicity type in x : STRING
periodicity type in y : STRING
periodicity type in z : STRING
```

Specify the periodicity type in direction `x`, `y`, or `z`. Options include `plane`, `overlap`, or `polar`, If not specified the default is `none`.

```
! Initialization files
init grid file     : STRING
init solution file : STRING
```

Specify the name for the grid file and solution file that will be written. By default, they are `grid` and `data.init`, respectively.

4.2 Boundary conditions and patches

```
# Boundary conditions and patches ######################################
```

```
! Patches :
! Name            Type    normDir   iMin  iMax   jMin  jMax   kMin  kMax
! ------------------------------------------------------------------------
  STRING          STRING  INT       INT   INT    INT   INT    INT   INT
```

Grid patches are used in the code to execute specific functions within subregions of the domain (such as boundary conditions). `Name` can be any unique string. `Type` specifies which patch to use. A list of patch types can be found below. `normDir` represents the outward normal direction from the surface into the domain (when applied to domain boundaries), which can be either ± 1, ± 2 or ± 3. `iMin, iMax, jMin, jMax, kMin, kMax` specify the region where the patch will be active. A negative value represents the number of grid points from the end. For example `iMin = 1, iMax = -1` will make the patch active across all grid points in the x-direction.

`SAT_FAR_FIELD`

Inflow/outflow characteristic boundary condition. Must be defined as a surface in 3D or a line in 2D.

`SAT_SLIP_WALL`

No penetration boundary condition. Must be defined as a surface in 3D or a line in 2D.

`SAT_ISOTHERMAL_WALL`

No-slip isothermal boundary condition. Must be defined as a surface in 3D or a line in 2D.

`SAT_ADIABATIC_WALL`

No-slip adiabatic wall boundary condition. Must be defined as a surface in 3D or a line in 2D.

`SPONGE`	Used to dampen the solution to a specified target solution near the boundary. Defined as a volume in 3D and an area in 2D.
`VISUALIZATION`	Used to define a specific region to write visualization files. If not specified, the post processing routine will output data of the entire grid size.
`ACTUATOR`	The control region used during an adjoint run. Must be defined as a volume in 3D and an area in 2D.
`COST_TARGET`	The target region used to compute the cost functional during an adjoint run. Must be defined as a volume in 3D and an area in 2D.

4.3 General runtime parameters

`! Files` `data overwrite : LOGICAL`	Specifies if the data file is overwritten during runtime. The default is false for adjoint simulations and is true otherwise.
`use serial io : LOGICAL`	Specifies if processors read/write data to separate files (serial I/O) or to a single file (parallel I/O). The default is false.
`grid file : STRING` `solution file to read : STRING`	Specifies the name of the grid and solution file to be read by `jCODE` (required).
`solution file to write : STRING`	Specifies the name for writing the solution file (required).
`save interval : INT`	Specifies the frequency for writing solution files.

`! Target state`
`use target state : LOGICAL`
`target state file : STRING`
	Specifies if a target file is provided for setting the boundary values (the default is true).

`! Manual processor decomposition` `use manual domain decomposition : LOGICAL`	Choose whether or not manual processor decomposition should be used. The default value is false. If false, the code will automatically determine how best to distribute the processors across the domain.
`processor decomposition : INT INT INT`	Specify the processor decomposition in each direction if manual domain decomposition is used.
`! Thermo-fluid properties` `include viscous terms : LOGICAL`	Specifies if the flow is viscous. Its default value is false (inviscid flow).
`Reynolds number : DBLE`	Specifies the reference Reynolds number $\mathrm{Re}_c = \rho^\star_\infty L^\star c^\star_\infty / \mu^\star_\infty$.
`viscosity power law exponent : DBLE`	Specifies the temperature exponent of the viscosity, i.e. $\mu/\mu_\infty \propto (T/T_\infty)^n$. The default value is 0.666.
`bulk viscosity ratio : DBLE`	Specifies the ratio of the bulk viscosity to the dynamic viscosity. The default is 0.6.
`Prandtl number : DBLE`	Specifies the reference Prandtl number $\mathrm{Pr} = C^\star_{p,\infty}\mu^\star/k^\star$. Its default value is 0.7.
`ratio of specific heats : DBLE`	Specifies the ratio of specific heats, $\gamma = C_p/C_v$, which is constant in `jCODE` . Its default value is 1.4.
`! Chemistry` `equation of state : STRING`	Specifies the equation of state. It can be either `ideal gas` or `mixture ideal gas`. The default is `ideal gas`.

`! Gravity`
`include gravity : LOGICAL`
`gravity norm : DBLE DBLE DBLE`
`Froude number : DBLE`
	Specifies if the gravity force exists (by default is false). If it exists, then `gravity norm` must be set as a unit vector to specify the direction of gravity in each direction. `Froude number` specifies the magnitude of gravity according to $\mathrm{Fr}_c = {c^\star_\infty}^2/(g^\star L^\star)$.

```
! Discretization scheme
default discretization scheme : STRING
first derivative scheme : STRING
```

Specifies the discretization schemes as defined in Sec. 2.2. The options are `SBP 1-2`, `SBP 2-4`, `SBP 3-6`, `SBP 4-8` (default), and `DRP 13 point`.

```
time integration scheme : STRING
```

Specifies the time integration scheme to be used. The options are `EULER`, `RK2`, `RK3-TVD`, and `RK4` (default).

```
convective splitting : LOGICAL
```

Specified whether or not to use a split-convective form of the inviscid fluxes. The default value is true.

```
! Artificial dissipation
add dissipation    : LOGICAL
dissipation amount : DBLE
```

Specifies if artificial dissipation should be used and the dissipation amount. The default is false. Otherwise, `dissipation amount` is required.

```
composite dissipation : LOGICAL
```

Specifies if the composite dissipation operator is used (if true, dissipation is not a function of local grid spacing).

```
artifical dissipation scheme : STRING
```

Specifies the artificial dissipation scheme. By default it is the same as `default discretization scheme`.

```
! Shock capturing
use shock capturing : LOGICAL
shock capturing coefficient : DBLE
```

Specifies if there is a shock capturing scheme (false by default). If true, then the shock capturing coefficient may be specified (default value is 1).

```
! Time stepping options
use constant CFL mode : LOGICAL
cfl                   : DBLE
time step size        : DBLE
```

Specifies if the CFL number is constant (default) or not. If true, then its value is set by `cfl` (default is 0.5) and the timestep size is computed during run time. Otherwise, `time step size` is required.

```
number of timesteps : INT
```

Specifies the number of simulation timesteps (required).

```
report interval : INT
```

Specifies the frequency of reporting data to the screen and `/monitor`. The default is 1.

```
! Solution limits
enable solution limits : LOGICAL
```

Specifies if the density, temperature, and mass fraction (for multi-component flows) must be within certain intervals. The default is false. Otherwise, must specify: `minimum density, maximum density, minimum temperature, maximum temperature` and `minimum mass fraction, maximum mass fraction`. The code will end when the quantities are outside these values.

4.4 Species / combustion parameters

```
Schmidt number : DBLE DBLE ...
```

Specifies the Schmidt number for each species $\mathrm{Sc}_k = \mu^\star/(\rho^\star D_k^\star)$, for multi-component flows. It can be a single value for all species or a list of values for each species. By default $\mathrm{Sc}_k = \mathrm{Pr}$.

```
Atwood number: DBLE
```

Specifies the Atwood number $\mathrm{At} = (W_1 - W_2)/(W_1 + W_2)$ for two-component flows with W_1 and W_2 denoting the molecular weight of the heavy and light fluids, respectively.

```
reference species: STRING
```

Specifies the reference species for the molecular weight for multi-component flow. It can be set as `air, Ar, C, CH4, CO, CO2, H, H2, H2O, HO2, H2O2, N, N2, O, O2, OH`. The default is `air`.

```
active species: STRING STRING ...
```

Specifies the active species (during combustion). If there are N species, then $N - 1$ names must be provided.

```
inert species: STRING
```

Specifies the inert species (during combustion). The default is `N2`.

`combustion model: STRING`

Specifies the combustion model. It can be `none` (no combustion, default), `one-step` (one-step irreversible reaction), or `boivin skeletal` (12-step hydrogen combustion). Note that each combustion model may have its own parameters.

4.5 Particle parameters

`! Particle parameters`

`include particles : LOGICAL`

Determine whether particles are present in the simulation.

`particle density : DBLE`

The non-dimensional particle density $\rho_p^\star/\rho_\infty^\star$.

`two way coupling : LOGICAL`

Determine whether two-way coupling is used.

`particle collisions : LOGICAL`

Determine whether particles collisions should be enabled.

`collision time : DBLE`

The time spent during each collision when `particle collision` is true. General rule of thumb: the collision time should be 20–30 times the simulation time step.

`coefficient of restitution : DBLE`

Determine the coefficient of restitution, the value should be between 0 and 1 and by default is 0.85.

`drag model : STRING`

Specify the drag model. Available drag models include `stokes`, `tenneti`, `gidaspow`, `henderson`, `loth`, `parmar`, `basset`, or `schiller naumann`, with `stokes` being the default. If `none` is selected the drag force is set to zero.

`particle filter size : DBLE`

Determine the filter size for projecting data to the grid (must be positive, otherwise it is not used and particle data is extrapolated to nearest grid points). The default value is -1.0.

4.6 Visualization parameters

`! Output` `output type : STRING`	Specifies the type for data output. `ensight` for EnSight Gold format, which can be read by most visualization software. Default is none.
`output frequency : DBLE`	Specifies how often (in simulation time) the data are dumped.
`dump VAR : LOGICAL`	Optional `VAR` to dump: `viscosity`, `Q criterion`, `vorticity`, `dilatation`, `schlieren`, `dissipation`, `jacobian`, `grid spacing`, `arc length`, `filtered velocity`.

4.7 Adjoint parameters

```
# Adjoint parameters ##############################################
```

`! Adjoint flags` `disable adjoint solver : LOGICAL`	Determine if the adjoint solver should be used. The default value is true (no adjoint).
`use continuous adjoint : LOGICAL`	Determine whether a continuous or discrete adjoint is used. The default value is false.
`single controlled prediction : LOGICAL` `actuation amount : DBLE`	If `single controlled prediction` is true, only one forward simulation will be performed while the control forcing terms are perturbed by `actuation amount`. The default values are false and 1, respectively.
`baseline prediction available : LOGICAL`	Specifies if the forward simulation data is available. The default value is false.
`adjoint gradient available : LOGICAL`	Specifies if the adjoint gradient is available. The default value is false.
`! Files` `adjoint file to write : STRING`	Specifies the name of the adjoint solution file to be written.

```
! Cost functional
cost functional type : STRING
```

Determine the type of the cost functional (required for adjoint simulations). Available cost functionals include `sound`, `pressure drag`, `drag`, `reynolds stress`, `temperature`, `heat release`, `reactant`, `binary mixing`, `velocity norm`, `mixing norm`, and `data assimilation`.

```
! Controller
controller type : STRING
```

Specify the type of the control forcing used. Available control forcing types include `thermal actuator`, `momentum actuator`, `fuel actuator`, `ignition actuator`, `chemical actuator`, `perturbation actuator`, and `inital condition actuator`. Each one may require additional information. There is no control forcing by default.

```
gradient buffer size : INT
```

If the control forcing term is a function of grid space and time, then the frequency of input/output of its gradient must be specified. The default value is 20.

```
! Gradient accuracy
check gradient accuracy : LOGICAL
```

Test case for determining accuracy of the gradient obtained from the adjoint solution. Compares the adjoint gradient to finite difference approximations.

```
initial actuation amount : DBLE
```

Specifies the initial step size for perturbing the control parameters (required if `check gradient accuracy:.true.`).

```
minimize cost functional : LOGICAL
```

Specifies the search direction. If true (default) parameters are adjusted by moving down the gradient, otherwise the parameters are adjusted in order to maximize the cost functional.

```
output control iterations : LOGICAL
```

If false, data are only output during the initial baseline solution. The default value is false.

```
! Optimization
find optimal forcing : LOGICAL
```

Specifies if the control forcing must be optimized (true) or not (false). The default value is false.

`optimization library : STRING`	Specifies the optimization type. The options are the `jCODE` (default), `snopt` (licensed package `SNOPT`), and `python` (the python open source package `SciPy`). Each optimization package may require their own parameters.
`optimization tolerance : DBLE`	Specifies the tolerance below which the optimization will be stopped. The default value is 10^{-6}.
`restart forward evaluation : INT`	Specifies the number of forward iterations that were previously performed (for restarting the optimizer). The default value is restart control iteration.

5 Simulation examples

5.1 General work flow

Initialization

Once the code is properly compiled, change to the directory that contains your `input` file. To perform the initialization, run the program `init_flow` by typing

```
<path-to-jcode>/bin/init_flow input
```

Relevant initialization information will be output to the screen. By default, this will create two new files: `grid`, `data.init`, unless other names are specified in the `input` file. `jCODE` requires these two files to run the simulation. If particles or the immersed boundary method is enabled, additional files will be written during initialization.

Running

To run the code on a single processor:

```
<path-to-jcode>/bin/jcode input
```

To run with `<n>` processors:

```
<path-to-mpirun> -np <n> <path-to-jcode>/bin/jcode input
```

An example run with four processors will output the following to the screen:

```
jCODE
--------------------------------
4 processes reporting for duty

Simulation is 3D
--------------------------------
Grid size :  128 x   65 x    8 points
min. Jacobian =  1.25E+04 at (   1,   33,    1)
max. Jacobian =  2.97E+04 at (   1,    1,    1)


New forward run:
--------------------------------

     0, time =  0.00000E+00, dt =  4.22841E-03
     1, time =  4.22841E-03, dt =  4.22841E-03
     2, time =  8.45528E-03, dt =  4.22687E-03
     3, time =  1.26806E-02, dt =  4.22535E-03
     4, time =  1.69052E-02, dt =  4.22453E-03
     5, time =  2.11296E-02, dt =  4.22439E-03
     6, time =  2.53540E-02, dt =  4.22446E-03
     7, time =  2.95781E-02, dt =  4.22406E-03
     8, time =  3.38012E-02, dt =  4.22313E-03
     9, time =  3.80239E-02, dt =  4.22270E-03
    10, time =  4.22469E-02, dt =  4.22302E-03
```

The number of processors and grid diagnostics are first reported. This includes the number of grid points and the min/max Jacobian (corresponding to the min/max volume element). Next, the columns correspond to

- # : number of the timestep
- Time : running time of the simulation
- dt/cfl : The minimum timestep or the maximum CFL, depending on the time stepping configuration

In addition, two directories have been created

- `monitor`: contains files to monitor the behavior of the code
- `ensight-3D`: contains files used to visualize the flow field

5.2 Sod shock tube

This case solves the one-dimensional Sod shock tube problem on a two-dimensional grid of size 401×32. Characteristic boundary conditions are imposed in direction 1 and periodic boundary conditions imposed in direction 2. In addition, an absorbing sponge zone is included on the left and right boundaries. The pre- and post-shock density and pressure are specified under the `# Initialization parameters`. Execute the following commands in the terminal to change to the appropriate directory, initialize, and run:

```
cd <path-to-jcode>/examples/sod_shock_tube
<path-to-jcode>/bin/init_flow input
<path-to-mpirun> -np 2 <path-to-jcode>/bin/jcode input
```

The density profile at $t = 0.2$ (400 time steps) should look like:

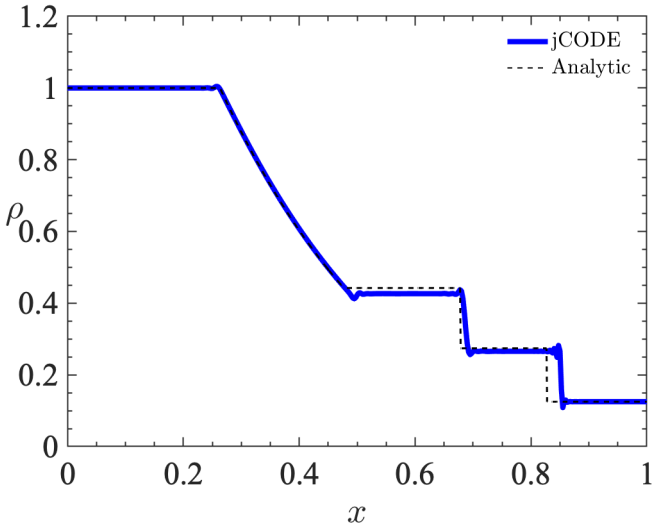


Figure 2: Solution to the Sod shock tube example at $t = 0.2$.

5.3 Particle-laden shock tube

This case considers a shock interacting with a curtain of particles. The domain is of size 513×64 with grid spacing $\Delta x = 2d_p$. The reference quantities correspond to air ($\rho^*_\infty = 1.2$ kg/m^3, $\mu^*_\infty = 1.8 \times 10^{-5}$ Pa·s, and $c^*_\infty = 343$ m/s). The reference length corresponds to the particle diameter ($d_p = 1$) with $L^* = 100$ µm. With this, the code Reynolds number is set to $\mathrm{Re}_c = 2286.7$. Particles are initially randomly distributed within a thin layer in the pre-shock region of the domain. Particle collisions are turned on in this example, and the `collision time` is set to 10 (approximately $30\Delta t$). This example can be run by executing the following commands:

```
cd <path-to-jcode>/examples/particle_shock
<path-to-jcode>/bin/init_flow input
<path-to-mpirun> -np 2 <path-to-jcode>/bin/jcode input
```

After running `init_flow` the code will write a particle file consisting of 684 particles randomly distributed within a thin layer in the middle of the domain with a mean volume fraction of 0.21. An instantaneous snapshot of particle position and Mach number at $t = 660$ (after 1360 time steps):

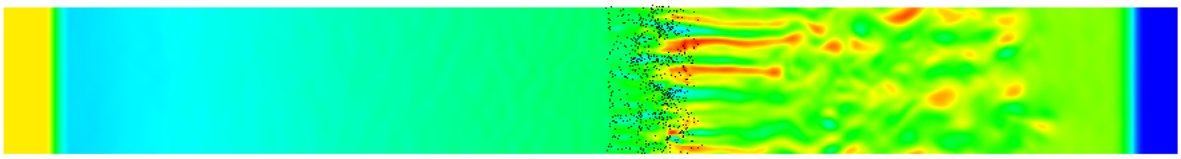


Figure 3: Particle shock interaction at $t = 660$. Particle position (black), Mach number (color).

5.4 Shock-wedge interaction using immersed boundaries

The code has capabilities to account for complex geometries using a direct forcing immersed boundary method (IBM) adapted for compressible flows. The IBM enforces appropriate boundary conditions by placing discrete markers (particles) along the surface of the geometry and projecting a source term to neighboring grid points. Placement of the markers can be handled in two ways: (i) by analytically assigning the position to each marker; or (ii) automatic generation of the markers from a stereolythography (STL) file. A utility to convert STL files to the appropriate IBM file needed by the code is located in `/bin/stl2ibm`. The file `/src/utils/init_flow/ibm_init.f90` generates several canonical geometries analytically, including cylinders, plates, and wedges.

This case solves a shock passing a triangular wedge on a two-dimensional grid of size 513×415. Characteristic boundary conditions are imposed at the left and right side of the domain, and slip-wall conditions are imposed at the top and bottom. In addition, an absorbing sponge zone is included on the left and right boundaries. 400 marker particles are placed along the sides of the wedge. It is important that there exists at least 1 marker per grid point (i.e., the average distance between marker particles should be less than Δx). The wedge position and size can be adjusted in the `input` file. This example can be run by executing the following commands:

```
cd <path-to-jcode>/examples/ibm_wedge
<path-to-jcode>/bin/init_flow input
<path-to-mpirun> -np 2 <path-to-jcode>/bin/jcode input
```

During run time, additional files are generated to monitor the progress of the IBM (`monitor/ibm` and `monitor/ibm_force`). If EnSight output is enabled, the IBM markers will automatically be dumped for visualization (as shown in blue in Fig. 4). Numerical schlieren at $t = 5$ (750 time steps) should look like:

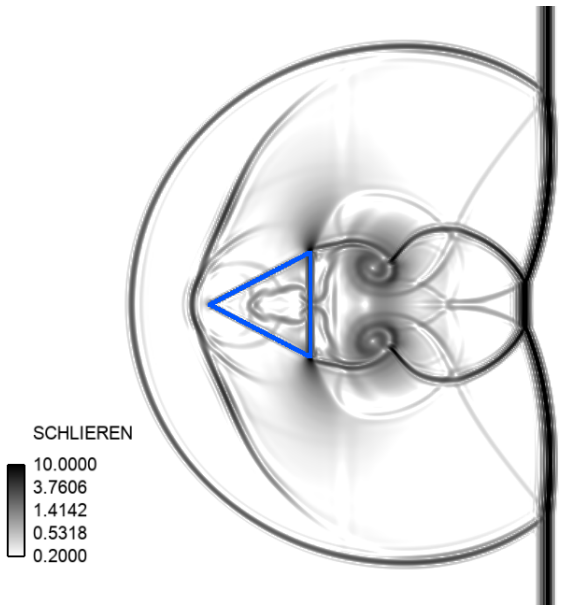


Figure 4: Numerical schlieren of a shock-wedge interaction at $t = 5$. IBM marker particles shown in blue.

5.5 Von Karman vortex street on a curvilinear grid

This case solves a flow past a cylinder on a two-dimensional grid in polar coordinates. The grid is discretized with 257 grid points in the radial direction and 129 grid points in the azimuthal direction. 'O-periodic' boundary conditions are enforced in the azimuthal direction by specifying `periodicity type in y : overlap` in the `input` file. In this example it is important to specify `curvilinear domain : .true.` in the `input` file. Execute the following commands in the terminal to change to the appropriate directory, initialize, and run:

```
cd <path-to-jcode>/examples/von_karman
<path-to-jcode>/bin/init_flow input
<path-to-mpirun> -np 2 <path-to-jcode>/bin/jcode input
```

The Mach number at $t = 68$ (3980 time steps) should look like:

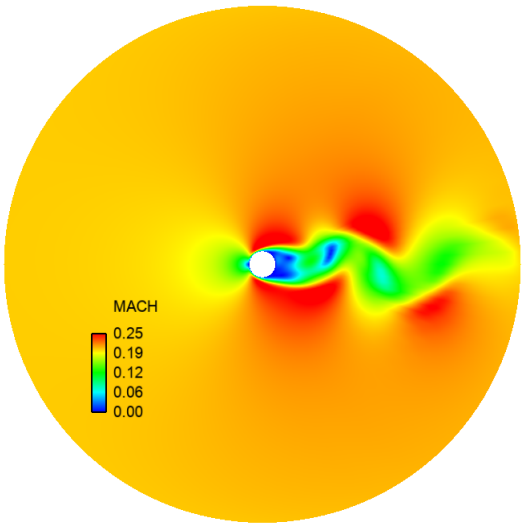


Figure 5: Von Kàrmàn vortex street using curvilinear coordinates.

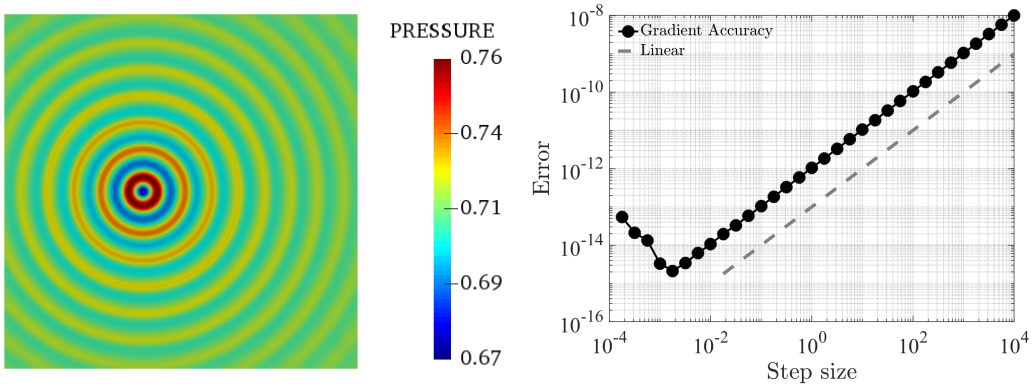
5.6 Adjoint sensitivity of an acoustic monopole

This case solves a two-dimensional flow with acoustic forcing and the corresponding adjoint sensitivity. The gradient accuracy of the adjoint sensitivity is compared with a finite difference approximation. The cost functional is defined as the time-integrated pressure fluctuations within a specified target region. A source term on the right-hand-side of the energy equation is adjusted to act as a controller. The `cost functional type` is `sound` and `controller type` is `thermal actuator`. Parameters to adjust the location and extent of the target region and control region can be found under `# Adjoint parameters` in the `input` file. Execute the following commands in the terminal to change to the appropriate directory, initialize, and run:

```
cd <path-to-jcode>/examples/acoustic_monopole
<path-to-jcode>/bin/init_flow input
<path-to-mpirun> -np 2 <path-to-jcode>/bin/jcode input
```

The code will initially perform a 'forward run', i.e. a baseline prediction with zero actuation. During this run the instantaneous and time-integrated cost functional are output to `monitor/functional`. Once the baseline run is complete, the adjoint equations are solved backward in time. During this time, sensitivity information is written to `monitor/adjoint_sensitivity` and `monitor/adjoint_timing`. In this example, the control forcing is a function of space and time, and its values are saved to `gradient_controlRegion.dat` as a binary file. Once the adjoint solution is complete, a steepest-descent is performed by running several forward runs with varying actuation amounts. This is done to perform a finite difference estimation of the sensitivity gradient. New files are written to the `monitor` directory during each iteration. The gradient

accuracy information is written to `gradient_error.txt`. This file contains the actuation amount (i.e. the finite difference step size), cost functional, cost sensitivity, and the gradient error for each iteration. The error is shown in Fig. 6b.



(a) (b)

Figure 6: (a) Pressure distribution for acoustic monopole at simulation time 24, (b) gradient accuracy of the adjoint sensitivity

Appendices

Appendix A Drag laws used in jCODE

A.1 Basic form – the drag coefficient and the superficial velocity

Let's consider fluid with viscosity μ and density ρ flowing with velocity $\boldsymbol{u}$ past a solid spherical particle with velocity $\boldsymbol{u}_p$, diameter d_p, and density ρ_p. The drag coefficient is defined as

$$C_D = \frac{\boldsymbol{f}_{\text{drag}}}{\frac{1}{2}\rho\|\boldsymbol{v}_r\|\boldsymbol{v}_r A}, \tag{25}$$

where $\boldsymbol{f}_{\text{drag}}$ is the drag force appearing in (14), $\boldsymbol{v}_r = \alpha(\boldsymbol{u} - \boldsymbol{u}_p)$ is the *superficial* relative velocity, with α the fluid volume fraction, and $A = \pi d_p^2/4$ is the frontal area of the particle. The superficial velocity attempts to approximate the local velocity as if the particle was not there. In general, the fluid velocity can be expressed as $u = Q/A$, where Q is the volume flow rate of the fluid, and A is the cross sectional area under consideration. For an incompressible flow, Q will change according to the area change. Thus, for a flow moving with velocity u through a suspension of particles with volume fraction α, the superficial velocity can be determined by equating $Q = u_1 A_1 = u_2 A_2$, where $u_1 = u$ is the actual velocity, $A_1 = \alpha A$ is the area the fluid is flowing through, $u_2 = u_s$ is the velocity we desire through $A_2 = A$, the area in the absence of particles. Thus, the superficial velocity is $u_s = \alpha u$.

The following section will define different drag models available in jCODE, which are used to update the particle position (assuming spherical particles) according to

$$\frac{\mathrm{d}\boldsymbol{u}_p}{\mathrm{d}t} = \frac{F_D}{\tau_p}\alpha\left(\boldsymbol{u} - \boldsymbol{u}_p\right) \quad \text{with} \quad F_D = \frac{\mathrm{Re}_p}{24}C_D, \tag{26}$$

where $\tau_p = \rho_p d_p^2/(18\mu)$ is the particle response time (valid in the Stokes regime), $\mathrm{Re}_p = \rho d_p\|\boldsymbol{v}_r\|/\mu$ is the particle Reynolds numbers, and $F_D = F_d(\mathrm{Re}_p, \alpha, \text{etc.})$ is a non-dimensional correction to Stokes drag.

A.2 Stokes [1]

In 1851, Sir George Gabriel Stokes derived the drag force for an isolated particle in a low Reynolds number flow, which is given by

$$\boldsymbol{F}_d^{\text{Stokes}} = 3\pi d_p \mu \boldsymbol{v}_r. \tag{27}$$

This expression can be rearranged to form an acceleration: $\boldsymbol{F}_D/m_p = \boldsymbol{v}_r/\tau_p$, where $m_p = \rho_p d_p^3/6$ is the particle mass. Using (25) and (26), this expression can be rearranged to form the drag coefficient in the Stokes limit:

$$C_D = \frac{24}{\mathrm{Re}_p} \quad \text{and thus} \quad F_D = 1. \tag{28}$$

A.3 `Schiller-Naumann` [2]

When $\mathrm{Re}_p > 1$ and the flow is sufficiently dilute (i.e., $\alpha \approx 1$), Schiller and Naumann (1933) [2] derived an expression for the drag coefficient expressed as

$$C_D = \frac{24}{\mathrm{Re}_p}\left(1 + 0.15\mathrm{Re}_p^{0.687}\right) \quad \text{and} \quad F_D = 1 + 0.15\mathrm{Re}_p^{0.687}. \tag{29}$$

A.4 `Gidaspow` [3]

Dimitri Gidaspow's 1994 textbook [3] presents an expression for the drag force that varies with both the local Reynolds number and volume fraction for applications to fluidized beds. When $\alpha \leq 0.8$ (i.e., for dense suspensions), the formulation presented by Ergun [14] is employed, whereas the formulation by Wen & Yu [15] is used when $\alpha > 0.8$ (i.e., for dilute suspensions):

$$\beta = \begin{cases} 150\frac{\alpha_p^2\mu}{\alpha d_p^2} + 1.75\frac{\rho\alpha_p \boldsymbol{v}_r}{\alpha d_p}, & \alpha \leq 0.8 \\ \frac{3}{4}C_D\frac{\rho\alpha_p \boldsymbol{v}_r}{d_p}\alpha^{-2.65}, & \alpha > 0.8, \end{cases} \tag{30}$$

with

$$C_D = \begin{cases} \frac{24}{\mathrm{Re}_p}\left(1 + 0.15\mathrm{Re}_p^{0.687}\right), & \mathrm{Re}_p < 1000 \\ 0.44, & \mathrm{Re}_p \geq 1000. \end{cases} \tag{31}$$

The interphase coefficient β is related to F_D according to

$$F_D = \frac{\tau_p \beta}{\rho_p \alpha_p}. \tag{32}$$

A.5 `Tenneti` [4]

In more recent years, particle-resolved direct numerical simulations (PR-DNS) have been used to develop improved drag laws. The drag law of Tenneti et al. [4] is valid for $0 \leq \alpha \leq 0.5$ and $0 \leq \mathrm{Re}_p \leq 300$. It reduces to `Schiller-Naumann` in the limit $\alpha \to 0$ and `Stokes` when $\mathrm{Re}_p \to 0$. The non-dimensional drag correction is given by

$$F_D = \frac{1 + 0.15\mathrm{Re}_p^{0.687}}{\alpha^2} + b_1(\alpha) + b_2(\alpha, \mathrm{Re}_p), \tag{33}$$

where

$$b_1 = \frac{5.81\alpha_p}{\alpha^2} + \frac{0.48\alpha_p^{1/3}}{\alpha^3} \quad \text{and} \quad b_2 = \alpha\alpha_p^3\mathrm{Re}_p\left(0.95 + \frac{0.61\alpha_p^3}{\alpha^2}\right). \tag{34}$$

It should be noted that there is an extra factor of α appears in this expression compared to the original paper to remove the contribution of the mean pressure gradient used in the PR-DNS.

References

[1] G. G. Stokes. On the effect of internal friction of fluids on the motion of pendulums. *Transactions of the Cambridge Philosophical Society*, 9, 1851.

[2] L. Schiller and A. Naumann. Fundamental calculations in gravitational processing. *Zeitschrift Des Vereines Deutscher Ingenieure*, 77:318–320, 1933.

[3] D. Gidaspow. *Multiphase flow and fluidization: continuum and kinetic theory descriptions*. Academic Press, 1994.

[4] S. Tenneti, R. Garg, and S. Subramaniam. Drag law for monodisperse gas–solid systems using particle-resolved direct numerical simulation of flow past fixed assemblies of spheres. *International Journal of Multiphase Flow*, 37(9):1072–1092, 2011.

[5] R. Vishnampet Ganapathi Subramanian. *An exact and consistent adjoint method for high-fidelity discretization of the compressible flow equations*. PhD thesis, University of Illinois at Urbana-Champaign, 2015.

[6] B. Strand. Summation by parts for finite difference approximations for d/dx. *Journal of Computational Physics*, 110(1):47–67, 1994.

[7] K. Mattsson, M. Svärd, and J. Nordström. Stable and accurate artificial dissipation. *Journal of Scientific Computing*, 21(1):57–79, 2004.

[8] M. H. Carpenter, D. Gottlieb, and S. Abarbanel. Time-stable boundary conditions for finite-difference schemes solving hyperbolic systems: Methodology and application to high-order compact schemes. *Journal of Computational Physics*, 111(2):220–236, 1994.

[9] S. Pirozzoli. Stabilized non-dissipative approximations of euler equations in generalized curvilinear coordinates. *Journal of Computational Physics*, 230(8):2997–3014, 2011.

[10] J. Capecelatro and O. Desjardins. An euler–lagrange strategy for simulating particle-laden flows. *Journal of Computational Physics*, 238:1–31, 2013.

[11] G. S. Shallcross, R. O. Fox, and J. Capecelatro. A volume-filtered description of compressible particle-laden flows. *International Journal of Multiphase Flow*, 122:103138, 2020.

[12] S. Gottlieb and C.-W. Shu. Total variation diminishing Runge-Kutta schemes. *Mathematics of computation*, 67(221):73–85, 1998.

[13] J. Capecelatro, D. J. Bodony, and J. B. Freund. Adjoint-based sensitivity and ignition threshold mapping in a turbulent mixing layer. *Combustion Theory and Modelling*, pages 1–33, 2018.

[14] S. Ergun. Fluid flow through packed columns. *Chemical Engineering and Processing*, 48:89–94, 1952.

[15] C. Y. Wen and Y. H. Yu. Mechanics of fluidization. *Chemical Engineering Progress Symposium Series*, 62:100–105, 1966.