可压缩流动数值方法

针对 *Sod* 激波管问题求解一维欧拉方程

$$\frac{\partial \mathbf{U}}{\partial t}+\frac{\partial \mathbf{F}(\mathbf{U})}{\partial x}=0 \tag{1}$$

其中

$$\mathbf{F}=[\rho, \rho u, E]^{T}, \tag{2}$$

$$\mathbf{F}(\mathbf{U})=\left[\rho u, \rho u^{2}+p, u(E+p)\right]^{T}, \tag{3}$$

$$E=\rho e=\rho\left(C_{v} T+\frac{1}{2}\right) \tag{4}$$

$t=0$ 时刻初始条件为：$x<0$ 处，$(u_L, \rho_L, p_L)=(0,1,1)$；$x \geq 0$ 处，$(u_R, \rho_R, p_R)=(0,0.125,0.1)$。

请采用本课程介绍的数值方法求解。要求：

- 请比较密度、速度、压力的分布，并与精确解进行比较；
- *Sod* 问题的 *Riemann* 精确解可查询网络、参考书得到，报告中直接给出精确解的方程形式即可；
- 计算域、计算网格自选，请进行相关讨论；
- 请至少选择两种方法处理数值通量，例如：*Van Leer* 流通矢量分裂，*Roe* 格式等；
- 自选激波捕捉格式，如：*TVD, NND, WENO* 等；
- 时间推进格式选用 *3* 阶 *Runge-Kutta*；
- 附加题：请尝试不同精度的差分格式，并讨论其影响。

问题解答

如图 **1** 所示，题设所给 Sod 激波管问题，是典型的一维无粘流动初始间断的演化问题，即一维 Riemann 问题。

作业 4

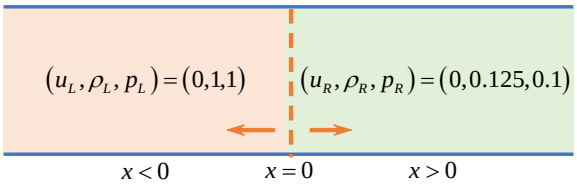


图 1 一维 Sod 激波管 Riemann 问题

对于典型的 Riemann 问题，一般有以下两种求解思路：

1) **精确解**：利用空气动力学知识（积分关系式 + 特征线方法）；

2) **近似解**：利用积分近似或微分近似方法。

由于本文旨在通过直接求解一维 Euler 方程以获得 Sod 激波管问题的解，故下面直接给出一维 Riemann 问题的精确解的求解思路与方程，以供后续的数值实验参考。

一维 Riemann 问题精确解

根据空气动力学知识，该 Sod 激波管中可能出现三种波：1) 激波。经过激波，流体的密度、速度、压力均发生突变，且满足 Rankine-Hugoniot (R-H) 关系式；2) 接触间断。经过接触间断，流体仅密度发生突变，速度与压力不变；3) 膨胀波（稀疏波）。一种等熵波，其内部物理量连续、光滑，头、尾物理量连续但导数不连续（弱间断）且 Riemann 不变量不变。

考虑一般情况，管中有五种组合波的可能性。根据质量通量、动量通量、能量通量守恒的条件，取随激波运动、厚度充分小的控制体，可根据以下分类讨论列写方程并求解。

1) **情况 1)**：左激波、右激波。可分区列写积分关系式如下：

（1 - 3 两区，激波 R-H 关系式）

$$\begin{cases} \rho_1 (u_1 - Z_1) = \rho^{*L} (u^* - Z_1) \\ \rho_1 u_1 (u_1 - Z_1) + p_1 = \rho^{*L} u^* (u^* - Z_1) + p^* \\ E_1 (u_1 - Z_1) + u_1 p_1 = E^{*L} (u^* - Z_1) + p^* u^* \end{cases} \tag{5}$$

（2 - 4 两区，激波 R-H 关系式）

$$\begin{cases} \rho_2 (u_2 - Z_2) = \rho^{*R} (u^* - Z_2) \\ \rho_2 u_2 (u_2 - Z_2) + p_2 = \rho^{*R} u^* (u^* - Z_2) + p^* \\ E_2 (u_2 - Z_2) + u_2 p_2 = E^{*R} (u^* - Z_2) + p^* u^* \end{cases} \tag{6}$$

其中：

$$E_k = \frac{p_k}{(\gamma - 1)} + \rho_k \frac{u_k^2}{2} \quad k = 1, 2, L, R \tag{7}$$

以上变量说明从略。综上 6 个方程、6 个未知数，故方程组可解，求解可使用消元法。联立以上两个方程组，可得

$$u_1 - u_2 = f(p^*, p_1, \rho_1) + f(p^*, p_2, \rho_2) \tag{8}$$

上述一元一次方程可采用 Newton 法求解。解出 p^* 后，代入原始方程组，即可以求出其余的未知数。

2) **情况 2**)：右激波、左膨胀波（注：一维 Sod 激波管实际起动后流场演化即属于该情况）。可分区列写关系式如下：

（1 - 3 两区，等熵关系式）

$$p^* / \left(\rho^{*L}\right)^\gamma = p_1 / (\rho_1)^\gamma \tag{9}$$

$$u_1 + \frac{2c_1}{\gamma - 1} = u^* + \frac{2c^L}{\gamma - 1} \tag{10}$$

其中，$c^L = \sqrt{\gamma p^* / \rho^{*L}}$。

（2 - 4 两区，激波 R-H 关系式）

$$\begin{cases} \rho_2 (u_2 - Z_2) = \rho^{*R} (u^* - Z_2) \\ \rho_2 u_2 (u_2 - Z_2) + p_2 = \rho^{*R} u^* (u^* - Z_2) + p^* \\ E_2 (u_2 - Z_2) + u_2 p_2 = E^{*R} (u^* - Z_2) + p^* u^* \end{cases} \tag{11}$$

以上变量说明从略。综上 5 个方程、5 个未知数，故方程组可解，求解方法与左右双激

波情况类似。联立以上两个方程组，解出 3、4 区内速度对压力的依赖关系，有

$$u^* = u_1 - f\left(p^*, p_1, \rho_1\right) \tag{12}$$

其中，满足 $f\left(p^*, p_i, \rho_i\right) = \frac{2c_i}{\gamma-1}\left[\left(\frac{p^*}{p_i}\right)^{\frac{\gamma-1}{2\gamma}} - 1\right]$。

注意到，激波、膨胀波前后速度-压力的依赖关系可写成统一的形式：

左波（激波或膨胀波）

$$u^* = u_1 - f\left(p^*, p_1, \rho_1\right) \tag{13}$$

右波（激波或膨胀波）

$$u^* = u_2 + f\left(p^*, p_2, \rho_2\right) \tag{14}$$

以上 u^*, p^* 表示 3、4 区内的速度与压力，其中

$$f\left(p^*, p_i, \rho_i\right) = \begin{cases} \frac{p^* - p_i}{\rho_i c_i \left[\frac{\gamma+1}{2\gamma}\left(\frac{p^*}{p_i}\right) + \frac{\gamma-1}{2\gamma}\right]^{1/2}}, & p^* > p_i \\ \frac{2c_i}{\gamma-1}\left[\left(\frac{p^*}{p_i}\right)^{\frac{\gamma-1}{2\gamma}} - 1\right], & p^* < p_i \end{cases} \tag{15}$$

求解上式可得到 3、4 区内的压力，然后可以解得速度和密度。

3) 对于**膨胀波内部**物理量的计算，首先由波头传播速度 $u_1 - c_1$ 与波尾传播速度 $u^* - c^{*L}$ 可计算膨胀波的范围。在膨胀波区内，利用特征相容关系和等熵关系计算物理量，可利用简单波的特性来简化计算。以下直接给出各个物理量的计算表达式：

$$c(t,x) = \frac{\gamma-1}{\gamma+1}\left(u_1 - \frac{x}{t}\right) + \frac{2}{\gamma+1}c_1 \tag{16}$$

$$u(x,t) = c + x/t \tag{17}$$

$$p = p_1\left(c/c_1\right)^{2\gamma/\gamma-1} \tag{18}$$

$$\rho = \gamma p / c^2 \tag{19}$$

以上求解步骤完全适用于另外的组合波情况，说明从略。区分组合波的情况，可由激波后压力升高，膨胀波后压力降低来进行判断。如仅需要判断左、右波的性质，无需求解

$F(p^*) = u_1 - u_2$，在计算得到 p^* 后，利用 $F(p^*)$ 函数的单调递增特性即可。

综上所述，一维 Riemann 问题的精确解的求解思路与方程介绍完毕。本文 Sod 激波管参考精确解程序来自于 *MATLAB* 官方开源文档 (Gogol, 2021) [1]，详细参考代码见**附录 B**。

由于精确 Riemann 解的计算量非常大，故实际常用近似 Riemann 解。值得注意的是，精确 Riemann 解的效果也未必比近似 Riemann 解好。近似 Riemann 解主要包括积分型 (Harten, Lax & van Leer (HLL), HLLC (Toro 发展的三波模型))、微分型 (Roe)，其中 Roe 格式是目前运用最广泛的近似 Riemann 解，将在下文详细介绍。

流通矢量分裂技术

目前，数值求解守恒型求解一维 Euler 方程采用的的常用技术主要包括流通矢量分裂 (Flux Vector Splitting, FVS) 与流通差分分裂 (Flux Vector Splitting, FVS) 两类。流通矢量分裂方法，是依据利用特征线方法可以将方程的解表示成特征向量的组合，同时结合特征解的物理背景而提出的。此方法将方程中代表质量、动量和能量的流通矢量按照雅克比 (Jacobian) 矩阵的特征值进行分裂，然后按分裂后的流通质量构造迎风类型的格式。其优势是稳定性好、低振荡。流通差分分裂方法的特点在于对流通矢量的导数进行分裂，首先运用差分格式构造半点处的左右守恒变量，再利用各类近似 Riemann 解构造半点流通矢量。

1) **流通矢量分裂技术 (FVS)**

下面简述流通矢量分裂技术的基本流程。首先介绍 **Jacobian 系数矩阵及其性质**。

首先将原始守恒变量 U 与流通矢量 $F(U)$ 写成如下形式

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} \tag{20}$$

$$\mathbf{F}(U) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ u(E+p) \end{pmatrix} = \begin{pmatrix} w_2 \\ \frac{w_2^2}{w_1} + p \\ \frac{w_2}{w_1}(w_3 + p) \end{pmatrix} \tag{21}$$

注意到总能量 E 满足

$$E = \rho e = \rho\left(c_v T + \frac{1}{2}u^2\right) \tag{22}$$

考虑完全气体状态方程 $p = \rho RT$，压力 p 可进一步表示为

$$p = (\gamma - 1)\left(E - \frac{1}{2}\rho u^2\right) = (\gamma - 1)\left(w_3 - \frac{1}{2}\frac{w_2^2}{w_1}\right) \tag{23}$$

利用上式，将流通矢量 $\mathbf{F}(\mathbf{U})$ 可进一步整理为

$$\mathbf{F}(\mathbf{U}) = \begin{pmatrix} w_2 \\ \frac{w_2^2}{w_1} + p \\ \frac{w_2}{w_1}(w_3 + p) \end{pmatrix} = \begin{pmatrix} w_2 \\ \frac{3-\gamma}{2}\frac{w_2^2}{w_1} + (\gamma - 1)w_3 \\ \frac{\gamma w_2 w_3}{w_1} - \frac{\gamma - 1}{2}\frac{w_2^3}{w_1^2} \end{pmatrix} \tag{24}$$

通过微分运算，容易得到以下关系式

$$\begin{cases} dw_1 = d\rho \\ dw_2 = u d\rho + \rho du \\ dw_3 = \frac{dp}{\gamma - 1} + \frac{1}{2}u^2 d\rho + \rho u du \end{cases} \tag{25}$$

根据通量表达式，可以得到 Jacobian 矩阵 $\mathbf{A}$ 为

$$\mathbf{A}(\mathbf{U}) = \frac{\partial \mathbf{F}(U)}{\partial \mathbf{U}} = \begin{pmatrix} \frac{\partial F_1}{\partial U_1} & \frac{\partial F_1}{\partial U_2} & \frac{\partial F_1}{\partial U_3} \\ \frac{\partial F_2}{\partial U_1} & \frac{\partial F_2}{\partial U_2} & \frac{\partial F_2}{\partial U_3} \\ \frac{\partial F_3}{\partial U_1} & \frac{\partial F_3}{\partial U_2} & \frac{\partial F_3}{\partial U_3} \end{pmatrix} \tag{26}$$

将相关变量代入上式矩阵，可计算得到矩阵 $\mathbf{A}$ 为

$$\mathbf{A}(\mathbf{U}) = \begin{pmatrix} 0 & 1 & 0 \\ \frac{\gamma - 3}{2}\frac{w_2^2}{w_1^2} & (3 - \gamma)\frac{w_2}{w_1} & \gamma - 1 \\ -\frac{\gamma w_2 w_3}{w_1^2} + (\gamma - 1)\frac{w_2^3}{w_1^3} & \frac{\gamma w_3}{w_1} - \frac{3(\gamma - 1)}{2}\frac{w_2^2}{w_1^2} & \frac{\gamma w_2}{w_1} \end{pmatrix} \tag{27}$$

该矩阵也可以用原始变量表示，有

$$\mathbf{A}(\mathbf{U}) = \begin{pmatrix} 0 & 1 & 0 \\ \frac{\gamma-3}{2}u^2 & (3-\gamma)u & \gamma-1 \\ -\frac{\gamma p u^2}{(\gamma-1)\rho} + \frac{\gamma-2}{2}u^3 & \frac{\gamma p}{(\gamma-1)\rho} - \left(\gamma - \frac{3}{2}\right)u^2 & \gamma u \end{pmatrix} \tag{28}$$

进一步，可以计算矩阵 $\mathbf{A}$ 的特征值。引入声速 c，其满足

$$c^2 = \frac{\gamma p}{\rho} = \gamma(\gamma-1)\left(\frac{w_3}{w_1} - \frac{1}{2}\frac{w_2^2}{w_1^2}\right) \tag{29}$$

则矩阵 $\mathbf{A}$ 的特征方程可写为

$$|\mathbf{A} - \lambda\mathbf{I}| = \begin{pmatrix} -\lambda & 1 & 0 \\ \frac{\gamma-3}{2}u^2 & (3-\gamma)u - \lambda & \gamma-1 \\ -\frac{\gamma p u^2}{(\gamma-1)\rho} + \frac{\gamma-2}{2}u^3 & \frac{\gamma p}{(\gamma-1)\rho} - \left(\gamma - \frac{3}{2}\right)u^2 & \gamma u - \lambda \end{pmatrix} \tag{30}$$

$$\begin{aligned} &= -\lambda^3 + 3u\lambda^2 - \left(3u^2 - c^2\right)\lambda + \left(u^3 - uc^2\right) \\ &= -(\lambda - u)(\lambda - (u+c))(\lambda - (u-c)) = 0 \end{aligned}$$

故可得 Jacobian 矩阵 $\mathbf{A}$ 的特征值为

$$\lambda_1 = u - c, \quad \lambda_2 = u, \quad \lambda_3 = u + c. \tag{31}$$

在构造耗散型格式时，对简单波方程可根据系数的符号来确定对空间导数采用向前差分或向后差分，但对于方程组，则需要事先对流通矢量进行分裂。常用方法主要包括 Steger-Warming (S-W) 分裂方法 [2]、Lax-Friedrichs (L-F) 分裂方法、van Leer 分裂方法 [3] 以及 Liou-Steffen (Advection Upstream Splitting Method, AUSM) 分裂系列方法 [4] 等。

首先阐述以特征值矩阵分裂为核心的 FVS 方法，其代表是 S-W、L-F 分裂方法。

不难验证，当状态方程有如下形式时：

$$p = \rho g(e) \tag{32}$$

则流通矢量 $\mathbf{F}(\mathbf{U})$ 为守恒矢量 $\mathbf{U}$ 的一次齐次函数，即关系式

$$\mathbf{F}(\alpha\mathbf{U}) = \alpha\mathbf{F}(\mathbf{U}) \tag{33}$$

对任意参数 α 都成立。对上式两侧都取对 α 的导数，再令 $\alpha = 1$，则可得

$$\mathbf{F}(\mathbf{U}) = \mathbf{A}\mathbf{U} \tag{34}$$

同时，利用这一性质，易知

$$\frac{\partial(\mathbf{A}\mathbf{U})}{\partial x} = \mathbf{A}\frac{\partial\mathbf{U}}{\partial x}, \quad \frac{\partial(\mathbf{A}\mathbf{U})}{\partial t} = \mathbf{A}\frac{\partial\mathbf{U}}{\partial t}, \tag{35}$$

从形式上看，这里 Jacobian 矩阵 $\mathbf{A}$ 如同常数矩阵，可以提到偏微分号外边来。

现在，进一步讨论关于流通矢量 $\mathbf{F}(\mathbf{U})$ 的分裂。对单波方程可对流通量进行如下分裂：

$$f^{\pm} = c^{\pm}u, \quad c^{\pm} = \frac{c \pm |c|}{2} \tag{36}$$

满足

$$f = f^{+} + f^{-}, \quad c = c^{+} + c^{-} \tag{37}$$

然后按照分裂后 $c^{\pm}$ 的符号来构造相应的耗散型格式。对流通矢量 $\mathbf{F}(\mathbf{U})$，可通过对 Jacobian 系数矩阵的特征分裂来完成类似的分裂过程。设 λ_k 为 Jacobian 矩阵 $\mathbf{A}$ 的特征值，λ_k^+ 和 λ_k^- 为分裂后的特征值，要求

$$\lambda_k = \lambda_k^+ + \lambda_k^- \tag{38}$$

其中，$\lambda_k^+ \geq 0$, $\lambda_k^- \leq 0$。由分裂后的特征值组成的对角矩阵 (diagonal matrix) 为

$$\Lambda^{\pm} = \begin{bmatrix} \lambda_1^{\pm} & 0 & 0 \\ 0 & \lambda_1^{\pm} & 0 \\ 0 & 0 & \lambda_1^{\pm} \end{bmatrix} \tag{39}$$

利用对角矩阵 $\Lambda^{\pm}$ 可定义出相应于矩阵 $A^{\pm}$ 和分裂后的流通矢量 $\mathbf{F}^{\pm}$ 如下

$$\mathbf{A}^{\pm} = \mathbf{S}^{-1}\,{}^{\pm}\mathbf{S}, \quad \mathbf{F}^{\pm} = \mathbf{A}^{\pm}\mathbf{U} \tag{40}$$

其中，满足 $\mathbf{A} = \mathbf{A}^{+} + \mathbf{A}^{-}$, $\mathbf{F} = \mathbf{F}^{+} + \mathbf{F}^{-}$。

对于特征值分裂，有多种方式，简述如下：

a) **Steger-Warming (S-W) 分裂法**

$$\lambda_k^{\pm} = \frac{\lambda_k \pm |\lambda_k|}{2} \tag{41}$$

b) **简单分裂法**

$$\lambda_k^{+} = \lambda^{+} \geq max0, \lambda_k \tag{42}$$

$$\lambda_k^{-} = \lambda_k - \lambda^{*} \tag{43}$$

c) **Lax-Friedrichs (L-F) 分裂法**

$$\lambda_k^{\pm} = \frac{\lambda_k \pm \lambda^{*}}{2} \tag{44}$$

利用分裂后的特征值可构造对应的 Jacobian 系数矩阵，由此可得分裂后的流通矢量。Steger 等利用关系式 $\mathbf{F}^{\pm} = \mathbf{S}^{-1}\,{}^{\pm}\mathbf{SU}$，给出以下 $\mathbf{F}^{\pm}$ 的具体表达形式

$$\mathbf{F}(\widetilde{\lambda}) = \frac{\rho}{2\gamma}\begin{bmatrix} 2(\gamma-1)\widetilde{\lambda}_1 + \widetilde{\lambda}_2 + \widetilde{\lambda}_3 \\ 2(\gamma-1)\widetilde{\lambda}_1 u + \widetilde{\lambda}_2(u-c) + \widetilde{\lambda}_3(u+c) \\ (\gamma-1)\widetilde{\lambda}_1 u^2 + \frac{\widetilde{\lambda}_2}{2}(u-c)^2 + \frac{\widetilde{\lambda}_3}{2}(u+c)^2 + w \end{bmatrix} \tag{45}$$

这里

$$w = \frac{(3-\gamma)\left(\widetilde{\lambda}_2 + \widetilde{\lambda}_3\right)c^2}{2\,(\gamma-1)} \tag{46}$$

在以上两式中，若取 $\widetilde{\lambda}_k$ 等于矩阵 $\mathbf{A}$ 的特征值，则可得到原始流通矢量 $\mathbf{F}$；若取 $\widetilde{\lambda}_k = \lambda_k^{\pm}$，则可得到分裂后的流通矢量 $\mathbf{F}^{\pm}$。

现对上述流通矢量分裂方法做简单讨论。第一种 Steger-Warming 分裂法是按照物理上的特征走向进行分裂的，且在解的光滑区具有较小的截断误差。应指出，未经分裂的特征值 $\lambda_k(\mathbf{U})$ 是守恒变量 $\mathbf{U}$ 的连续函数，而分裂后的特征函数 $\lambda_k^{\pm}$ 的零点（如在声速线上）可能出现奇点。这有时会给计算带来困难，例如，在无黏流占主导的声速线附近可能有数值振荡产生。为克服在奇点附近数值振荡现象，可对特征函数的分裂做如下修正：

$$\lambda_k^{\pm} = \frac{\lambda_k \pm (\lambda_k^2 + \varepsilon^2)^{\frac{1}{2}}}{2} \tag{47}$$

按第二种简单分裂法可使得计算大大简化，不难验证

$$\mathbf{F}^+ = \lambda^* \mathbf{U}, \quad \mathbf{F}^- = \mathbf{F} - \lambda_k^+ \mathbf{U} \tag{48}$$

在采用第三种 Lax-Friedrichs 分裂法时有

$$\mathbf{F}^+ = \frac{1}{2}\left(\mathbf{F} + \lambda^* \mathbf{U}\right), \quad \mathbf{F}^- = \frac{1}{2}\left(\mathbf{F} - \lambda^* \mathbf{U}\right) \tag{49}$$

L-F 方法具有形式上的对称性，可避免引入新的奇点，在数学上光滑性质比 S-W 方法更好，但同时也增加了光滑区截断误差，导致耗散偏大。可采用局部分裂，在每个点上计算 $\lambda^* = |u| + c$；或进行全局分裂，即在整个计算域中取 $\lambda^* = max(|u| + c)$。

下面介绍基于马赫数 Ma 分裂的 van Leer 分裂法与 Liou-Steffen 的 AUSM 分裂法。

a) **van Leer 分裂法**

该分裂方法以 Jacobian 矩阵的特征值分裂为基础。首先，若 $|\mathrm{Ma}| \leq 1$，可按上风情况处理。当 $\mathrm{Ma} \geq 1$，可将流通矢量 $\mathbf{F}$ 分裂为

$$\mathbf{F}^+ = \mathbf{F}, \quad \mathbf{F}^- = 0 \tag{50}$$

当 $\mathrm{Ma} \leq -1$，有

$$\mathbf{F}^+ = 0, \quad \mathbf{F}^- = \mathbf{F} \tag{51}$$

若 $|\mathrm{Ma}| \leq 1$ 时，可按特征马赫数 Ma 进行分裂。将流通矢量 $\mathbf{F}(\mathbf{U})$ 表示成密度 ρ、声

速 c 以及马赫数 $\mathrm{Ma}=\frac{u}{a}$ 的函数，即有

$$\mathbf{F}=\mathbf{F}(\rho,c,\mathrm{Ma})=\begin{bmatrix}\rho c\,\mathrm{Ma}\\ \rho c^2\left(\mathrm{Ma}^2+\frac{1}{\gamma}\right)\\ \rho c^3\,\mathrm{Ma}\left(\frac{1}{2}\,\mathrm{Ma}^2+\frac{1}{\gamma-1}\right)\end{bmatrix}\equiv\begin{bmatrix}f_{\mathrm{mas}}\\ f_{\mathrm{mom}}\\ f_{\mathrm{ene}}\end{bmatrix}\tag{52}$$

其中，质量通量 f_{mas} 可利用 Ma 的二次函数分裂为

$$f^{+}_{\mathrm{mas}}=\frac{1}{4}\rho c(1+\mathrm{Ma})^2,\quad f^{-}_{\mathrm{mas}}=-\frac{1}{4}\rho c(1-\mathrm{Ma})^2\tag{53}$$

类似地，动量通量 f_{mom} 可分裂为

$$f^{+}_{\mathrm{mom}}=f^{+}_{\mathrm{mas}}\frac{2c}{\gamma}\left[\frac{(\gamma-1)}{2}\,\mathrm{Ma}+1\right],\quad f^{-}_{\mathrm{mom}}=f^{-}_{\mathrm{mas}}\frac{2c}{\gamma}\left[\frac{(\gamma-1)}{2}\,\mathrm{Ma}-1\right]\tag{54}$$

能量通量 f_{ene} 可分裂为

$$f^{+}_{\mathrm{ene}}=\frac{\gamma^2}{2\left(\gamma^2-1\right)}\frac{\left[f^{+}_{\mathrm{mom}}\right]^2}{f^{+}_{\mathrm{mas}}},\quad f^{-}_{\mathrm{ene}}=\frac{\gamma^2}{2\left(\gamma^2-1\right)}\frac{\left[f^{-}_{\mathrm{mom}}\right]^2}{f^{-}_{\mathrm{mas}}}\tag{55}$$

将以上分裂后的通量整理为统一列矢量形式为

$$\mathbf{F}^{\pm}=\pm\frac{1}{4}\rho a(1\pm\mathrm{Ma})^2\begin{bmatrix}1\\ \frac{2c}{\gamma}\left(\frac{\gamma-1}{2}\,\mathrm{Ma}\pm1\right)\\ \frac{2c^2}{\gamma^2-1}\left(\frac{\gamma-1}{2}\,\mathrm{Ma}\pm1\right)^2\end{bmatrix}\tag{56}$$

b) **Liou-Steffen (AUSM) 分裂法**

由 Liou 与 Steffen 发展的 AUSM 系列分裂方法，其核心思想在于将流通矢量 $\mathbf{F}$ 分解为 **对流部分**矢量 $\mathbf{F}^{(c)}$ 与 **压力部分**矢量 $\mathbf{F}^{(p)}$，即有

$$\mathbf{F}(\mathbf{U})=\begin{bmatrix}\rho u\\ \rho u^2+p\\ u(E+p)\end{bmatrix}=\begin{bmatrix}\rho u\\ \rho u^2\\ \rho uH\end{bmatrix}+\begin{bmatrix}0\\ p\\ 0\end{bmatrix}\equiv\mathbf{F}^{(c)}+\mathbf{F}^{(p)}\tag{57}$$

通过引入马赫数 Ma 与比焓 H

$$\mathrm{Ma} = \frac{u}{c}, \quad H = \frac{E+p}{\rho} \tag{58}$$

可将对流部分矢量写为

$$\mathbf{F}^{(c)} = \mathrm{Ma} \begin{bmatrix} \rho a \\ \rho a u \\ \rho a H \end{bmatrix} \equiv \mathrm{Ma}\, \hat{\mathbf{F}}^{(c)} \tag{59}$$

将格点间数值通量 $\mathbf{F}_{j+\frac{1}{2}}$ 写为

$$\mathbf{F}_{j+\frac{1}{2}} = \mathbf{F}^{(c)}_{j+\frac{1}{2}} + \mathbf{F}^{(c)}_{j+\frac{1}{2}} \tag{60}$$

其中对流通量部分可写为

$$\mathbf{F}^{(c)}_{j+\frac{1}{2}} = \mathrm{Ma}_{j+\frac{1}{2}} \left[\hat{\mathbf{F}}^{(c)}\right]_{j+\frac{1}{2}} \tag{61}$$

有定义

$$[\bullet]_{j+\frac{1}{2}} = \begin{cases} [\bullet]_j & \text{if } \mathrm{Ma}_{j+\frac{1}{2}} \geq 0 \\ [\bullet]_{j+1} & \text{if } \mathrm{Ma}_{j+\frac{1}{2}} \leq 0 \end{cases} \tag{62}$$

注意到流通矢量的对流部分，即 $\mathbf{F}^{(c)}_{j+\frac{1}{2}}$ 的迎风方向由对流的符号确定，同时速度由格点间马赫数 $\mathrm{Ma}_{j+\frac{1}{2}}$ 显示。因此，Liou 与 Steffen 将这种分裂方法命名为 Advection Upstream Splitting Method (AUSM)，意为 **“对流迎风分裂法”**，并在后续延伸发展出多种新格式，如 AUSM+ 等，构成 AUSM 系列格式。

格点间马赫数与压力项可以分裂为

$$\mathrm{Ma}_{j+\frac{1}{2}} = \mathrm{Ma}^+_j + \mathrm{Ma}^-_{j+1} \tag{63}$$

$$p_{j+\frac{1}{2}} = p^+_j + p^-_{j+1} \tag{64}$$

其中，对于 Ma 分裂，Liou 与 Steffen 参照 van Leer 分裂法，即有

$$\mathrm{Ma}^{\pm}=\begin{cases}\pm\frac{1}{4}(\mathrm{Ma}\pm1)^2 & \text{if } |\mathrm{Ma}|\le 1\\ \frac{1}{2}(\mathrm{Ma}\pm|\mathrm{Ma}|) & \text{if } |\mathrm{Ma}|>1\end{cases}\tag{65}$$

对于压力项分裂，Liou 与 Steffen 提供了两种方法

$$p^{\pm}=\begin{cases}\frac{1}{2}p(1\pm\mathrm{Ma}) & \text{if } |\mathrm{Ma}|\le 1,\\ \frac{1}{2}p\frac{(\mathrm{Ma}\pm|\mathrm{Ma}|)}{\mathrm{Ma}} & \text{if } |\mathrm{Ma}|>1.\end{cases}\tag{66}$$

或

$$p^{\pm}=\begin{cases}\frac{1}{4}p(\mathrm{Ma}\pm1)^2(2\mp\mathrm{Ma}) & \text{if } |\mathrm{Ma}|\le 1,\\ \frac{1}{2}p\frac{(\mathrm{Ma}\pm|\mathrm{Ma}|)}{\mathrm{Ma}} & \text{if } |\mathrm{Ma}|>1.\end{cases}\tag{67}$$

关于 AUSM 分裂方法的更多详细资料可以参考 Liou 与 Steffen 的原始文献 [4]。

至此，流通矢量分裂技术 (FVS) 的基本流程介绍完毕。FVS 的特点在于其稳定性好，数值振荡较小。同时相比以下介绍的 FDS 类方法，计算量更小，目前在工程中广泛使用。

2) **流通差分分裂技术 (FDS)**

下面简述流通差分分裂技术的基本流程，其可分为以下三个步骤：

a) 如图 **2** 所示，运用各类差分格式，计算在格点间同一位置 $(j+\frac{1}{2})$ 的左右数值守恒通量 $\mathbf{U}^L_{j+\frac{1}{2}}$ 与 $\mathbf{U}^R_{j+\frac{1}{2}}$；

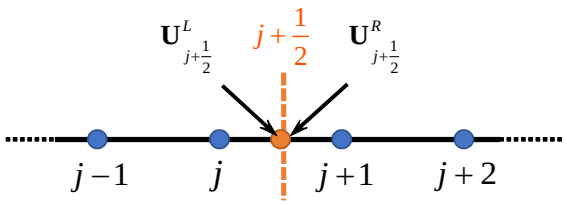


图 2 FDS 格式计算格点间左右数值守恒通量

b) 运用近似 Riemann 解，计算 $\mathbf{F}_{j+\frac{1}{2}}=\mathbf{F}\left(\mathbf{U}^L_{j+\frac{1}{2}},\mathbf{U}^R_{j+\frac{1}{2}}\right)$；

c) 进行差分重构，有 $\left(\frac{\partial\mathbf{F}}{\partial x}\right)_j=\frac{\mathbf{F}_{j+\frac{1}{2}}-\mathbf{F}_{j-\frac{1}{2}}}{\Delta x}$。

下面介绍在 FDS 类方法中应用最广泛的近似 Riemann 解 **Roe 格式** [5]。

首先介绍单方程的 Roe 格式。对于一般非线性单波方程

$$\frac{\partial u}{\partial t}+\frac{\partial f(u)}{\partial x}=0 \tag{68}$$

可化为

$$\frac{\partial u}{\partial t}+a(u)\frac{u}{\partial x}=0 \tag{69}$$

其中，$a(u)=\frac{\partial f(u)}{\partial u}$ 为非线性项。下面将该项线性化，主要目标是在 $[j,\ j+1]$ 区间内，采用平均变化率代替变化的 $a(u)$。如图 **3** 所示，根据 Lagrange 中值定理，在 $[u_L,\ u_R]$ 内必有一点 u_{Roe}，该点的斜率为区间平均变化率，同时注意到，若 $f(u)$ 为二次函数，则有 $u_{Roe}=(u_L+u_R)/2$ 成立。u_{Roe} 可由 Roe 公式计算，将在下文介绍。

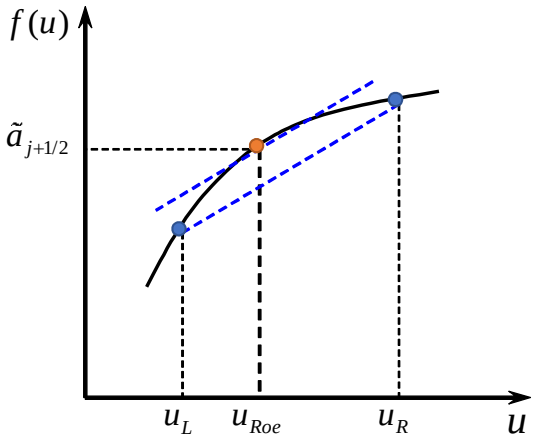


图 3　确定代替区间平均变化率的 u_{Roe} 点位置

故可构造平均变化率

$$\widetilde{a}_{j+\frac{1}{2}}=f\left(u_{Roe}\right)=\frac{f\left(u_{j+1}\right)-f\left(u_j\right)}{u_{j+1}-u_j} \tag{70}$$

在简化情形下，考虑一阶迎风差分，可根据 $\widetilde{a}_{j+\frac{1}{2}}$ 的符号构造 $j+\frac{1}{2}$ 处的数值通量如下

$$f_{j+\frac{1}{2}}=\begin{cases} f_j & \widetilde{a}_{j+\frac{1}{2}}>0, \\ f_{j+1} & \widetilde{a}_{j+\frac{1}{2}}<0. \end{cases} \tag{71}$$

或写为

$$f_{j+\frac{1}{2}}=\frac{1}{2}\left[f_j+f_{j+1}\right]-\frac{1}{2}\left|\widetilde{a}_{j+\frac{1}{2}}\right|\left(u_{j+1}-u_j\right) \tag{72}$$

最后进行差分重构，则有

$$\left(\frac{\partial f}{\partial x}\right)_j=\frac{f_{j+\frac{1}{2}}-f_{j-\frac{1}{2}}}{\Delta x} \tag{73}$$

类似地，考虑方程组情况的 Roe 格式。对于一般的双曲守恒律方程组

$$\frac{\partial \mathbf{U}}{\partial t}+\frac{\partial \mathbf{F}(\mathbf{U})}{\partial x}=0 \tag{74}$$

可化为

$$\frac{\partial \mathbf{U}}{\partial t}+\mathbf{A}(\mathbf{U})\frac{\partial \mathbf{U}}{\partial x}=0 \tag{75}$$

其中 Jacobian 矩阵 $\mathbf{A}=\frac{\mathbf{F}(\mathbf{U})}{\partial \mathbf{U}}$。与前述单波方程情况类似，现在目标化为在 $[j,\ j+1]$ 或记为 $[R,\ L]$ 区间内，寻找一个矩阵 $\widetilde{\mathbf{A}}=\widetilde{\mathbf{A}}\left(\mathbf{U}_R,\ \mathbf{U}_L\right)$，采用该常矩阵（平均变化率）代替变化的 $\mathbf{A}(\mathbf{U})$。平均变化率矩阵 $\widetilde{\mathbf{A}}$ 应具有以下两个性质：

a) 满足 $\mathbf{F}(\mathbf{U}_R)-\mathbf{F}(\mathbf{U}_L)=\widetilde{\mathbf{A}}\left(\mathbf{U}_R,\ \mathbf{U}_L\right)\left(\mathbf{U}_R-\mathbf{U}_L\right)$。同样，根据 Lagrange 中值定理可知，在 $[\mathbf{U}_L,\ \mathbf{U}_R]$ 内必有 $\mathbf{U}_{Roe}$，使得 $\widetilde{\mathbf{A}}\left(\mathbf{U}_R,\ \mathbf{U}_L\right)=\mathbf{U}_{Roe}$。

b) $\widetilde{\mathbf{A}}\left(\mathbf{U}_R,\ \mathbf{U}_L\right)$ 可通过相似变换进行对角化。

故原方程组可化为常矩阵系数方程

$$\frac{\partial \mathbf{U}}{\partial t}+\widetilde{\mathbf{A}}\frac{\partial \mathbf{U}}{\partial x}=0 \tag{76}$$

对于其近似 Riemann 解，可得在 $[j,\ j+1]$ 即 $[R,\ L]$ 格点间的数值通量为

$$\mathbf{F}_{j+\frac{1}{2}} = \frac{1}{2}\left[\mathbf{F}\left(\mathbf{U}_R\right) + \mathbf{F}\left(\mathbf{U}_L\right)\right] - \frac{1}{2}\left|\widetilde{\mathbf{A}}\left(\mathbf{U}_R,\ \mathbf{U}_L\right)\right|\left(\mathbf{U}_R - \mathbf{U}_L\right) \tag{77}$$

其中，$\left|\widetilde{\mathbf{A}}\left(\mathbf{U}_R,\ \mathbf{U}_L\right)\right| = \mathbf{S}^{-1}\left|\mathbf{\Lambda}\right|\mathbf{S}$。上式具体推导过程从略。

下面详细说明矩阵 $\widetilde{\mathbf{A}}\left(\mathbf{U}_R,\ \mathbf{U}_L\right)$ 的构造方法。由于原始流通矢量 $\mathbf{F}\left(\mathbf{U}\right)$ 与守恒矢量 $\mathbf{U}$ 不满足二次齐次函数关系，故给出两种解决思路如下：

a) **思路 1**：如前所述，在 $[\mathbf{U}_L,\ \mathbf{U}_R]$ 内直接寻找 $\mathbf{U}_{Roe}$，使得 $\widetilde{\mathbf{A}}\left(\mathbf{U}_R,\ \mathbf{U}_L\right) = \mathbf{U}_{Roe}$，即满足该点增长率为区间平均增长率。但事实上，直接寻找满足条件的 $\mathbf{U}_{Roe}$ 十分困难。

b) **思路 2**：注意到，若流通矢量 $\mathbf{F}$ 表示成某个自变量的二次齐函数，则区间的中点即为 Roe 点。故可首先进行坐标变换（映射），将 $\mathbf{F}\left(\mathbf{U}\right)$ 化为某个齐次函数，之后可立刻得到所需的 $\mathbf{U}_{Roe}$ 信息。

显然，在一般情况下，后一种思路实现更为简单。下面根据后一种思路，以题设所给一维 Euler 方程组为例，对矩阵 $\widetilde{\mathbf{A}}\left(\mathbf{U}_R,\ \mathbf{U}_L\right)$ 进行具体构造。

首先进行坐标变换（映射），引入新变量

$$\mathbf{W} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \rho^{\frac{1}{2}} \begin{bmatrix} 1 \\ u \\ H \end{bmatrix} \tag{78}$$

则有

$$\mathbf{F}(\mathbf{W}) = \begin{bmatrix} w_1 w_2 \\ \frac{\gamma-1}{\gamma} w_1 w_3 + \frac{1+\gamma}{2\gamma} w_2^2 \\ w_3 w_2 \end{bmatrix} \tag{79}$$

易知，$\mathbf{F}(\mathbf{W})$ 为符合要求的二次齐次函数。这一结论也可通过观察增长率矩阵为线性函数矩阵得到，即有

$$\mathbf{C}(\mathbf{W}) = \frac{\partial \mathbf{F}(\mathbf{W})}{\partial \mathbf{W}} = \begin{bmatrix} w_2 & w_1 & 0 \\ \frac{\gamma-1}{\gamma} w_3 & 2w_2 - \frac{\gamma-1}{\gamma} w_2 & \frac{\gamma-1}{\gamma} w_1 \\ 0 & w_3 & w_2 \end{bmatrix} \tag{80}$$

故在 $[j,\ j+1]$ 即 $[R,\ L]$ 区间内，有

$$\mathbf{F}\left(\mathbf{U}_R\right)-\mathbf{F}\left(\mathbf{U}_L\right)=\mathbf{C}\left(\overline{\mathbf{W}}\right)\left(\mathbf{W}_R-\mathbf{W}_L\right) \tag{81}$$

其中，$\overline{\mathbf{W}}$ 即为变换坐标后的 Roe 点，满足 $\overline{\mathbf{W}}=\left(\mathbf{W}_R+\mathbf{W}_L\right)/2$，故以原始守恒矢量表示的 Roe 点为

$$\mathbf{U}_{Roe}=\overline{\mathbf{U}}=\mathbf{U}\left(\overline{\mathbf{W}}\right) \tag{82}$$

最终，可得平均增长率矩阵为

$$\widetilde{\mathbf{A}}\left(\mathbf{U}_R,\ \mathbf{U}_L\right)=\mathbf{A}\left(\overline{\mathbf{U}}\right) \tag{83}$$

其中，$\overline{\mathbf{U}}$ 中的变量可采用以下 Roe 平均（密度加权平均）公式计算，即有

$$\begin{aligned}
\bar{\rho}&=\left[\left(\sqrt{\rho_L}+\sqrt{\rho_R}\right)/2\right]^2\\
\bar{u}&=\left(\sqrt{\rho_L}u_L+\sqrt{\rho_R}u_R\right)/\left(\sqrt{\rho_L}+\sqrt{\rho_R}\right)\\
\bar{H}&=\left(\sqrt{\rho_L}H_L+\sqrt{\rho_R}H_R\right)/\left(\sqrt{\rho_L}+\sqrt{\rho_R}\right)
\end{aligned} \tag{84}$$

根据以上各式得到的密度、速度以及比焓的 Roe 平均量 $\bar{\rho},\ \bar{u},\ \bar{H}$，可计算其他量（如压力、温度、声速等）的 Roe 平均量如下

$$\bar{p}=\frac{\gamma-1}{\gamma}\left(\bar{\rho}\bar{H}-\frac{1}{2}\bar{\rho}\bar{u}^2\right),\quad \bar{c}^2=(\gamma-1)\left(\bar{H}-\frac{\bar{u}^2}{2}\right),\quad \rho H=E+p \tag{85}$$

综合上述推导，总结方程组情况 Roe 格式具体计算步骤如下。设 n 时刻所有网格点上的物理量均已知，现需构造 $\left(\frac{\partial\mathbf{F}(\mathbf{U})}{\partial x}\right)_j$。

a) 运用各类差分格式，计算在格点间 $(j+\frac{1}{2})$ 的左右数值守恒通量 $\mathbf{U}^L_{j+\frac{1}{2}}$ 与 $\mathbf{U}^R_{j+\frac{1}{2}}$；

b) 采用 Roe 平均（密度加权平均）公式计算 Roe 平均的守恒矢量 $\overline{\mathbf{U}}$，获得每个节点上的 $\bar{\rho},\ \bar{u},\ \bar{H},\ \bar{p},\ \bar{c}$ 值；

c) 将 Jacobian 矩阵 $\mathbf{A}(\mathbf{U})$ 进行特征分解，即 $\widetilde{\mathbf{A}}\left(\mathbf{U}_R,\ \mathbf{U}_L\right)=\mathbf{S}^{-1}\mathbf{\Lambda}\mathbf{S}$，计算 $\mathbf{S}^{-1},\ \mathbf{\Lambda},\ \mathbf{S}$；

d) 计算 $\left|\widetilde{\mathbf{A}}\left(\mathbf{U}_R,\ \mathbf{U}_L\right)\right|=\mathbf{S}^{-1}\left|\mathbf{\Lambda}\right|\mathbf{S}$，其中 $|\mathbf{\Lambda}|=\mathrm{diag}\left(|\lambda_1|,|\lambda_2|,\cdots,|\lambda_k|\right)$，实际使用时 $|\lambda_k|$ 可用如下函数代替，其中 ε 为接近零的小正值，即进行“**熵修正**”，在接近零的特征

值周围增加一定耗散。

$$f(\lambda)=\begin{cases}|\lambda| & |\lambda|>\varepsilon,\\ \frac{\lambda^2+\varepsilon^2}{2\varepsilon} & \lambda|\leq\varepsilon.\end{cases} \tag{86}$$

e) 计算数值通量 $\mathbf{F}_{j+\frac{1}{2}}=\frac{1}{2}\left[\mathbf{F}\left(\mathbf{U}_R\right)+\mathbf{F}\left(\mathbf{U}_L\right)\right]-\frac{1}{2}\left|\widetilde{\mathbf{A}}\left(\mathbf{U}_R,\,\mathbf{U}_L\right)\right|\left(\mathbf{U}_R-\mathbf{U}_L\right)$；

f) 进行差分重构，有 $\left(\frac{\partial\mathbf{F}(\mathbf{U})}{\partial x}\right)_j=\frac{\mathbf{F}_{j+\frac{1}{2}}-\mathbf{F}_{j-\frac{1}{2}}}{\Delta x}$。

相比 FVS 类方法，以 Roe 格式为代表的 FDS 类方法，能够在保持方程守恒性的同时，严格保证了特征方向，以达到耗散低、分辨率高的效果。同时 Roe 格式便于推广到高精度格式，在**特征投影分裂**技术中使用 Roe 平均公式即可，效果优于算数平均方法。

但是，原始的 Roe 格式本身精度只有一阶，推广到高阶后，特征方向无法严格保证，且从一维推广到二维或三维后，特征方向同样无法严格保证，将出现一定数值振荡现象。另外，相比 FVS 类方法，Roe 格式计算量更大。

至此，以 Roe 格式为代表的流通差分分裂技术 (FDS) 的基本流程介绍完毕。

高效激波捕捉格式

下面介绍关于激波的数值计算。在可压缩流动的流场中可能有激波（或间断）的存在。Euler 方程所描述的流场中的激波是 Reynolds 数趋近无穷大的极限情况，此时激波厚度为零。在这种情况下，要求能够正确给出激波位置，激波速度和满足激波前后跳跃关系的流动参数。在有激波存在时，流场中不同区域流动结构的特征尺度差异极大，无黏激波厚度的特征尺度为零，而流场特征长度一般为有限值，流动参数穿过激波有间断产生，这给激波的数值计算带来很大困难。

从历史上看，关于激波的数值计算方法可分为两大类，分别为：1) **激波装配法 (Shock Fitting Method)**；2) **激波捕捉法 (Shock Capturing Method)**。激波装配法是把激波作为动边界来处理。本文重点介绍激波捕捉法及其常用格式。在激波捕捉法中，不将激波分离出来作为动边界处理，无需求解激波位置，包含于计算区域内的激波是整个计算的自然产物。但是，在采用高于一阶精度的差分方法数值计算激波时在激波附近的流动参数中有数值振荡产生，常常使得计算无法继续进行或直接发散。而人们的期望是计算所得到的激波比较陡峭，且流动参数在激波前后无或基本无数值振荡产生。为达到这一目标，自然希望流动参数在穿越激波时能够保持单调变化。针对单波方程，Godunov 给出了为使格式为单

调型的充分与必要条件，并证明，不存在高于一阶精度的单调格式 [6]。

在激波数值模拟中，非物理振荡的根源通常来自于三方面，也成为了各类激波模拟方法的理论依据，总结如下：1) **粘性不足，纯物理无法压制振荡**，由此发展了人工粘性方法 (Artificial Viscosity)；2) **差分格式失去（保）单调性**，由此发展了 TVD (Total Variation Diminishing Methods) [7] - [10] 等保单调限制器方法；3) **色散误差导致间断色散**，由此发展了群速度控制方法 (Group Velocity Control, GVC) [11]、耗散比拟方法 [12] 等。同时，其他高效、高精度的激波捕捉格式，如 NND (Non-oscillatory, Non-free-parameters Dissipative Difference Scheme) [13] - [15]，ENO [16] - [18]，WENO (Weighted Essentially Non-oscillatory Scheme) [19] - [20] 等也得到了深入的发展与应用。

下面介绍常用的高效激波捕捉格式，主要介绍公式应用，详细推导可参考相关文献。

1) **TVD 格式 (Total Variation Diminishing Methods)**

TVD 意为“总变差不增”，这种格式的主要思想，是对现有格式进行改造，使其符合 **Harten 条件**，即差分格式可表示为如下形式：

$$u_j^{n+1} = u_j^n + C\left(u_{j+1}^n - u_j^n\right) - D\left(u_j^n - u_{j-1}^n\right) \tag{87}$$

其中，满足 $C \geq 0,\ D \geq 0,\ C + D \leq 1$。

一种常用构造 TVD 格式的方法，是基于一阶迎风格式，增加一个修正项并采用限制函数进行调节与控制。对于单波情况，在格点间 $\left(j + \frac{1}{2}\right)$ 处，改造后守恒形式差分格式为

$$u_{j+\frac{1}{2}} = u_{j+\frac{1}{2}}^L = u_j + \varphi_j^L \cdot \frac{u_{j+1} - u_j}{2} \tag{88}$$

其中，$\varphi_j^L = \varphi\left(r_j^L\right),\ r_j^L = \left(u_j - u_{j-1}\right) / \left(u_{j+1} - u_j\right)$。注意到上式为左值重构，为正通量情况，即对应单波方程中 $a > 0$ 情况，故用 $u_{j+\frac{1}{2}}^L$ 表示。而对应单波方程中 $a < 0$, 即负通量情况，需要进行如下右值重构

$$u_{j-\frac{1}{2}} = u_{j-\frac{1}{2}}^R = u_j - \varphi_j^R \cdot \frac{u_j - u_{j-1}}{2} \tag{89}$$

其中，$\varphi_j^R = \varphi\left(r_j^R\right),\ r_j^R = \left(u_{j+1} - u_j\right) / \left(u_j - u_{j-1}\right)$。

易知，当 $\varphi(r) = 1$ 时，TVD 格式即为二阶中心差分；当 $\varphi(r) = r$ 时，TVD 格式即为二阶迎风差分。所谓限制器 (Limiter) $\varphi(r)$，即通过判断区间光滑程度，来兼顾所得格式的光滑性与相对低耗散。目前发展的限制器 $\varphi(r)$ 种类众多，如图 4 所示，图源 *Wikipedia*。

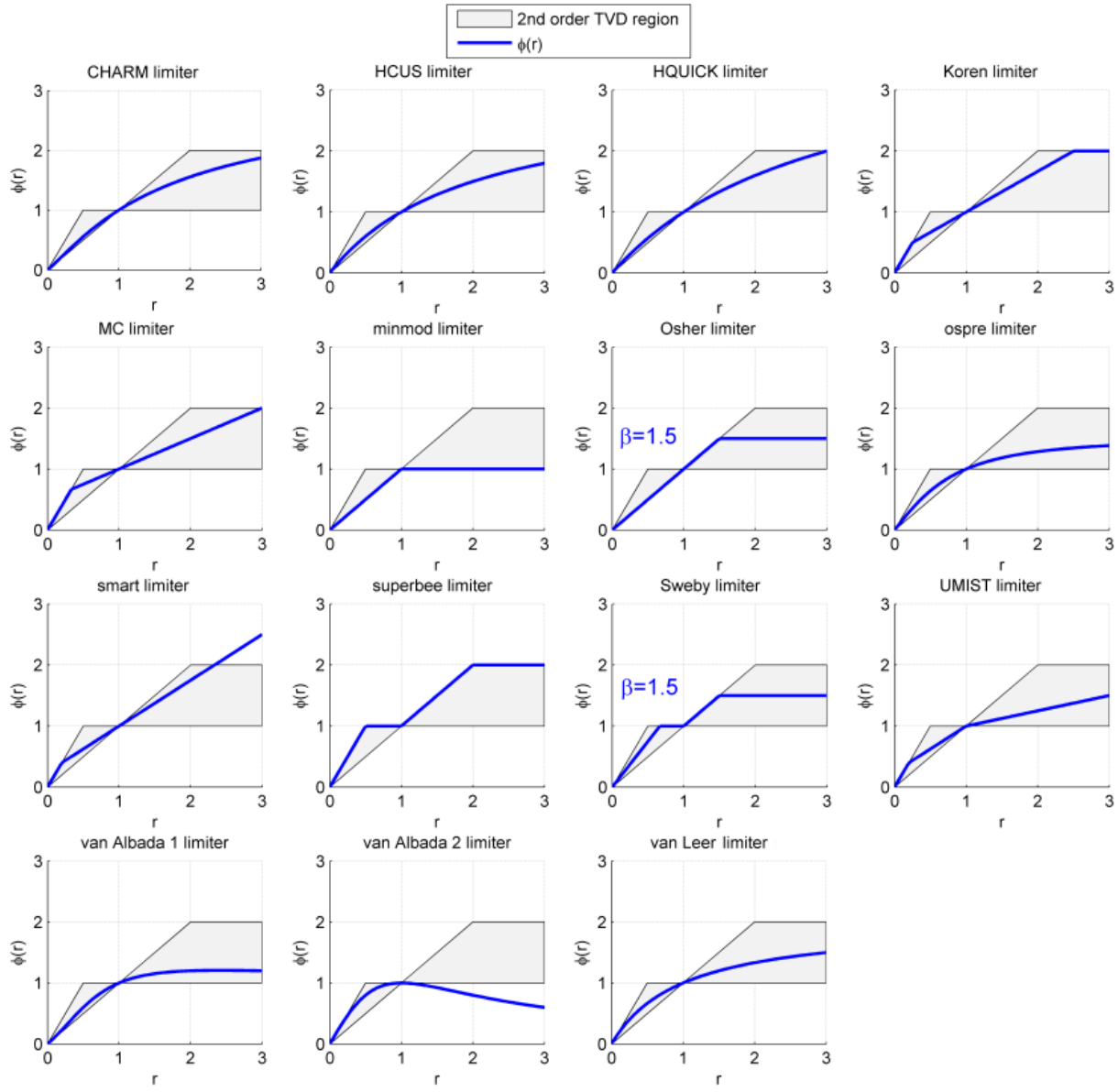


图 4 TVD 格式中常用的 $\varphi(r)$ 限制器 (图源：*Wikipedia*)

本文测试算例选用 van Leer 型限制器，满足 $\varphi(r)=\frac{|r|+r}{|r|+1}$，该限制函数光滑性良好。

2) **NND 格式 (Non-oscillatory, Non-free-parameters Dissipative Difference Scheme)**

NND 格式的基本思想，是将两个不同的格式组合起来，构造出一个在间断前后都具有较好特性的的组合格式。根据色散分析，在间断前二阶迎风格式特性良好，在间断后二阶中心差分格式特性良好，将这两种格式组合起来便得到原始的 NND 格式，意为无波动、无自由参数的耗散差分格式。原始 NND 格式最早由我国著名学者、计算流体力学专家、中国科学院院士**张涵信**提出。

采用前文所述的 FVS 或 FDS 分裂方法，可以得到 $\mathbf{F}^+$, $\mathbf{F}^-$。方程组形式的 NND 格式为

$$\left(\frac{\partial \mathbf{U}}{\partial t}\right)_j=-\frac{1}{\Delta x}\left(\mathbf{F}_{j+\frac{1}{2}}-\mathbf{F}_{j-\frac{1}{2}}\right) \tag{90}$$

其中

$$\begin{gathered}\mathbf{F}_{j+\frac{1}{2}}=\mathbf{F}^+_{j+\frac{1}{2}L}+\mathbf{F}^-_{j+\frac{1}{2}R}\\ \mathbf{F}^+_{j+\frac{1}{2}L}=\mathbf{F}^+_j+\tfrac{1}{2}\min\operatorname{mod}\left(\Delta\mathbf{F}^+_{j-\frac{1}{2}},\Delta\mathbf{F}^+_{j+\frac{1}{2}}\right)\\ \mathbf{F}^-_{j+\frac{1}{2}R}=\mathbf{F}^-_{j+1}-\tfrac{1}{2}\operatorname{minmod}\left(\Delta\mathbf{F}^-_{j+\frac{1}{2}},\Delta\mathbf{F}^-_{j+\frac{3}{2}}\right)\\ \Delta\mathbf{F}^\pm_{j+\frac{1}{2}}=\mathbf{F}^\pm_{j+1}-\mathbf{F}^\pm_j\end{gathered} \tag{91}$$

这里的函数 minmod 定义为

$$\operatorname{minmod}(x,y)=\frac{1}{2}[\operatorname{sign}(x)+\operatorname{sign}(y)]\min(|x|,|y|) \tag{92}$$

NND 格式的本质是把中心差分格式和二阶迎风格式组合起来，这两个格式本身都是不能直接使用的。Lax–Wendroff 格式和迎风 Warming–Beam 格式也可以用来构造 NND 格式。此二格式本身用来解一阶波动方程时都是不发散的，可以预料，组合起来的格式应该比由中心差分格式和二阶迎风格式组合得到的格式效果更好。具体公式可参考吴望一、蔡庆东等 [21]。

NND 格式是我国的科研工作者独立提出的一个成功的数值方法，在国防武器研制中已经获得了成功的应用，同时，格式本身也获得了很大的发展，形成了隐式 NND 格式，各种高阶 NND 格式等一个系列，并且开发了实用的数值计算软件。

3) **WENO 格式 (Weighted Essentially Non-oscillatory Scheme)**

1994 年 Liu 在文献 [22] 中，对高精度的 ENO 格式 (Essentially Non-oscillatory Scheme) 进行改进，利用对模板加权平均的思想提高了方法的计算效率，构造了 WENO 格式。随后，Jiang 等在文献 [19] 中对该方法进一步进行了改进，构造了新的光滑度量因子，进一步提高了计算效率。Jiang 和 Shu 给出的 WENO 格式是目前应用最广泛的 WENO 方法。近年来，众多学者对 Jiang 和 Shu 的 WENO 进行了改进和优化，推出了一系列 WENO 格式。WENO 格式最大的特点在于，其保持高精度的同时还具有非常好的鲁棒性 (Robustness)。

WENO 系列格式的基本构造思路可总结如下：

a) 若需要以差分方法高精度逼近 $\frac{\partial \mathbf{U}}{\partial x}$，则必然需要利用多个基架点；

b) 如果该基架点内函数存在激波（间断），会导致数值振荡；

c) 但激波（间断）不可能处处存在；

d) 将基架点分成多个组（模板），每个模板独立计算所需导数的逼近，得到多个差分形式；

e) 根据每个模板的光滑程度，设定权重；

f) 对多个差分结果进行加权平均。光滑度越高，权重越大。如果该模板内存在间断，则设置其权重趋于零；如果都光滑，则组合成为更高阶的格式。

下面给出应用最广泛的 Jiang 和 Shu 给出的 **5 阶 WENO 原始格式**。

对于以下单波方程，考虑正通量情况 (a > 0)。

$$\frac{\partial u}{\partial x}+\frac{\partial f(u)}{\partial x}=0, \quad f(u)=au, a>0 \tag{93}$$

5 阶 WENO 格式

$$\begin{aligned}
&f_{j+\frac{1}{2}}^{\mathrm{WENO}}=\omega_1 f_{j+1/2}^{(1)}+\omega_2 f_{j+1/2}^{(2)}+\omega_3 f_{j+1/2}^{(3)}\\
&f_{j+\frac{1}{2}}^{(1)}=\tfrac{1}{3}f_{j-2}-\tfrac{7}{6}f_{j-1}+\tfrac{11}{6}f_j\\
&f_{j+\frac{1}{2}}^{(2)}=-\tfrac{1}{6}f_{j-1}+\tfrac{5}{6}f_j+\tfrac{1}{3}f_{j+1}\\
&f_{j+\frac{1}{2}}^{(3)}=\tfrac{1}{3}f_j+\tfrac{5}{6}f_{j+1}-\tfrac{1}{6}f_{j+2}\\
&\omega_k=\tfrac{\alpha_k}{\alpha_1+\alpha_2+\alpha_3} \quad \alpha_k=\tfrac{C_k}{(\varepsilon+IS_k)^p}, \quad k=1,2,3\\
&p=2,\varepsilon=10^{-6}, \quad C_1=\tfrac{1}{10},C_2=\tfrac{6}{10},C_3=\tfrac{3}{10}
\end{aligned} \tag{94}$$

其中光滑度量因子 IS_k 为

$$\begin{aligned} IS_1 &= \tfrac{1}{4}\left(f_{j-2}-4f_{j-1}+3f_j\right)^2+\tfrac{13}{12}\left(f_{j-2}-2f_{j-1}+f_j\right)^2 \\ IS_2 &= \tfrac{1}{4}\left(f_{j-1}-f_{j+1}\right)^2+\tfrac{13}{12}\left(f_{j-1}-2f_j+f_{j+1}\right)^2 \\ IS_3 &= \tfrac{1}{4}\left(3f_j-4f_{j+1}+f_{j+2}\right)^2+\tfrac{13}{12}\left(f_j-2f_{j+1}+f_{j+2}\right)^2 \end{aligned} \tag{95}$$

由此，可进行差分重构，最终有

$$\frac{\partial f_j^{\mathrm{WENO}}}{\partial x}=\frac{f_{j+\frac{1}{2}}^{\mathrm{WENO}}-f_{j-\frac{1}{2}}^{\mathrm{WENO}}}{\Delta x} \tag{96}$$

利用对称性，正通量差分格式中下标“$j+k$”改成“$j-k$”即得到负通量 (a <0) 的差分格式，但差分重构方式保持不变。具体形式不再给出。

至此，本文涉及的常用高效激波捕捉格式介绍完毕。除此之外，本文编写的程序还提供一阶、二阶、三阶偏斜、五阶偏斜**迎风格式**用于数值测试对比，均采用统一的守恒形式编写。

显式时间推进格式

下面介绍本文程序提供的时间推进方法。利用前文所述流通矢量分裂以及各类高精度差分方法，可以得到题设一维 Euler 方程组中的无黏通量项 $\frac{\partial \mathbf{F}(\mathbf{U})}{\partial x}$，则时间推进方程为

$$\frac{\partial \mathbf{U}}{\partial t}=-\frac{\partial \mathbf{F}(\mathbf{U})}{\partial x}=\mathbf{Q}(\mathbf{U}) \tag{97}$$

本文程序提供多种**显式**时间推进格式，简述公式如下：

a) *Euler* 向前显式方法

$$\mathbf{U}^{n+1}=\mathbf{U}^n+\Delta t\mathbf{Q}(\mathbf{U}^n) \tag{98}$$

b) 梯形显式公式

$$\begin{cases} \mathbf{U}^*=\mathbf{U}^n+\Delta t\mathbf{Q}\left(\mathbf{U}^n\right) \\ \mathbf{U}^{n+1}=\mathbf{U}^n+\frac{\Delta t}{2}\left[\mathbf{Q}\left(\mathbf{U}^*\right)+\mathbf{Q}\left(\mathbf{U}^n\right)\right] \end{cases} \tag{99}$$

c) 二阶显式 *Runge-Kutta* 公式 (*Heun* 公式)

$$\begin{cases} \mathbf{U}_1 = \mathbf{U}^n + \Delta t \mathbf{Q}\left(\mathbf{U}^n\right) \\ \mathbf{U}_2 = \mathbf{U}^n + \frac{2\Delta t}{3}\mathbf{Q}\left(\mathbf{U}_1\right) \\ \mathbf{U}^{n+1} = \mathbf{U}^n + \frac{\Delta t}{4}\left[\mathbf{Q}\left(\mathbf{U}_1\right) + 3\mathbf{Q}\left(\mathbf{U}_2\right)\right] \end{cases} \tag{100}$$

d) 三阶保单调 (TVD) 显式 *Runge-Kutta* 公式

$$\begin{cases} \mathbf{U}_1 = \mathbf{U}^n + \Delta t \mathbf{Q}\left(\mathbf{U}^n\right) \\ \mathbf{U}_2 = \frac{3}{4}\mathbf{U}^n + \frac{1}{4}\left[\mathbf{U}_1 + \Delta t \mathbf{Q}\left(\mathbf{U}_1\right)\right] \\ \mathbf{U}^{n+1} = \frac{1}{3}\mathbf{U}^n + \frac{2}{3}\left[\mathbf{U}_2 + \Delta t \mathbf{Q}\left(\mathbf{U}_2\right)\right] \end{cases} \tag{101}$$

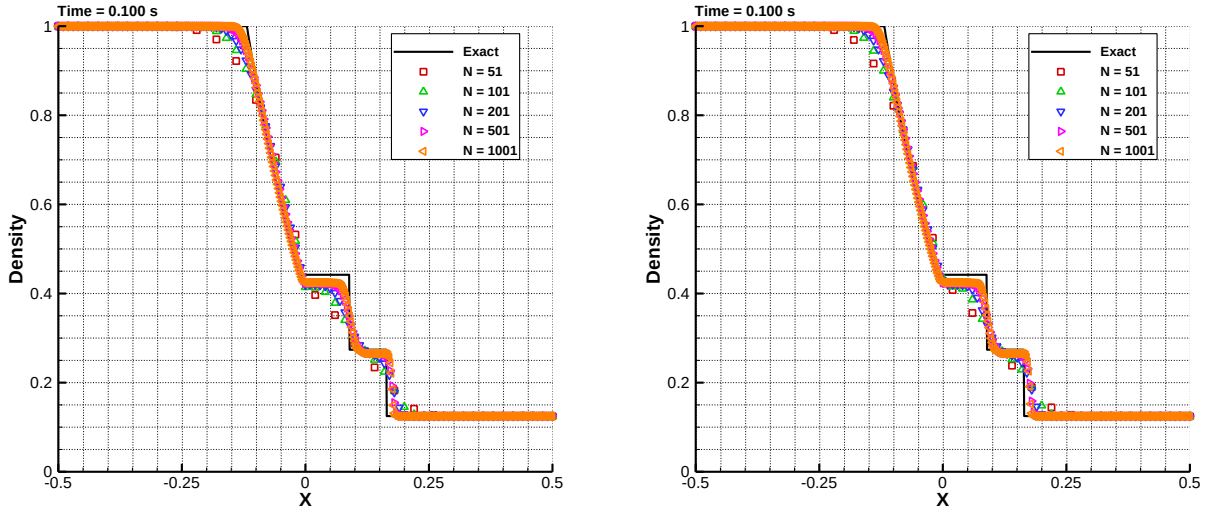
e) 四阶显式 *Runge-Kutta* 公式

$$\begin{cases} \mathbf{U}_1 = \mathbf{U}^n + \Delta t \mathbf{Q}\left(\mathbf{U}^n\right) \\ \mathbf{U}_2 = \mathbf{U}^n + \frac{\Delta t}{2}\mathbf{Q}\left(\mathbf{U}_1\right) \\ \mathbf{U}_3 = \mathbf{U}^n + \frac{\Delta t}{2}\mathbf{Q}\left(\mathbf{U}_2\right) \\ \mathbf{U}_4 = \mathbf{U}^n + \Delta t \mathbf{Q}\left(\mathbf{U}_3\right) \\ \mathbf{U}^{n+1} = \mathbf{U}^n + \frac{\Delta t}{6}\left[\mathbf{Q}\left(\mathbf{U}_1\right) + 2\mathbf{Q}\left(\mathbf{U}_2\right) + 2\mathbf{Q}\left(\mathbf{U}_3\right) + \mathbf{Q}\left(\mathbf{U}_4\right)\right] \end{cases} \tag{102}$$

至此，本文给出的所有显式时间推进方法介绍完毕。

数值计算结果

下面采用前述方法，进行一维 Sod 激波管数值模拟。主程序模块代码详见 **附录 A**，各函数模块代码见 **附录 B**。定义计算域为 $x \in [-0.5,\ 0.5]$。首先进行**网格收敛性分析**。图 **5** 为采用 van Leer 分裂法与 Roe 格式在网格点数 $N = 51,\ 101,\ 201,\ 501,\ 1001$ 时在 t= 0.100s 时计算得到的 Sod 管内密度分布，采用一阶迎风格式，三阶 TVD R-K 时间推进格式。观察两种分裂方法下计算曲线随网格点数 N 变化的趋势，可知当 N 增大时，计算网格域对激波的分辨率随之提升，但当 N 达到 501 时，在两个算例中，继续增大 N 对激波分辨率影响均不大，可认为达到了网格收敛。为节约计算量，且不使得时间步长过小，故在后续数值测试中选择 $N = 501$ 即可。注意到，在题设一维问题中，网格量对计算量的影响较小，但是在二维、三维复杂几何、计算量大的算例中影响很大，故网格收敛性分析具有重要意义。



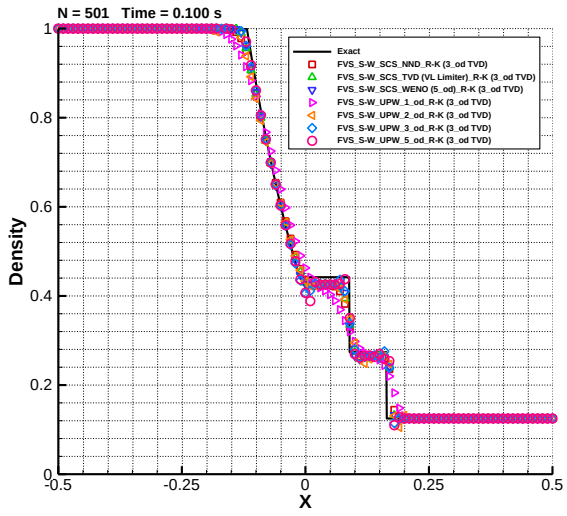
(a) van Leer 分裂法　　(b) Roe 格式

图 5　一维 Sod 激波管内**密度**分布 (t= 0.100s，不同网格数，两种流通矢量分裂方法)

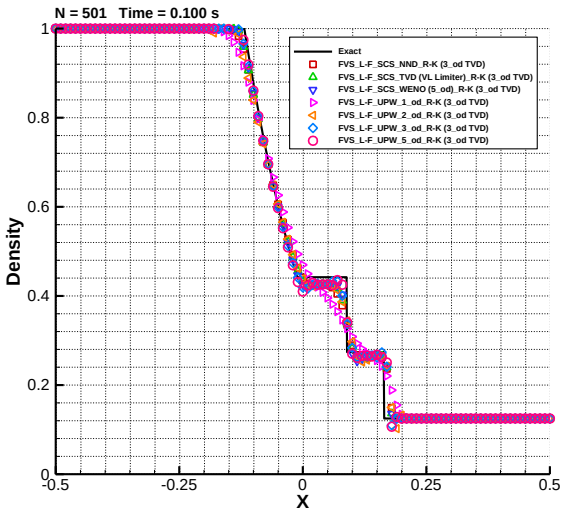
下面展示采用不同的流通矢量分裂方法，得到的一维 Sod 激波管的流场主要变量在 t= 0.100s 时刻与精确解的对比结果，定义计算域为 $x \in [-0.5,\ 0.5]$，网格节点数 $N = 501$。

图 6, 7, 8, 9 分别展示了采用**不同 FVS 分裂方法**得到的激波管内**密度**、**速度**、**压力**以及**比内能**分布，同时给出采用各阶迎风格式 (一阶、二阶、三阶、五阶)、以及各个激波捕捉格式 (TVD, NND, WENO) 的曲线以供对比，时间推进格式选用三阶 TVD R-K 方法。观察曲线图可知，采用**低精度迎风格式**时，采用 van Leer 和 AUSM 分裂方法比 S-W 和 L-F 分裂方法捕捉激波效果更好，但总体精度均较差。当采用**高精度迎风格式**（三阶、五阶）时，采用 S-W 和 L-F 分裂方法激波分辨率有较大改善，但是在间断附近产生一些数值振荡，而采用 van Leer 和 AUSM 分裂方法容易发散，数值振荡很大（由于计算有发散趋势，解中出现复数值，故图中未给出曲线）。当采用各个**激波捕捉格式**时，各种 FVS 分裂方法均能与精确解吻合较好，且数值振荡小，说明 FVS 分裂方法与这些格式结合能够达到较高精度。

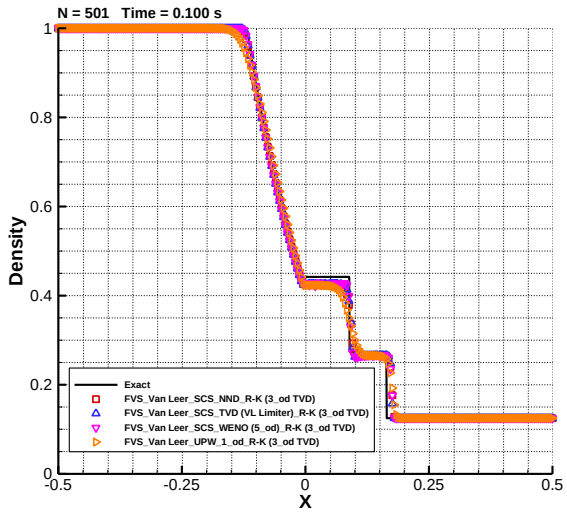
图 10 展示了采用 **Roe 格式**得到的激波管内**密度**、**速度**、**压力**以及**比内能**分布，同时给出采用各阶迎风格式 (仅有一阶，高阶在时间步长较大时均有发散趋势，图中未给出)、以及各个激波捕捉格式 (TVD, NND, WENO) 的曲线以供对比。由图可知，采用**低精度迎风格式**时，Roe 格式稳定但捕捉激波效果较差，采用**高精度迎风格式**时 Roe 格式稳定性很差，且计算量大。当采用各个**激波捕捉格式**时，Roe 格式计算结果均能与精确解吻合较好，且数值振荡小，说明 Roe 格式耗散较低，与高效激波捕捉格式组合能达到较好的激波分辨率。



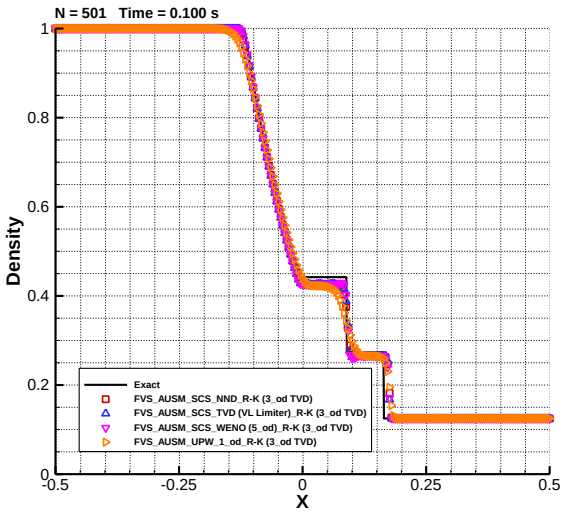
(a) Steger-Warming (S-W)



(b) Lax-Friedrichs (L-F)

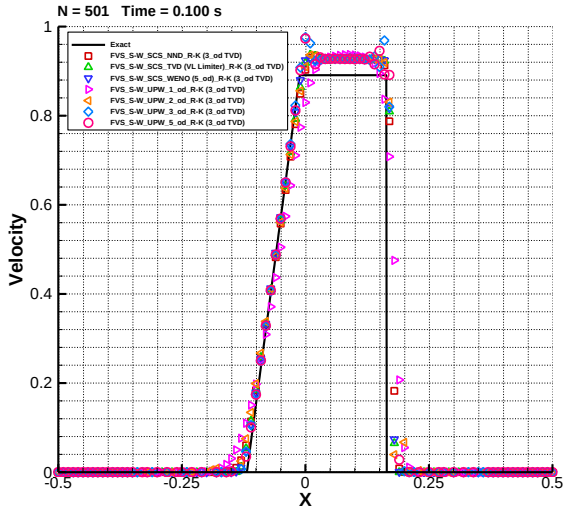


(c) van Leer

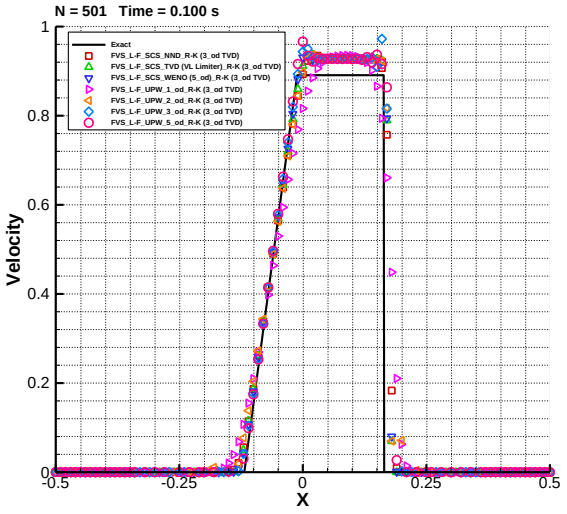


(d) Liou-Steffen (AUSM)

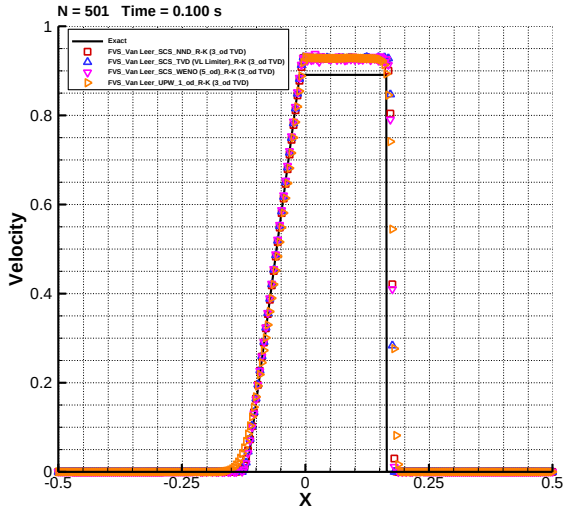
图 6 一维 Sod 激波管内**密度**分布 (t= 0.100s，不同 FVS 分裂方法，每两点一个符号)



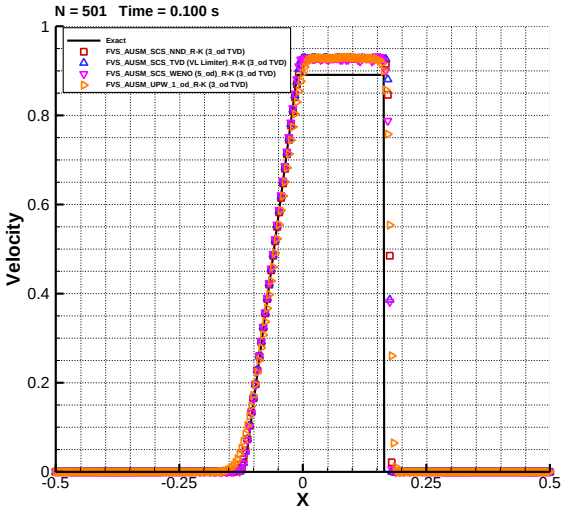
(a) Steger-Warming (S-W)



(b) Lax-Friedrichs (L-F)



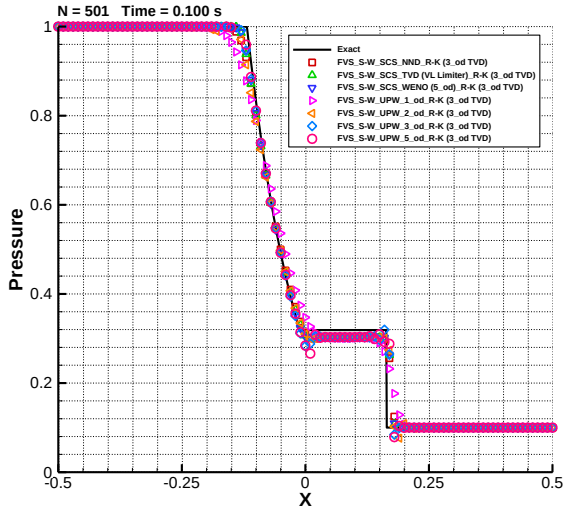
(c) van Leer



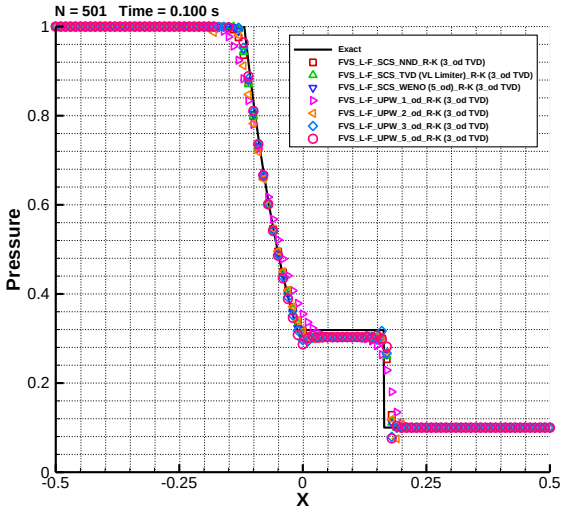
(d) Liou-Steffen (AUSM)

图 7 一维 Sod 激波管内**速度**分布 (t= 0.100s，不同 FVS 分裂方法，每两点一个符号)

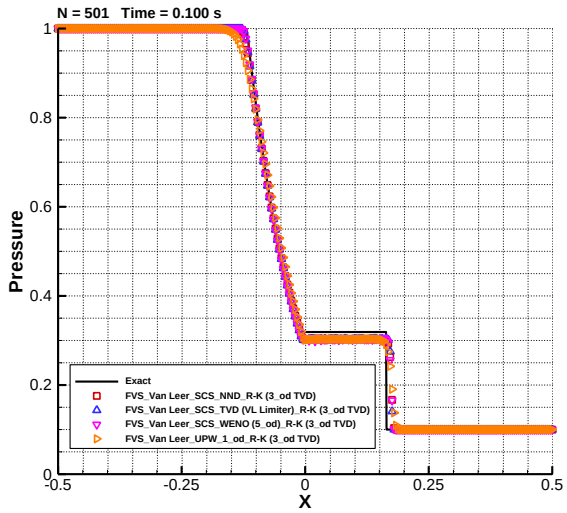
作业 4



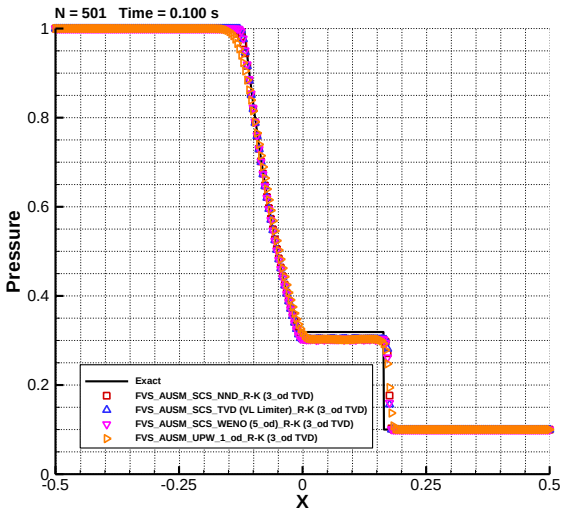
(a) Steger-Warming (S-W)



(b) Lax-Friedrichs (L-F)

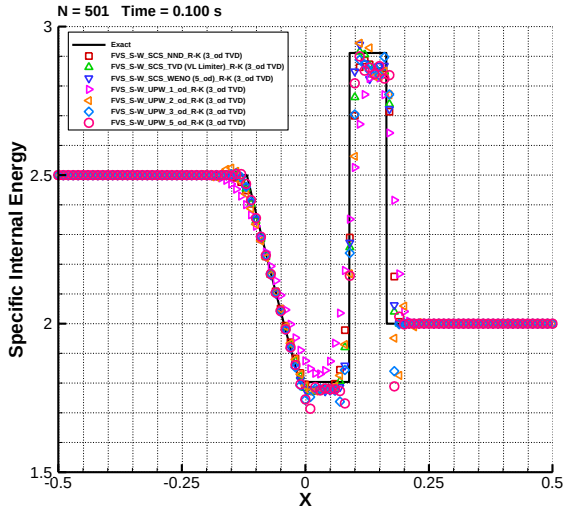


(c) van Leer

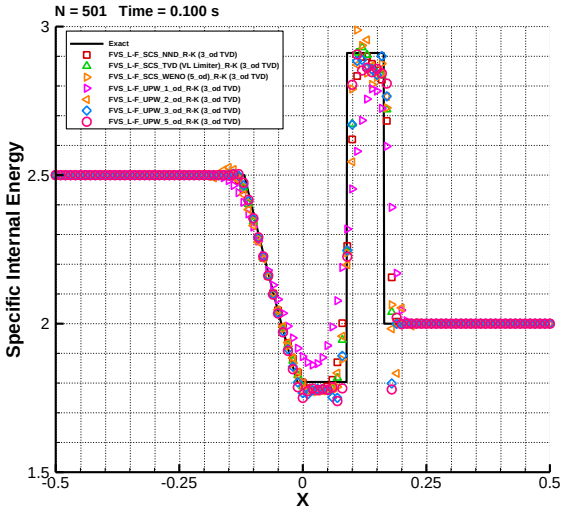


(d) Liou-Steffen (AUSM)

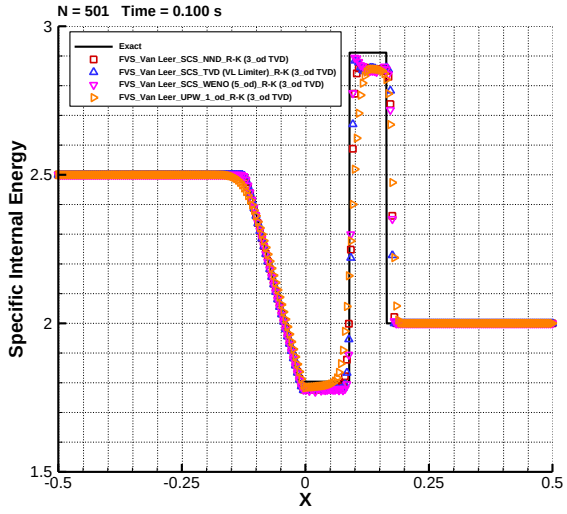
图 8 一维 Sod 激波管内**压力**分布 (t= 0.100s，不同 FVS 分裂方法，每两点一个符号)



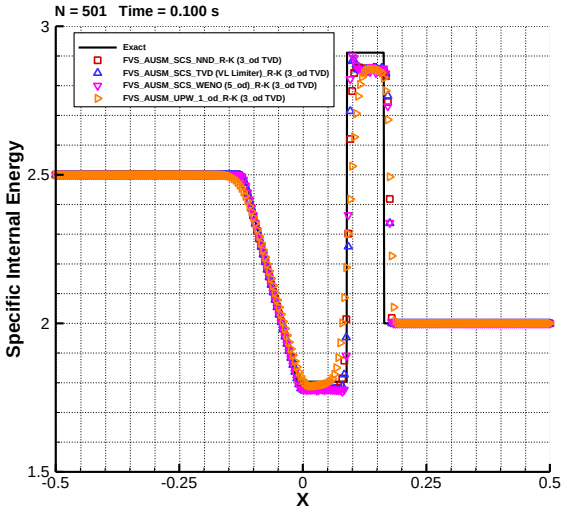
(a) Steger-Warming (S-W)



(b) Lax-Friedrichs (L-F)

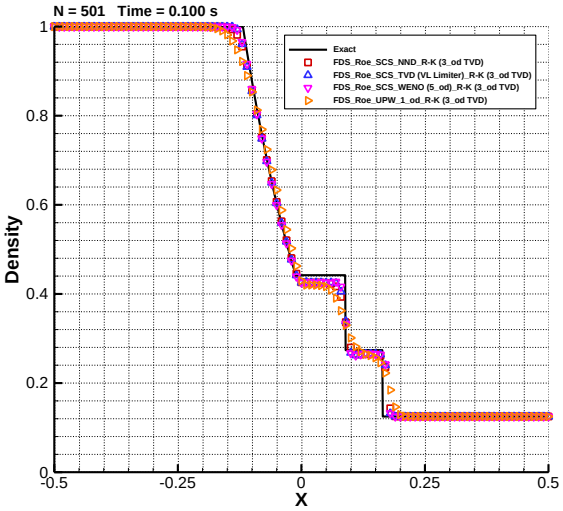


(c) van Leer

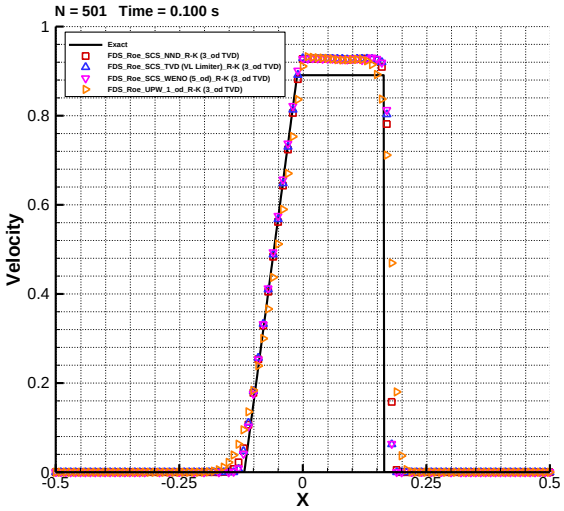


(d) Liou-Steffen (AUSM)

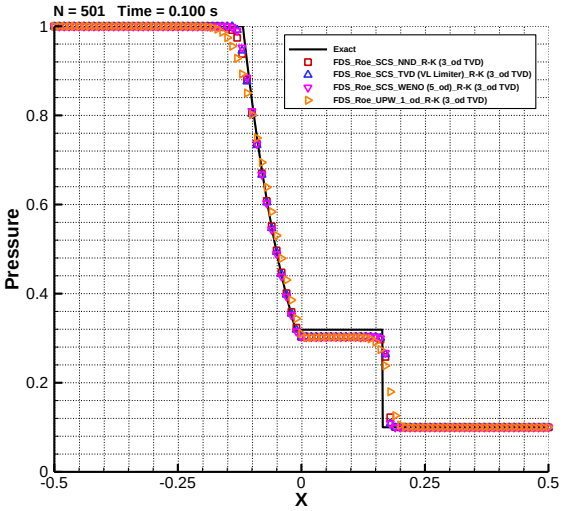
图 9 一维 Sod 激波管内**比内能**分布 (t= 0.100s，不同 FVS 分裂方法，每两点一个符号)



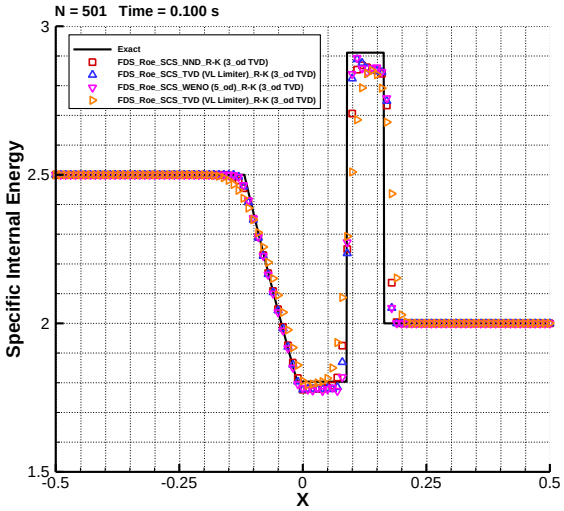
(a) 密度 ρ



(b) 速度 u



(c) 压力 p



(d) 比内能 e

图 10 一维 Sod 激波管内各**变量**分布 (t= 0.100s，**Roe 格式**，每两点一个符号)

参考文献

[1] Gogol (2021). Sod Shock Tube Problem Solver (https://www.mathworks.com/matlabcentral/fileexchange/46311-sod-shock-tube-problem-solver), *MATLAB Central File Exchange.* Retrieved December 28, 2021.

[2] Steger, J. L., & Warming, R. F. (1981). Flux vector splitting of the inviscid gasdynamic equations with application to finite-difference methods. *Journal of computational physics*, 40(2), 263-293.

[3] Van Leer, B. (1997). Flux-vector splitting for the Euler equation. In Upwind and high-resolution schemes (pp. 80-89). *Springer*, Berlin, Heidelberg.

[4] Liou, M. S., & Steffen Jr, C. J. (1993). A new flux splitting scheme. *Journal of Computational physics*, 107(1), 23-39.

[5] Roe, P. L. (1981). Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of computational physics*, 43(2), 357-372.

[6] Godunov, S., & Bohachevsky, I. (1959). Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics. *Matematičeskij sbornik*, 47(3), 271-306.

[7] Jennings, G. (1974). Discrete shocks. *Communications on pure and applied mathematics*, 27(1), 25-37.

[8] Van Leer, B. (1979). Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method. *Journal of computational Physics*, 32(1), 101-136.

[9] Yee, H. C., Warming, R. F., & Harten, A. (1985). Implicit total variation diminishing (TVD) schemes for steady-state calculations. *Journal of Computational Physics*, 57(3), 327-360.

[10] Sweby, P. K. (1984). High resolution schemes using flux limiters for hyperbolic conservation laws. *SIAM journal on numerical analysis*, 21(5), 995-1011.

[11] Fu, D., & Ma, Y. (1997). A high order accurate difference scheme for complex flow fields. *Journal of Computational physics*, 134(1), 1-15.

[12] 马延文, & 傅德薰. (1992). 计算空气动力学中一个新的激波捕捉法——耗散比拟法. *中国科学 (A 辑数学物理学天文学技术科学)*, 35(3), 263-271.

[13] 张涵信. (1984). 差分计算中激波上、下游解出现波动的探讨. *空气动力学学报* (01), 14-21.

[14] 张涵信. (1988). 无波动, 无自由参数的耗散差分格式. *空气动力学学报* (2).

[15] ZHUANG, F., & ZHANG, H. (1987). Computational fluid dynamics in China. *In 8th Computational Fluid Dynamics Conference* (p. 1134).

[16] Harten, A., Engquist, B., Osher, S., & Chakravarthy, S. R. (1987). Uniformly high order accurate essentially non-oscillatory schemes, III. In Upwind and high-resolution schemes (pp. 218-290). *Springer*, Berlin, Heidelberg.

[17] Shu, C. W., & Osher, S. (1988). Efficient implementation of essentially non-oscillatory shock-capturing schemes. *Journal of computational physics*, 77(2), 439-471.

[18] Chakravarthy, S. R. (1990). Some Aspects of Essentially Nonoscillatory (ENO) Formulations for the Euler Equations. *National Aeronautics and Space Administration, Office of Management, Scientific and Technical Information Division.*

[19] Jiang, G. S., & Shu, C. W. (1996). Efficient implementation of weighted ENO schemes. *Journal of computational physics*, 126(1), 202-228.

[20] 刘儒勋, & 舒其望. (2003). 计算流体力学的若干新方法. *科学出版社*.

[21] 吴望一, 蔡庆东.(2000). 时间空间均为二阶的新型 NND 差分格式. *应用数学和力学* (06),561-572.

[22] Liu, X. D., Osher, S., & Chan, T. (1994). Weighted essentially non-oscillatory schemes. *Journal of computational physics*, 115(1), 200-212.

附录 A

本文求解一维 Sod 激波管流场随时间演化问题的**主程序模块** *MATLAB* 代码展示如下。

```
1
2  %% Numerical simulation of 1-D compressible flow
3  %% Study Case: Sod Shock Tube
4  %% Type: Approximate Riemann solution
5  %% Solve the 1-D Euler equation
6  %% Using Compressible Solver
7  %% Involve flux splitting (FVS / FDS) + high-resolution upwind schemes ...
       (UPW) / shock capturing schemes (SCS)
8  %% Version: 2.0
9  %% Date: 2021/12/21
10
11 %% HW: Fundamentals of Computational Fluid Dynamics
12 %% Name: Feng Zhenghao
13 %% College: College of Engineering
14 %% ID: 2101112008
15
16 %% Program Initiation
17 clear
18 clf
19 close all
20 clc
21
22 warning off
23
24 %% Global Variables Definition
25
26 % global L Gamma R Cv;
27
28 % Characteristic length of the tube (x = [-L/2, L/2])
29 L = 1.0;
30
31 % Flow Paremeters
32 Gamma = 1.4;
33 R = 286.9;  % J/kg * K
```

```
% R = 8.314;
Cv = R / (Gamma - 1);              % Specific heat at constant volume
Cp = (Gamma * R) / (Gamma - 1);  % Specific heat at constant pressure

%% Grid Generation
N = 501;  % Number of grids space in the x coordinate
xp = linspace(-L / 2, L / 2, N)';  % x coordinate of grid points
dx = L / (N - 1);

xp_mid = ceil(N / 2); % The grid number of x = 0 position

% Arrays of Properties
u_arr = zeros(N, 1);      % velocity
rho_arr = zeros(N, 1);    % density
p_arr = zeros(N, 1);      % pressure

rho = zeros(N, 1);
u = zeros(N, 1);
p = zeros(N, 1);
E = zeros(N, 1);
T = zeros(N, 1);
c = zeros(N, 1);

% Time interval (values can be adjusted to ensure the stability)
dt = 0.0001;

% Max. cal. round
max_step = 1000;

% Max. elapsed time
max_tot_time = max_step * dt;

%% Pre-processing: I/O File Setting
% Set the global I/O file folder path
savefolder = ['Program_Sod_Shock_Tube', '_MaxTime_',num2str(max_tot_time, ...
    '%.3f')];
save_output_folder = ['.\', savefolder, '\'];
mkdir(save_output_folder);

```

```
%% Setting Initial Condition
% when x <  0, (ul, rhol, pl) = (0.0, 1.0   , 1.0);
% when x ≥ 0, (ur, rhor, pr) = (0.0, 0.125, 0.1);
u_arr(1 : (xp_mid-1)) =   0.0;
rho_arr(1 : (xp_mid-1)) = 1.0;
p_arr(1 : (xp_mid-1)) =   1.0;

u_arr(xp_mid : end) =   0.0;
rho_arr(xp_mid : end) = 0.125;
p_arr(xp_mid : end) =   0.1;

%% Pre-processing: Selected Setting & Algorithm
% Specify the Flux Splitting Method
% 1 - Flux Vector Splitting (FVS)
% 2 - Flux Differnce Splitting (FDS)
flag_flu_spl = 1;

% Family of FVS methods
% 1 - Steger-Warming (S-W)
% 2 - Lax-Friedrich (L-F)
% 3 - Van Leer
% 4 - Liou-Steffen Splitting - Advection Upstream Splitting (AUSM) Method
flag_fvs_met = 1;

% Specify the Flux Reconstruction Method
% 1 - Direct Reconstruction (for F(U))
% 2 - Characteristic Reconstruction
flag_flu_rec = 1;

% Family of FDS methods
% FDS - Roe scheme
flag_fds_met = 1;

% Specify the Algorithm of High-Resoluation Flux Spatial discretization ...
    (Shock Capturing Scheme)
% For Complete Flux Reconstruction
% Avilable types
% 1 - General Upwind & Compact Schemes (forward / backward)
% 2 - Special Shock-Capturing Schemes
```

```
110 flag_spa_typ = 1;
111
112 % Global
113 % [General Upwind & Compact Schemes (forward / backward)]
114 % 1 - 1_od upwind scheme (2 points) [Safest! Treated as Standard Benchmark]
115 % 2 - 2_od upwind scheme (3 points)
116 % 3 - 3_od upwind scheme (4 points with bias)
117 % 4 - 5_od upwind scheme (6 points with bias)
118 % compact scheme (five points with 2_od C.S.)
119 flag_upw_typ = 1;
120
121 % [Special Shock-Capturing Schemes]
122 % Godunov
123 % 1 - TVD - Total Variation Diminishing Scheme (Van Leer Limiter)
124 % 2 - NND - Non-oscillatory, Non-free-paremeters Dissipative Difference ...
        Scheme
125 % 3 - WENO - Weighted Essentially Non-Oscillatory Method (e.g. 5 od WENO ...
        proposed by Jiang & Shu)
126 % MUSCL - Monotone Upstream-Centered Schemes for Conservation Laws
127 flag_scs_typ = 1;
128
129 % Specify the Algorithm for temporal marching [OK]
130 % Global
131 % 1 - Euler time marching
132 % 2 - Trapezoid formula
133 % 3 - 2_od Runge-Kunta method (Heun formula)
134 % 4 - 3_od TVD Runge-Kunta method
135 % 5 - 4_od Runge-Kunta method
136 flag_tim_mar = 4;
137
138 % Zone title set (Export results)
139 zone_title_FVM = ["FVS", "FDS"];
140 zone_title_FVS = ["S-W", "L-F", "Van Leer", "AUSM"];
141 zone_title_FDS = ["Roe"];
142 zone_title_SPA = ["UPW", "SCS"];
143 zone_title_UPW = ["1_od", "2_od", "3_od", "5_od"];
144 zone_title_SCS = ["TVD (VL Limiter)", "NND", "WENO (5_od)"];
145 zone_title_MAR = ["Euler", "Trape", "R-K (2_od)", "R-K (3_od TVD)", "R-K ...
        (4_od)"];
```

```
146
147 if (flag_flu_spl == 1)
148 zone_title_FVM_comb = [char(zone_title_FVM(flag_flu_spl)), '_', ...
        char(zone_title_FVS(flag_fvs_met))];
149 elseif (flag_flu_spl == 2)
150 zone_title_FVM_comb = [char(zone_title_FVM(flag_flu_spl)), '_', ...
        char(zone_title_FDS(flag_fds_met))];
151 end
152 if (flag_spa_typ == 1)
153 zone_title_SPA_comb = [char(zone_title_SPA(flag_spa_typ)), '_', ...
        char(zone_title_UPW(flag_upw_typ))];
154 elseif (flag_spa_typ == 2)
155 zone_title_SPA_comb = [char(zone_title_SPA(flag_spa_typ)), '_', ...
        char(zone_title_SCS(flag_scs_typ))];
156 end
157 zone_title_MAR_comb = char(zone_title_MAR(flag_tim_mar));
158
159 zone_title_comb = [zone_title_FVM_comb, '_', zone_title_SPA_comb, '_', ...
        zone_title_MAR_comb];
160
161 % Export movie settings
162 flag_exp_mov = 0;
163 flag_exp_avi = 0;
164 flag_exp_gif = 1;
165
166 %% Array Initiation
167 lambda = zeros(3, 1);      % eigenvalues matrix
168 lambda_p = zeros(3, 1);  % pos. eigenvalues matrix
169 lambda_n = zeros(3, 1);  % neg. eigenvalues matrix
170
171 U = zeros(N, 1);      % variables vector
172 % F = zeros(N, 1);      % invisid vector
173 F_p = zeros(N, 3);    % pos. invisid vector
174 F_n = zeros(N, 3);    % neg. invisid vector
175 Fx = zeros(N, 3);     % invisid term
176 Fx_p = zeros(N, 3);   % pos. invisid term
177 Fx_n = zeros(N, 3);   % neg. invisid term
178
179 Fh_p = zeros((N - 1), 3); % half point (j + 1/2) invisid vector
```

```
180 Fh_n = zeros((N - 1), 3); % half point (j + 1/2) invisid vector
181
182 % Cal. the initial rho, u, p, T, c
183 for i = 1 : N
184
185 % Step 1: Cal. rho, u, p, T, c
186 rho = rho_arr(i);
187 u = u_arr(i);
188 p = p_arr(i);
189
190 T = p / (rho * R);
191 c = sqrt(Gamma * p / rho);
192
193 E = rho * ((Cv * T) + (0.5 * u * u));
194
195 U(i, 1) = rho;
196 U(i, 2) = rho * u;
197 U(i, 3) = E;
198
199 end
200
201 %% Movie Setting
202 % Create the movie handle
203 F = struct('cdata',[],'colormap',[]);
204 % Export path for flow field movie
205 % avi_obj = VideoWriter('test.avi', 'Uncompressed AVI');
206 if (flag_exp_avi == 1)
207
208 avi_obj = VideoWriter([save_output_folder, 'Sod_Shock_Tube.avi'], 'Motion ...
      JPEG AVI');
209 avi_obj.Quality = 95;
210 avi_obj.FrameRate = 100;
211 % open the .avi handle
212 open(avi_obj);
213
214 end
215
216 %% Start Calculation & Time marching
217 % The real cal. time
```

```
218 t = 0;
219 cnt_step = 0;
220
221 while (cnt_step < max_step)
222
223 t = t + dt;
224 cnt_step = cnt_step + 1;
225
226 % Temporal discretization
227 if (flag_tim_mar == 1)
228 % 1 - Euler time marching
229
230 if (flag_flu_spl == 1)
231 % 1 - Flux Vector Splitting (FVS)
232 [F_p, F_n] = Flux_Vect_Split_Common(U, N, Gamma, Cp, Cv, R, flag_fvs_met);
233 [xs_new, xt_new, Fh_p, Fh_n, Fx, Fx_p, Fx_n] = Diff_Cons_Common(N, dx, ...
        F_p, F_n, flag_spa_typ, flag_upw_typ, flag_scs_typ);
234
235 elseif (flag_flu_spl == 2)
236 % 2 - Flux Differnce Splitting (FDS)
237 [xs_new, xt_new, Fx] = Flux_Diff_Split_Common(U, N, dx, Gamma, Cp, Cv, R, ...
        flag_fds_met, flag_spa_typ, flag_upw_typ, flag_scs_typ);
238
239 end
240
241 U = U + (dt * ((-1) * Fx));  % Q(U) = (-1) * Fx
242
243 elseif (flag_tim_mar == 2)
244 % 2 - Trapezoid formula (2_od, i.e. improved Euler formula)
245
246 if (flag_flu_spl == 1)
247 % 1 - Flux Vector Splitting (FVS)
248 [F_p, F_n] = Flux_Vect_Split_Common(U, N, Gamma, Cp, Cv, R, flag_fvs_met);
249 [¬, ¬, ¬, ¬, Fx, ¬, ¬] = Diff_Cons_Common(N, dx, F_p, F_n, flag_spa_typ, ...
        flag_upw_typ, flag_scs_typ);
250
251 U_1 = U + (dt * ((-1) * Fx));
252 [F_p_1, F_n_1] = Flux_Vect_Split_Common(U_1, N, Gamma, Cp, Cv, R, ...
        flag_fvs_met);
```

```
253 [xs_new, xt_new, Fh_p, Fh_n, Fx_1, Fx_p, Fx_n] = Diff_Cons_Common(N, dx, ...
        F_p_1, F_n_1, flag_spa_typ, flag_upw_typ, flag_scs_typ); % Q(U_1) = ...
        (-1) * Fx_1
254
255 U = (0.5 * U) + (0.5 * U_1) + ((0.5 * dt) * ((-1) * Fx_1));
256
257 elseif (flag_flu_spl == 2)
258 % 2 - Flux Differnce Splitting (FDS)
259 [¬, ¬, Fx] = Flux_Diff_Split_Common(U, N, dx, Gamma, Cp, Cv, R, ...
        flag_fds_met, flag_spa_typ, flag_upw_typ, flag_scs_typ);
260
261 U_1 = U + (dt * ((-1) * Fx));
262 [xs_new, xt_new, Fx_1] = Flux_Diff_Split_Common(U_1, N, dx, Gamma, Cp, ...
        Cv, R, flag_fds_met, flag_spa_typ, flag_upw_typ, flag_scs_typ);
263
264 U = (0.5 * U) + (0.5 * U_1) + ((0.5 * dt) * ((-1) * Fx_1));
265
266 end
267
268 elseif (flag_tim_mar == 3)
269 % 3 - 2_od Runge-Kunta method (Heun formula)
270
271 if (flag_flu_spl == 1)
272 % 1 - Flux Vector Splitting (FVS)
273 [F_p, F_n] = Flux_Vect_Split_Common(U, N, Gamma, Cp, Cv, R, flag_fvs_met);
274 [¬, ¬, ¬, ¬, Fx, ¬, ¬] = Diff_Cons_Common(N, dx, F_p, F_n, flag_spa_typ, ...
        flag_upw_typ, flag_scs_typ);
275
276 U_1 = U + ((dt / 3) * ((-1) * Fx));
277 [F_p_1, F_n_1] = Flux_Vect_Split_Common(U_1, N, Gamma, Cp, Cv, R, ...
        flag_fvs_met);
278 [¬, ¬, ¬, ¬, Fx_1, ¬, ¬] = Diff_Cons_Common(N, dx, F_p_1, F_n_1, ...
        flag_spa_typ, flag_upw_typ, flag_scs_typ); % Q(U_1) = (-1) * Fx_1
279
280 U_2 = U + (((2 * dt) / 3) * ((-1) * Fx_1));
281 [F_p_2, F_n_2] = Flux_Vect_Split_Common(U_2, N, Gamma, Cp, Cv, R, ...
        flag_fvs_met);
282 [xs_new, xt_new, Fh_p, Fh_n, Fx_2, Fx_p, Fx_n] = Diff_Cons_Common(N, dx, ...
        F_p_2, F_n_2, flag_spa_typ, flag_upw_typ, flag_scs_typ); % Q(U_2) = ...
```

```
    (-1) * Fx_2

U = ((1 / 4) * U) + ((3 / 4) * U_1) + (((3 / 4) * dt) * ((-1) * Fx_2));

elseif (flag_flu_spl == 2)
% 2 - Flux Differnce Splitting (FDS)
[~, ~, Fx] = Flux_Diff_Split_Common(U, N, dx, Gamma, Cp, Cv, R, ...
    flag_fds_met, flag_spa_typ, flag_upw_typ, flag_scs_typ);

U_1 = U + ((dt / 3) * ((-1) * Fx));
[~, ~, Fx_1] = Flux_Diff_Split_Common(U_1, N, dx, Gamma, Cp, Cv, R, ...
    flag_fds_met, flag_spa_typ, flag_upw_typ, flag_scs_typ);

U_2 = U + (((2 * dt) / 3) * ((-1) * Fx_1));
[xs_new, xt_new, Fx_2] = Flux_Diff_Split_Common(U_2, N, dx, Gamma, Cp, ...
    Cv, R, flag_fds_met, flag_spa_typ, flag_upw_typ, flag_scs_typ);

U = ((1 / 4) * U) + ((3 / 4) * U_1) + (((3 / 4) * dt) * ((-1) * Fx_2));

end

elseif (flag_tim_mar == 4)
% 4 - 3_od TVD Runge-Kunta method

if (flag_flu_spl == 1)
% 1 - Flux Vector Splitting (FVS)
[F_p, F_n] = Flux_Vect_Split_Common(U, N, Gamma, Cp, Cv, R, flag_fvs_met);
[~, ~, ~, ~, Fx, ~, ~] = Diff_Cons_Common(N, dx, F_p, F_n, flag_spa_typ, ...
    flag_upw_typ, flag_scs_typ);

U_1 = U + (dt * ((-1) * Fx));
[F_p_1, F_n_1] = Flux_Vect_Split_Common(U_1, N, Gamma, Cp, Cv, R, ...
    flag_fvs_met);
[~, ~, ~, ~, Fx_1, ~, ~] = Diff_Cons_Common(N, dx, F_p_1, F_n_1, ...
    flag_spa_typ, flag_upw_typ, flag_scs_typ); % Q(U_1) = (-1) * Fx_1

U_2 = ((3 / 4) * U) + ((1 / 4) * U_1) + (((1 * dt) / 4) * ((-1) * Fx_1));
[F_p_2, F_n_2] = Flux_Vect_Split_Common(U_2, N, Gamma, Cp, Cv, R, ...
    flag_fvs_met);
```

```
314 [xs_new, xt_new, Fh_p, Fh_n, Fx_2, Fx_p, Fx_n] = Diff_Cons_Common(N, dx, ...
        F_p_2, F_n_2, flag_spa_typ, flag_upw_typ, flag_scs_typ); % Q(U_2) = ...
        (-1) * Fx_2
315
316 U = ((1 / 3) * U) + ((2 / 3) * U_2) + (((2 / 3) * dt) * ((-1) * Fx_2));
317
318 elseif (flag_flu_spl == 2)
319 % 2 - Flux Differnce Splitting (FDS)
320 [¬, ¬, Fx] = Flux_Diff_Split_Common(U, N, dx, Gamma, Cp, Cv, R, ...
        flag_fds_met, flag_spa_typ, flag_upw_typ, flag_scs_typ);
321
322 U_1 = U + (dt * ((-1) * Fx));
323 [¬, ¬, Fx_1] = Flux_Diff_Split_Common(U_1, N, dx, Gamma, Cp, Cv, R, ...
        flag_fds_met, flag_spa_typ, flag_upw_typ, flag_scs_typ);
324
325 U_2 = ((3 / 4) * U) + ((1 / 4) * U_1) + (((1 * dt) / 4) * ((-1) * Fx_1));
326 [xs_new, xt_new, Fx_2] = Flux_Diff_Split_Common(U_2, N, dx, Gamma, Cp, ...
        Cv, R, flag_fds_met, flag_spa_typ, flag_upw_typ, flag_scs_typ);
327
328 U = ((1 / 3) * U) + ((2 / 3) * U_2) + (((2 / 3) * dt) * ((-1) * Fx_2));
329
330 end
331
332 elseif (flag_tim_mar == 5)
333 % 5 - 4_od Runge-Kunta method
334
335 if (flag_flu_spl == 1)
336 % 1 - Flux Vector Splitting (FVS)
337 [F_p, F_n] = Flux_Vect_Split_Common(U, N, Gamma, Cp, Cv, R, flag_fvs_met);
338 [¬, ¬, ¬, ¬, Fx, ¬, ¬] = Diff_Cons_Common(N, dx, F_p, F_n, flag_spa_typ, ...
        flag_upw_typ, flag_scs_typ);
339
340 U_1 = U + ((1 / 2) * (dt * ((-1) * Fx)));
341 [F_p_1, F_n_1] = Flux_Vect_Split_Common(U_1, N, Gamma, Cp, Cv, R, ...
        flag_fvs_met);
342 [¬, ¬, ¬, ¬, Fx_1, ¬, ¬] = Diff_Cons_Common(N, dx, F_p_1, F_n_1, ...
        flag_spa_typ, flag_upw_typ, flag_scs_typ); % Q(U_1) = (-1) * Fx_1
343
344 U_2 = U + ((1 / 2) * (dt * ((-1) * Fx_1)));
```

```
345  [F_p_2, F_n_2] = Flux_Vect_Split_Common(U_2, N, Gamma, Cp, Cv, R, ...
         flag_fvs_met);
346  [¬, ¬, ¬, ¬, Fx_2, ¬, ¬] = Diff_Cons_Common(N, dx, F_p_2, F_n_2, ...
         flag_spa_typ, flag_upw_typ, flag_scs_typ); % Q(U_2) = (-1) * Fx_2
347
348  U_3 = U + (dt * ((-1) * Fx_2));
349  [F_p_3, F_n_3] = Flux_Vect_Split_Common(U_3, N, Gamma, Cp, Cv, R, ...
         flag_fvs_met);
350  [xs_new, xt_new, Fh_p, Fh_n, Fx_3, Fx_p, Fx_n] = Diff_Cons_Common(N, dx, ...
         F_p_3, F_n_3, flag_spa_typ, flag_upw_typ, flag_scs_typ); % Q(U_3) = ...
         (-1) * Fx_3
351
352  U = ((1 / 3) * (((-1) * U) + U_1 + (2 * U_2) + U_3)) + ((1 / 6) * dt * ...
         ((-1) * Fx_3));
353
354  elseif (flag_flu_spl == 2)
355  % 2 - Flux Differnce Splitting (FDS)
356  [¬, ¬, Fx] = Flux_Diff_Split_Common(U, N, dx, Gamma, Cp, Cv, R, ...
         flag_fds_met, flag_spa_typ, flag_upw_typ, flag_scs_typ);
357
358  U_1 = U + ((1 / 2) * (dt * ((-1) * Fx)));
359  [¬, ¬, Fx_1] = Flux_Diff_Split_Common(U_1, N, dx, Gamma, Cp, Cv, R, ...
         flag_fds_met, flag_spa_typ, flag_upw_typ, flag_scs_typ);
360
361  U_2 = U + ((1 / 2) * (dt * ((-1) * Fx_1)));
362  [¬, ¬, Fx_2] = Flux_Diff_Split_Common(U_2, N, dx, Gamma, Cp, Cv, R, ...
         flag_fds_met, flag_spa_typ, flag_upw_typ, flag_scs_typ);
363
364  U_3 = U + (dt * ((-1) * Fx_2));
365  [xs_new, xt_new, Fx_3] = Flux_Diff_Split_Common(U_3, N, dx, Gamma, Cp, ...
         Cv, R, flag_fds_met, flag_spa_typ, flag_upw_typ, flag_scs_typ);
366
367  U = ((1 / 3) * (((-1) * U) + U_1 + (2 * U_2) + U_3)) + ((1 / 6) * dt * ...
         ((-1) * Fx_3));
368
369  end
370
371  end
372
```

```
373 if (flag_exp_mov == 1)
374
375 % Save the final solution of U at time t_final
376 U_tem = U;
377
378 % Cal. the final properties at time t_final
379 % Cal. rho, u, E, T, p, c, H according to U
380 rho_tem = U_tem(:, 1);
381 u_tem = U_tem(:, 2) ./ U_tem(:, 1);
382 E_tem = U_tem(:, 3);  % Specific total energy (per unit mass)
383 T_tem = ((E_tem ./ rho_tem) - (0.5 .* u_tem .* u_tem)) / Cv;
384 p_tem = rho_tem .* R .* T_tem;
385 c_tem = sqrt(Gamma * p_tem ./ rho_tem);
386 H_tem = (0.5 .* u_tem .* u_tem) + (Cp .* T_tem);  % Specific entropy (per ...
        unit mass)
387 e_tem = (E_tem ./ rho_tem) - (0.5 .* u_tem .* u_tem);  % Specific ...
        internal energy (per unit mass)
388
389 % data_tem = analytic_sod(t);  % Call analytic_sod.m function, obtain the ...
        analytical solution
390
391 % Monitor properties
392 if (cnt_step == 1)
393
394 % Plot the basic contour frame (initial fig.)
395 h = figure;
396 set(0,'CurrentFigure', h)
397 set(gcf,'Position',[680,278,660,520]);
398 set(gcf,'color','w');
399 Plot_Props(t, xp, rho_tem, p_tem, u_tem, e_tem);
400
401 ax = gca;
402 ax.NextPlot = 'replaceChildren';
403 if (flag_exp_mov == 1)
404 F = getframe(h);
405 end
406
407 % Export the movie
408 % .avi video
```

```
409 % Add into the object of pre-created .avi handle
410 if (flag_exp_avi == 1)
411 writeVideo(avi_obj, F);
412 end
413 % .gif picture
414 % Write in the file with GIF89a format
415 if (flag_exp_gif == 1)
416 im = frame2im(F);
417 [I, map] = rgb2ind(im, 256);
418 imwrite(I, map, [save_output_folder, 'Sod_Shock_Tube.gif'], 'GIF', ...
        'Loopcount', inf, 'DelayTime', 0.1);
419 end
420
421 else
422
423 set(0,'CurrentFigure', h)
424 Plot_Props(t, xp, rho_tem, p_tem, u_tem, e_tem);
425
426 drawnow;
427 if (flag_exp_mov == 1)
428 F = getframe(h);
429 end
430
431 % Export the movie
432 % .avi video
433 % Add into the object of pre-created .avi handle
434 if (flag_exp_avi == 1)
435 writeVideo(avi_obj, F);
436 end
437 % .gif picture
438 % Write in the file with GIF89a format
439 if (flag_exp_gif == 1)
440 im = frame2im(F);
441 [I, map] = rgb2ind(im, 256);
442 imwrite(I, map, [save_output_folder, 'Sod_Shock_Tube.gif'], 'GIF', ...
        'WriteMode', 'append', 'DelayTime', 0.1);
443 end
444
445 end
```

```
446
447 end
448
449 end
450
451 %% Post-processing
452
453 % close the .avi handle
454 if (flag_exp_avi == 1)
455 close(avi_obj);
456 end
457
458 % Save the final solution of U at time t_end
459 U_end = U;
460 t_end = t;
461
462 % Cal. the final properties at time t_final
463 % Cal. rho, u, E, T, p, c, H according to U
464 rho_end = U_end(:, 1);
465 u_end = U_end(:, 2) ./ U_end(:, 1);
466 E_end = U_end(:, 3);
467 T_end = ((E_end ./ rho_end) - (0.5 .* u_end .* u_end)) / Cv;
468 p_end = rho_end .* R .* T_end;
469 c_end = sqrt(Gamma * p_end ./ rho_end);
470 H_end = (0.5 .* u_end .* u_end) + (Cp .* T_end);
471 e_end = (E_end ./ rho_end) - (0.5 .* u_end .* u_end);
472
473 % Call analytic_sod.m function, obtain the analytical solution
474 data_end = analytic_sod(t_end);
475
476 % Visualization
477 h_end = figure;
478 % plot(xp, U(:,1)); % rho
479 set(0,'CurrentFigure', h_end);
480 set(gcf,'Position',[680,278,660,520]);
481 set(gcf,'color','w');
482
483 % title(zone_title_comb);
484
```

```
485 subplot(2,2,1);
486 hold on;
487 plot(data_end.x, data_end.rho, '-b', 'LineWidth', 1.5);
488 plot(xp, rho_end, 'bo', 'MarkerSize', 3);
489 hold off;
490 legend('Exact', 'Numerical');
491 xlabel('x');
492 ylabel('Density');
493 title(['Plot of Density vs. Position', ' T = ', num2str(t_end, '%.3f'), ' ...
        s']);
494 axis([-0.5, 0.5, 0.0, 1.0]);
495 grid on;
496
497 subplot(2,2,2);
498 hold on;
499 plot(data_end.x, data_end.P, '-g', 'LineWidth', 1.5);
500 plot(xp, p_end, 'go', 'MarkerSize', 3);
501 hold off;
502 legend('Exact', 'Numerical');
503 xlabel('x');
504 ylabel('Pressure');
505 title(['Plot of Pressure vs. Position', ' T = ', num2str(t_end, '%.3f'), ...
        ' s']);
506 axis([-0.5, 0.5, 0.0, 1.0]);
507 grid on;
508
509 subplot(2,2,3);
510 hold on;
511 plot(data_end.x, data_end.u, '-r', 'LineWidth', 1.5);
512 plot(xp, u_end, 'ro', 'MarkerSize', 3);
513 hold off;
514 legend('Exact', 'Numerical');
515 xlabel('x');
516 ylabel('Velocity');
517 title(['Plot of Velocity vs. Position', ' T = ', num2str(t_end, '%.3f'), ...
        ' s']);
518 axis([-0.5, 0.5, 0.0, 1.0]);
519 grid on;
520
```

```
subplot(2,2,4);
hold on;
plot(data_end.x, data_end.e, '-m', 'LineWidth', 1.5);
plot(xp, e_end, 'mo', 'MarkerSize', 3);
hold off;
legend('Exact', 'Numerical');
xlabel('x');
ylabel('Specific Internal Energy');
title(['Plot of e vs. Position', ' T = ', num2str(t_end, '%.3f'), ' s']);
axis([-0.5, 0.5, 1.5, 3.0]);
grid on;

% Save the present figure
saveas(gcf, [save_output_folder, 'Results_', zone_title_comb, ...
    '_MaxTime_',num2str(max_tot_time, '%.3f'), '.fig']);

% Export the results to Tecplot
% Data preparation
title_cal = zone_title_comb;
zone_title_cal = zone_title_comb;
filename_cal = [save_output_folder, 'Results_Sod_Shock_Tube_', ...
    zone_title_cal, '_Grid_', num2str(N, '%d'), ...
    '_MaxTime_',num2str(max_tot_time, '%.3f'), '.plt'];
variables_cal = {'X', 'Density', 'Pressure', 'Velocity', 'Specific ...
    Internal Energy'};
Mat_Data_cal = [xp(:), rho_end(:), p_end(:), u_end(:), e_end(:)];
IJK_cal = length(xp);

% Create the file
if exist(filename_cal, 'file')
delete(filename_cal)
end
f_id_cal = fopen(filename_cal, 'a');
fclose(f_id_cal);

% Export the exact solution to Tecplot
% Create the header
plt_Head(filename_cal, title_cal, variables_cal);
% Create the format of zone(point)
```

```
556 plt_Zone(filename_cal, zone_title_cal, IJK_cal, Mat_Data_cal);
557
558 title_ana = 'Exact';
559 zone_title_ana = 'Exact';
560 filename_ana = [save_output_folder, 'Results_Sod_Shock_Tube_', ...
        zone_title_ana, '_Grid_', num2str(N, '%d'), ...
        '_MaxTime_',num2str(max_tot_time, '%.3f'), '.plt'];
561 variables_ana = {'X', 'Density', 'Pressure', 'Velocity', 'Specific ...
        Internal Energy'};
562 Mat_Data_ana = [data_end.x(:), data_end.rho(:), data_end.P(:), ...
        data_end.u(:), data_end.e(:)];
563 IJK_ana = length(data_end.x);
564
565 % Create the file
566 if exist(filename_ana, 'file')
567 delete(filename_ana)
568 end
569 f_id_ana = fopen(filename_ana, 'a');
570 fclose(f_id_ana);
571
572 % Create the header
573 plt_Head(filename_ana, title_ana, variables_ana);
574 % Create the format of zone(point)
575 plt_Zone(filename_ana, zone_title_ana, IJK_ana, Mat_Data_ana);
576
577 % Save data of the program
578 save([save_output_folder, 'Results_Variables_', zone_title_comb, ...
        '_MaxTime_',num2str(max_tot_time, '%.3f'), '.mat']);
579
580 % END OF THE PROGRAM!
581
582 % Function module: Create the header
583 function plt_Head(filename, title, variables)
584
585 % Create the header
586 f_id = fopen(filename, 'a');
587 % Name
588 if ¬isempty(title)
589 s = ['TITLE = "', title, '"'];
```

```
590  fprintf(f_id, '%s \r\n', s);
591  end
592  % Variables
593  v = numel(variables);
594  s = 'VARIABLES = ';
595  for k = 1 : v
596  if k ≠ 1
597  s = [s, ','];
598  end
599  s = [s, ' "', variables{k}, '"'];
600  end
601  fprintf(f_id, '%s \r\n', s);
602
603  fclose(f_id);
604
605  end
606
607  % Function module: Create the format of zone(point)
608  function plt_Zone(filename, zone_title, IJK, Mat_Data)
609
610  % Create the format of zone(point)
611  f_id = fopen(filename, 'a');
612  N = size(Mat_Data, 1);
613
614  Dim = numel(IJK);
615  if (Dim == 1)
616  s = ['zone I =', num2str(IJK(1))];
617  elseif (Dim == 2)
618  s = ['zone I =', num2str(IJK(1)), ', J =', num2str(IJK(2))];
619  elseif (Dim == 3)
620  s = ['zone I =', num2str(IJK(1)), ', J =', num2str(IJK(2)), ', K =', ...
         num2str(IJK(3))];
621  end
622
623  % Title
624  if ¬isempty(zone_title)
625  s = [s, ', t = "', zone_title, '"'];
626  end
627  fprintf(f_id, '%s \r\n', s);
```

```
% Point format
s = 'DATAPACKING = point';
fprintf(f_id, '%s \r\n', s);

% Data introduction
for k = 1 : N
fprintf(f_id, '%s \r\n', num2str(Mat_Data(k,:)));
end

fclose(f_id);

end
```

附录 B

本文求解一维 Sod 激波管流场随时间演化问题的**各函数模块**，包括流通矢量差分分裂、Roe 格式、空间差分格式、参考 Riemann 解析解、结果可视化等模块，*MATLAB* 代码展示如下，函数有标题与详细说明。

```

% Function Module: Flux Vector Splitting (FVS) with different methods

function [F_p, F_n] = Flux_Vect_Split_Common(U, N, Gamma, Cp, Cv, R, ...
    flag_fvs_met)

if (flag_fvs_met == 1)
% 1 - FVS - Steger-Warming (S-W)

% Initiation
lambda = zeros(3, 1);     % eigenvalues matrix
lambda_p = zeros(3, 1);   % pos. eigenvalues matrix
lambda_n = zeros(3, 1);   % neg. eigenvalues matrix

F_p = zeros(N, 3);    % pos. invisid vector
F_n = zeros(N, 3);    % neg. invisid vector

```

```
em = 1e-3;

% Splitting
for i = 1 : N

% Step 1: Cal. rho, u, p, T, c according to U
rho = U(i, 1);
u = U(i, 2) / U(i, 1);
E = U(i, 3);
T = ((E / rho) - (0.5 * u * u)) / Cv;
p = rho * R * T;
c = sqrt(Gamma * p / rho);

% Step 2: Cal. eigenvalues lamba
lambda(1) = u;
lambda(2) = u - c;
lambda(3) = u + c;

% Step 3: Splitting the eigenvalues lamba into lamba (S-W)
for k = 1 : 3
lambda_p(k) = (lambda(k) + sqrt((lambda(k)^2) + (em^2))) / 2;
lambda_n(k) = (lambda(k) - sqrt((lambda(k)^2) + (em^2))) / 2;
end

% Step 4: Cal. F+ & F-
w_p = ((3 - Gamma) * (lambda_p(2) + lambda_p(3)) * c * c) / (2 * (Gamma - ...
    1));
F_p(i, 1) = (rho / (2 * Gamma)) * ((2 * (Gamma - 1) * lambda_p(1)) + ...
    lambda_p(2) + lambda_p(3));
F_p(i, 2) = (rho / (2 * Gamma)) * ((2 * (Gamma - 1) * lambda_p(1) * u) + ...
    (lambda_p(2) * (u - c)) + (lambda_p(3) * (u + c)));
% !!!
F_p(i, 3) = (rho / (2 * Gamma)) * (((Gamma - 1) * lambda_p(1) * u * u) + ...
    ((lambda_p(2) / 2) * (u - c) * (u - c)) + (((lambda_p(3) / 2) * (u + ...
    c) * (u + c)) + w_p));

w_n = ((3 - Gamma) * (lambda_n(2) + lambda_n(3)) * c * c) / (2 * (Gamma - ...
    1));
F_n(i, 1) = (rho / (2 * Gamma)) * ((2 * (Gamma - 1) * lambda_n(1)) + ...
```

```
    lambda_n(2) + lambda_n(3));
F_n(i, 2) = (rho / (2 * Gamma)) * ((2 * (Gamma - 1) * lambda_n(1) * u) + ...
    (lambda_n(2) * (u - c)) + (lambda_n(3) * (u + c)));
% !!!
F_n(i, 3) = (rho / (2 * Gamma)) * (((Gamma - 1) * lambda_n(1) * u * u) + ...
    ((lambda_n(2) / 2) * (u - c) * (u - c)) + (((lambda_n(3) / 2) * (u + ...
    c) * (u + c)) + w_n));

end

elseif (flag_fvs_met == 2)
% 2 - FVS - Lax-Friedrich (L-F)

% Initiation
lambda = zeros(3, 1);      % eigenvalues matrix
lambda_p = zeros(3, 1);  % pos. eigenvalues matrix
lambda_n = zeros(3, 1);  % neg. eigenvalues matrix

F_p = zeros(N, 3);    % pos. invisid vector
F_n = zeros(N, 3);    % neg. invisid vector

% Splitting

lambda_s_global = 0;
for i = 1 : N

rho = U(i, 1);
u = U(i, 2) / U(i, 1);
E = U(i, 3);
T = ((E / rho) - (0.5 * u * u)) / Cv;
p = rho * R * T;
c = sqrt(Gamma * p / rho);

% Global splitting - plus eigenvalue
lambda_s_global = max(lambda_s_global, abs(u) + c);

end

for i = 1 : N
```

```
85
86  % Step 1: Cal. rho, u, p, T, c according to U
87  rho = U(i, 1);
88  u = U(i, 2) / U(i, 1);
89  E = U(i, 3);
90  T = ((E / rho) - (0.5 * u * u)) / Cv;
91  p = rho * R * T;
92  c = sqrt(Gamma * p / rho);
93
94  % Step 2: Cal. eigenvalues lamba
95  lambda(1) = u;
96  lambda(2) = u - c;
97  lambda(3) = u + c;
98
99  lambda_s_local = abs(u) + c;
100
101 % Step 3: Splitting the eigenvalues lamba into lamba (L-F)
102 % Local splitting - plus eigenvalue or Global
103 for k = 1 : 3
104 lambda_p(k) = (lambda(k) + lambda_s_local) / 2;
105 lambda_n(k) = (lambda(k) - lambda_s_local) / 2;
106 end
107
108 % Step 4: Cal. F+ & F-
109 w_p = ((3 - Gamma) * (lambda_p(2) + lambda_p(3)) * c * c) / (2 * (Gamma - ...
        1));
110 F_p(i, 1) = (rho / (2 * Gamma)) * ((2 * (Gamma - 1) * lambda_p(1)) + ...
        lambda_p(2) + lambda_p(3));
111 F_p(i, 2) = (rho / (2 * Gamma)) * ((2 * (Gamma - 1) * lambda_p(1) * u) + ...
        (lambda_p(2) * (u - c)) + (lambda_p(3) * (u + c)));
112 F_p(i, 3) = (rho / (2 * Gamma)) * (((Gamma - 1) * lambda_p(1) * u * u) + ...
        ((lambda_p(2) / 2) * (u - c) * (u - c)) + (((lambda_p(3) / 2) * (u + ...
        c) * (u + c)) + w_p));
113
114 w_n = ((3 - Gamma) * (lambda_n(2) + lambda_n(3)) * c * c) / (2 * (Gamma - ...
        1));
115 F_n(i, 1) = (rho / (2 * Gamma)) * ((2 * (Gamma - 1) * lambda_n(1)) + ...
        lambda_n(2) + lambda_n(3));
116 F_n(i, 2) = (rho / (2 * Gamma)) * ((2 * (Gamma - 1) * lambda_n(1) * u) + ...
```

```
    (lambda_n(2) * (u - c)) + (lambda_n(3) * (u + c)));
F_n(i, 3) = (rho / (2 * Gamma)) * (((Gamma - 1) * lambda_n(1) * u * u) + ...
    ((lambda_n(2) / 2) * (u - c) * (u - c)) + (((lambda_n(3) / 2) * (u + ...
    c) * (u + c)) + w_n));

end

elseif (flag_fvs_met == 3)
% 3 - Van Leer

% Initiation
F = zeros(1, 3);      % invisid vector
F_p = zeros(N, 3);    % pos. invisid vector
F_n = zeros(N, 3);    % neg. invisid vector

% Splitting
for i = 1 : N

% Step 1: Cal. rho, u, p, T, c, F according to U
rho = U(i, 1);
u = U(i, 2) / U(i, 1);
E = U(i, 3);
T = ((E / rho) - (0.5 * u * u)) / Cv;
p = rho * R * T;
c = sqrt(Gamma * p / rho);

% F(1) = rho * u;
% F(2) = (rho * u * u) + p;
% F(3) = u * (E + p);

F(1) = U(i, 2);
F(2) = ((Gamma - 1) * U(i, 3)) + (((3 - Gamma) / 2) * ((U(i, 2) * U(i, ...
    2)) / U(i, 1)));
F(3) = (Gamma * (U(i, 2) * U(i, 3)) / U(i, 1)) + (((Gamma - 1) / 2) * ...
    (U(i, 2)^3 / U(i, 1)^2));


% Step 2: Cal. Mach number Ma
Ma = u / c;
```

```
151
152 % Step 3: Splitting by discuss different Ma cases
153 if (Ma ≥ 1)
154 F_p(i, :) = F(1, :);
155 F_n(i, :) = zeros(1, 3);
156 elseif (Ma ≤ -1)
157 F_p(i, :) = zeros(1, 3);
158 F_n(i, :) = F(1, :);
159 else
160 F1_p = rho * c * (((Ma + 1) / 2)^2);
161 F1_n = (-1) * rho * c * (((Ma - 1) / 2)^2);
162
163 F_p(i, 1) = F1_p;
164 F_p(i, 2) = (F1_p / Gamma) * (((Gamma - 1) * u) + (2 * c));
165 F_p(i, 3) = (F1_p / (2 * (Gamma^2 - 1))) * (((Gamma - 1) * u) + (2 * c))^2;
166
167 F_n(i, 1) = F1_n;
168 F_n(i, 2) = (F1_n / Gamma) * (((Gamma - 1) * u) - (2 * c));
169 F_n(i, 3) = (F1_n / (2 * (Gamma^2 - 1))) * (((Gamma - 1) * u) - (2 * c))^2;
170 end
171
172 end
173
174 elseif (flag_fvs_met == 4)
175 % 4 - Liou-Steffen Splitting - Advection Upstream Splitting (AUSM) Method
176
177 % Initiation
178 Fc_p = zeros(1, 3);  % pos. invisid vector convective part
179 Fc_n = zeros(1, 3);  % neg. invisid vector convective part
180 Fp_p = zeros(1, 3);  % pos. invisid vector pressure part
181 Fp_n = zeros(1, 3);  % neg. invisid vector pressure part
182 F_p = zeros(N, 3);   % pos. invisid vector
183 F_n = zeros(N, 3);   % neg. invisid vector
184
185 % Splitting
186 for i = 1 : N
187
188 % Step 1: Cal. rho, u, p, T, c, F according to U
189 rho = U(i, 1);
```

```
190 u = U(i, 2) / U(i, 1);
191 E = U(i, 3);
192 T = ((E / rho) - (0.5 * u * u)) / Cv;
193 p = rho * R * T;
194 c = sqrt(Gamma * p / rho);
195 H = (0.5 * u * u) + (Cp * T);
196
197 % Step 2: Cal. Mach number Ma
198 Ma = u / c;
199
200 % Step 3: Splitting by discuss different Ma cases
201 if (Ma > 1)
202 Ma_p = Ma;
203 Ma_n = 0;
204 p_p = p;
205 p_n = 0;
206 elseif (Ma < -1)
207 Ma_p = 0;
208 Ma_n = Ma;
209 p_p = 0;
210 p_n = p;
211 else
212 Ma_p = ((Ma + 1)^2) / 4;
213 Ma_n = ((-1) * ((Ma - 1)^2)) / 4;
214 p_p = p * ((1 + Ma) / 2);
215 p_n = p * ((1 - Ma) / 2);
216 end
217
218 Fc_p(1) = rho * c * Ma_p;
219 Fc_p(2) = rho * c * Ma_p * u;
220 Fc_p(3) = rho * c * Ma_p * H;
221
222 Fc_n(1) = rho * c * Ma_n;
223 Fc_n(2) = rho * c * Ma_n * u;
224 Fc_n(3) = rho * c * Ma_n * H;
225
226 Fp_p(1) = 0;
227 Fp_p(2) = p_p;
228 Fp_p(3) = 0;
```

```
229
230 Fp_n(1) = 0;
231 Fp_n(2) = p_n;
232 Fp_n(3) = 0;
233
234 F_p(i, 1) = Fc_p(1) + Fp_p(1);
235 F_p(i, 2) = Fc_p(2) + Fp_p(2);
236 F_p(i, 3) = Fc_p(3) + Fp_p(3);
237
238 F_n(i, 1) = Fc_n(1) + Fp_n(1);
239 F_n(i, 2) = Fc_n(2) + Fp_n(2);
240 F_n(i, 3) = Fc_n(3) + Fp_n(3);
241
242 end
243
244 end
245
246 end
247
248
249 % Function Module: Flux Difference Splitting (FDS) with different methods
250
251 function [xs_new, xt_new, Fx] = Flux_Diff_Split_Common(U, N, dx, Gamma, ...
        Cp, Cv, R, flag_fds_met, flag_spa_typ, flag_upw_typ, flag_scs_typ)
252
253 if (flag_fds_met == 1)
254 % FDS - Roe scheme
255 % The variables of all grid points at time n are known.
256 Fh_l = zeros(N, 3);
257 Fh_r = zeros(N, 3);
258 U_ave = zeros(N, 3);  % Roe ave. bar(U) martix
259 F_ave = zeros(N, 3);  % Roe ave. F(bar(U)) martix
260 Fh = zeros(N, 3);
261 Fx = zeros(N, 3);
262 A_ave = zeros(3, 3);  % Roe ave. Jacobian A(bar(U)) martix
263
264 em = 1e-5;
265
266 % Step 1: Cal. Ur, Ul with difference schemes
```

```
267 % Cal. all the variables of all grid points
268 [xs, xt, Uh_l, Uh_r, ¬, ¬, ¬] = Diff_Cons_Common(N, dx, U, U, ...
        flag_spa_typ, flag_upw_typ, flag_scs_typ);
269
270 % !!! Special case: handling Upwind schemes & WENO schemes (F_l(j + 1/2), ...
        F_n(j - 1/2))
271 if (flag_spa_typ == 1) || ((flag_spa_typ == 2) && (flag_scs_typ == 3))
272
273 xt = xt - 1;
274 % Shifting Uh_r from (j - 1/2) to (j + 1/2)
275 for j = xs : xt
276 Uh_r(j, :) = Uh_r(j + 1, :);
277 end
278
279 end
280
281 for j = xs : xt
282
283 % Step 2: Cal. the Roe average value bar(U) with Roe formula
284 rho_l = Uh_l(j, 1);
285 u_l = Uh_l(j, 2) / Uh_l(j, 1);
286 E_l = Uh_l(j, 3);
287 T_l = ((E_l / rho_l) - (0.5 * u_l * u_l)) / Cv;
288 p_l = rho_l * R * T_l;
289 % c_l = sqrt(Gamma * p_l / rho_l);
290 H_l = (0.5 * u_l * u_l) + (Cp * T_l);  % Or H = (E + p) / rho
291
292 % Cal. F(Ul)
293 Fh_l(j, 1) = rho_l * u_l;
294 Fh_l(j, 2) = (rho_l * u_l * u_l) + p_l;
295 Fh_l(j, 3) = u_l * (E_l + p_l);
296
297 rho_r = Uh_r(j, 1);
298 u_r = Uh_r(j, 2) / Uh_r(j, 1);
299 E_r = Uh_r(j, 3);
300 T_r = ((E_r / rho_r) - (0.5 * u_r * u_r)) / Cv;
301 p_r = rho_r * R * T_r;
302 % c_r = sqrt(Gamma * p_r / rho_r);
303 H_r = (0.5 * u_r * u_r) + (Cp * T_r);
```

```
304
305 % Cal. F(Ur)
306 Fh_r(j, 1) = rho_r * u_r;
307 Fh_r(j, 2) = (rho_r * u_r * u_r) + p_r;
308 Fh_r(j, 3) = u_r * (E_r + p_r);
309
310 % Cal. ave. bar(U) martix
311 rho_ave = ((sqrt(rho_l) + sqrt(rho_r)) / 2)^2;
312 u_ave = ((sqrt(rho_l) * u_l + sqrt(rho_r) * u_r) / (2 * sqrt(rho_ave)));
313 H_ave = ((sqrt(rho_l) * H_l + sqrt(rho_r) * H_r) / (2 * sqrt(rho_ave)));
314 p_ave = ((Gamma - 1) / Gamma) * ((rho_ave * H_ave) - (0.5 * rho_ave * ...
        u_ave * u_ave));
315 c_ave = sqrt((Gamma - 1) * (H_ave - (0.5 * u_ave * u_ave)));
316 E_ave = (rho_ave * H_ave) - p_ave;
317
318 U_ave(j, 1) = rho_ave;
319 U_ave(j, 2) = rho_ave * u_ave;
320 U_ave(j, 3) = E_ave;
321
322 % Cal. ave. F(bar(U)) martix
323 F_ave(j, 1) = rho_ave * u_ave;
324 F_ave(j, 2) = (rho_ave * u_ave * u_ave) + p_ave;
325 F_ave(j, 3) = u_ave * (E_ave + p_ave);
326
327 % Cal. the Jacobian matrix A(bar(U)) using the relation F(U) = AU ...
        (Invalid! Jacobian quality fails!)
328 % Or using the direct cal. (Valid)
329 % A_ave = (F_ave(j, :)') / (U_ave(j, :)'); % Martix cal. failed
330
331 A_ave(1, 1) = 0;
332 A_ave(1, 2) = 1;
333 A_ave(1, 3) = 0;
334 A_ave(2, 1) = (-1) * ((3 - Gamma) / 2) * u_ave * u_ave;
335 A_ave(2, 2) = (3 - Gamma) * u_ave;
336 A_ave(2, 3) = Gamma - 1;
337 A_ave(3, 1) = (((Gamma - 2) / 2) * u_ave * u_ave * u_ave) - ((u_ave * ...
        c_ave * c_ave) / (Gamma - 1));
338 A_ave(3, 2) = ((c_ave * c_ave) / (Gamma - 1)) + (((3 - Gamma) / 2) * ...
        u_ave * u_ave);
```

```
339 A_ave(3, 3) = Gamma * u_ave;
340
341 % Step 3: Cal. the Jacobian matrix A(bar(U))
342 [V, G] = eig(A_ave);
343 S = inv(V);
344 % A_ave = inv(S) * G * S;  % Confirm
345
346 % Step 4: Cal. the absolute Jacobian matrix abs(A(bar(U)))
347 G_abs = zeros(3, 3);
348 % Entropy correction
349 for i = 1 : 3
350 if (abs(G(i, i)) > em)
351 G_abs(i, i) = abs(G(i, i));
352 else
353 G_abs(i, i) = ((G(i, i) * G(i, i)) + (em * em)) / (2 * em);
354 end
355 end
356 A_ave_abs = inv(S) * G_abs * S;
357
358 % Step 5: Cal. F(j + 1/2)
359 Fh(j, :) = ((0.5 * (Fh_r(j, :)' + Fh_l(j, :)')) - (0.5 * A_ave_abs * ...
        (Uh_r(j, :)' - Uh_l(j, :)')))';
360
361 end
362
363 xs_new = xs + 1;
364 xt_new = xt;
365
366 for j = xs_new : xt_new
367
368 % Step 6: Cal. the spatial derivative Fx_j
369 Fx(j, :) = (Fh(j, :) - Fh(j - 1, :)) / dx;
370
371 end
372
373 end
374
375 end
376
```

```
377
378 % Function Module: the approach to cal. difference for Fx from F_p and ...
      F_n with conservation form
379 % Note: the upwind schemes are converted into conservative form
380
381 function [xs_new, xt_new, Fh_p, Fh_n, Fx, Fx_p, Fx_n] = ...
      Diff_Cons_Common(N, dx, F_p, F_n, flag_spa_typ, flag_upw_typ, ...
      flag_scs_typ)
382
383 Fx = zeros(N, 3);    % invisid term
384 Fx_p = zeros(N, 3);  % pos. invisid term
385 Fx_n = zeros(N, 3);  % neg. invisid term
386
387 Fh_p = zeros(N, 3); % half point (j + 1/2) pos. invisid vector
388 Fh_n = zeros(N, 3); % half point (j + 1/2) neg. invisid vector
389 Fh = zeros(N, 3); % half point (j + 1/2) invisid vector
390
391 % Step 5: Cal. the flux derivative with different schemes (& ...
      shock-capturing)
392 if (flag_spa_typ == 1)
393 % 1 - General Upwind & Compact Schemes (forward / backward)
394
395 if (flag_upw_typ == 1)
396 % 1 - 1_od upwind scheme (2 points)
397 ks = -1;           % the start corner index relative to j
398 kt = 0;
399 kn = kt - ks + 1;  % number of the coefficients
400 kp = (ks : kt);
401
402 a = [];
403 a(1) = -1;
404 a(2) = 1;
405
406 b = zeros((kn - 1), 1);
407 b(1) = (-1) * a(1);
408 for k = 2 : (kn - 1)
409 b(k) = b(k - 1) - a(k);
410 end
411
```

```
412  elseif (flag_upw_typ == 2)
413  % 2 - 2_od upwind scheme (3 points)
414  ks = -2;
415  kt = 0;
416  kn = kt - ks + 1;
417  kp = (ks : kt);
418
419  a = [];
420  a(1) = 1 / 2;
421  a(2) = -4 / 2;
422  a(3) = 3 / 2;
423
424  b = zeros((kn - 1), 1);
425  b(1) = (-1) * a(1);
426  for k = 2 : (kn - 1)
427  b(k) = b(k - 1) - a(k);
428  end
429
430  elseif (flag_upw_typ == 3)
431  % 3 - 3_od upwind scheme (4 points with bias)
432  ks = -2;
433  kt = 1;
434  kn = kt - ks + 1;  % number of the coefficients
435  kp = (ks : kt);
436
437  a = [];
438  a(1) = 1 / 6;
439  a(2) = -6 / 6;
440  a(3) = 3 / 6;
441  a(4) = 2 / 6;
442
443  b = zeros((kn - 1), 1);
444  b(1) = (-1) * a(1);
445  for k = 2 : (kn - 1)
446  b(k) = b(k - 1) - a(k);
447  end
448
449  elseif (flag_upw_typ == 4)
450  % 4 - 5_od upwind scheme (6 points with bias)
```

```
451 ks = -3;
452 kt = 2;
453 kn = kt - ks + 1;  % number of the coefficients
454 kp = (ks : kt);
455
456 a = [];
457 a(1) = -2 / 60;
458 a(2) = 15 / 60;
459 a(3) = -60 / 60;
460 a(4) = 20 / 60;
461 a(5) = 30 / 60;
462 a(6) = -3 / 60;
463
464 b = zeros((kn - 1), 1);
465 b(1) = (-1) * a(1);
466 for k = 2 : (kn - 1)
467 b(k) = b(k - 1) - a(k);
468 end
469
470 end
471
472 % [Core algorithm]
473 % Uniform cal. procedure according to the coeff.
474 xs = 1 + abs(kp(2));
475 xt = N - abs(kp(2));
476 for j = xs : xt
477 for k = 1 : (kn - 1)
478 Fh_p(j, :) = Fh_p(j, :) + b(k) * F_p(j + kp(k + 1), :);  % F+ (j + 1/2)
479 Fh_n(j, :) = Fh_n(j, :) + b(k) * F_n(j - kp(k + 1), :);  % F- (j - 1/2)  ...
        different points?
480 end
481 end
482
483 xs_new = xs + 1;
484 xt_new = xt - 1;
485 for j = xs_new : xt_new
486 Fx_p(j, :) = (Fh_p(j, :) - Fh_p(j - 1, :)) / dx;
487 Fx_n(j, :) = (Fh_n(j + 1, :) - Fh_n(j, :)) / dx;
488 Fx(j, :) = Fx_p(j, :) + Fx_n(j, :);
```

```
489 end
490
491 elseif (flag_spa_typ == 2)
492 % 2 - Special Shock-Capturing Schemes
493
494 if (flag_scs_typ == 1)
495 % 1 - TVD - Total Variation Diminishing Scheme (Van Leer Limiter)
496 xs = 2;
497 xt = N - xs;
498
499 em = 1e-5;
500
501 for j = xs : xt
502 % Using Van Leer limiter (or other limiters)
503 r_p = (F_p(j, :) - F_p(j - 1, :)) ./ (F_p(j + 1, :) - F_p(j, :) + em);     ...
      % NaN? em!
504 r_n = (F_n(j + 2, :) - F_n(j + 1, :)) ./ (F_n(j + 1, :) - F_n(j, :) + em);
505 Phi_p = (r_p + abs(r_p)) ./ (1 + r_p);
506 Phi_n = (r_n + abs(r_n)) ./ (1 + r_n);
507
508 Fh_p(j, :) = F_p(j, :) + 0.5 * (Phi_p .* (F_p(j + 1, :) - F_p(j, :)));
509 Fh_n(j, :) = F_n(j + 1, :) - 0.5 * (Phi_n .* (F_n(j + 1, :) - F_n(j, :)));
510 end
511
512 % start point revision?
513 xs_new = xs + 1;
514 xt_new = xt;
515
516 for j = xs_new : xt_new
517 Fx_p(j, :) = (Fh_p(j, :) - Fh_p(j - 1, :)) / dx;
518 Fx_n(j, :) = (Fh_n(j, :) - Fh_n(j - 1, :)) / dx;
519 Fx(j, :) = Fx_p(j, :) + Fx_n(j, :);
520 end
521
522 elseif (flag_scs_typ == 2)
523 % 2 - NND - Non-oscillatory, Non-free-paremeters Dissipative Difference ...
      Scheme
524 xs = 2;
525 xt = N - xs;
```

```
526
527 for j = xs : xt
528 Fh_p(j, :) = F_p(j, :) +      (0.5 * Cal_Minmod((F_p(j, :) - F_p(j - 1, ...
        :)), (F_p(j + 1, :) - F_p(j, :))));
529 Fh_n(j, :) = F_n(j + 1, :) - (0.5 * Cal_Minmod((F_n(j + 1, :) - F_n(j, ...
        :)), (F_n(j + 2, :) - F_n(j + 1, :))));
530 Fh(j, :) = Fh_p(j, :) + Fh_n(j, :);
531 end
532
533 xs_new = xs + 1;
534 xt_new = xt;
535
536 for j = xs_new : xt_new
537 Fx(j, :) = (Fh(j, :) - Fh(j - 1, :)) / dx;
538 end
539
540 elseif (flag_scs_typ == 3)
541 % 3 - WENO - Weighted Essentially Non-Oscillatory Method
542 % (Jiang & Shu, 1996) 5 order WENO scheme
543
544 % Parameters
545 C = zeros(1, 3);
546
547 C(1) = 1 / 10;
548 C(2) = 6 / 10;
549 C(3) = 3 / 10;
550
551 p = 2;
552 em = 1e-6;
553
554 % a > 0 case
555 beta_p = zeros(3, 3);
556 alpha_p = zeros(3, 3);
557 omega_p = zeros(3, 3); % Weights
558 Fh_p_c = zeros(3, 3);
559
560 % Fh_p_1 = zeros(N, 3);  % Stencil 1
561 % Fh_p_2 = zeros(N, 3);  % Stencil 2
562 % Fh_p_3 = zeros(N, 3);  % Stencil 3
```

```
563 Fh_p = zeros(N, 3);  % Sum of weighted stencils (number = 3)
564
565 % a < 0 case
566 beta_n = zeros(3, 3);
567 alpha_n = zeros(3, 3);
568 omega_n = zeros(3, 3); % Weights
569 Fh_n_c = zeros(3, 3);
570 Fh_n = zeros(N, 3);  % Sum of weighted stencils (number = 3)
571
572 xs = 3;
573 xt = N - xs + 1;
574
575 for j = xs : xt
576
577 % a > 0, pos. flux case
578 % Cal. weights of each stencil !!!
579 beta_p(1, :) = ((1 / 4) * (F_p(j - 2, :) - 4 * F_p(j - 1, :) + 3 * F_p(j, ...
        :)).^2) + ((13 / 12) * (F_p(j - 2, :) - 2 * F_p(j - 1, :) + F_p(j, ...
        :)).^2);
580 beta_p(2, :) = ((1 / 4) * (F_p(j - 1, :) - F_p(j + 1, :)).^2) + ((13 / ...
        12) * (F_p(j - 1, :) - 2 * F_p(j, :) + F_p(j + 1, :)).^2);
581 beta_p(3, :) = ((1 / 4) * (3 * F_p(j, :) - 4 * F_p(j + 1, :) + F_p(j + 2, ...
        :)).^2) + ((13 / 12) * (F_p(j, :) - 2 * F_p(j + 1, :) + F_p(j + 2, ...
        :)).^2);
582
583 for k = 1 : 3
584 alpha_p(k, :) = C(k) ./ ((em + beta_p(k, :)).^p);
585 end
586 alpha_p_sum = sum(alpha_p);
587
588 for k = 1 : 3
589 omega_p(:, k) = alpha_p(:, k) / alpha_p_sum(k); % !!! notice orders
590 end
591
592 % Construct stencils !!! (F+ (j + 1/2))
593 Fh_p_c(1, :) = ((1 / 3) * F_p(j - 2, :)) - ((7 / 6) * F_p(j - 1, :)) + ...
        ((11 / 6) * F_p(j, :));
594 Fh_p_c(2, :) = ((-1) * (1 / 6) * F_p(j - 1, :)) + ((5 / 6) * F_p(j, :)) + ...
        ((1 / 3) * F_p(j + 1, :));
```

```
595 Fh_p_c(3, :) = ((1 / 3) * F_p(j, :)) + ((5 / 6) * F_p(j + 1, :)) - ((1 / ...
        6) * F_p(j + 2, :));
596
597 % omega_p' * Fh_p_c
598 omega_p_c = omega_p';
599
600 % Sum up all the stencils with weight to obtain F+ (j + 1/2)
601 for k = 1 : 3
602 Fh_p(j, k) = (omega_p_c(k, :) * Fh_p_c(:, k));
603 end
604
605 % a < 0, neg. flux case
606 % Cal. weights of each stencil !!!
607 beta_n(1, :) = ((1 / 4) * (F_n(j + 2, :) - 4 * F_n(j + 1, :) + 3 * F_n(j, ...
        :)).^2) + ((13 / 12) * (F_n(j + 2, :) - 2 * F_n(j + 1, :) + F_n(j, ...
        :)).^2);
608 beta_n(2, :) = ((1 / 4) * (F_n(j + 1, :) - F_n(j - 1, :)).^2) + ((13 / ...
        12) * (F_n(j + 1, :) - 2 * F_n(j, :) + F_n(j - 1, :)).^2);
609 beta_n(3, :) = ((1 / 4) * (3 * F_n(j, :) - 4 * F_n(j - 1, :) + F_n(j - 2, ...
        :)).^2) + ((13 / 12) * (F_n(j, :) - 2 * F_n(j - 1, :) + F_n(j - 2, ...
        :)).^2);
610
611 for k = 1 : 3
612 alpha_n(k, :) = C(k) ./ ((em + beta_n(k, :)).^p);
613 end
614 alpha_n_sum = sum(alpha_n);
615
616 for k = 1 : 3
617 omega_n(:, k) = alpha_n(:, k) / alpha_n_sum(k);
618 end
619
620 % Construct stencils !!! (F- (j - 1/2))
621 Fh_n_c(1, :) = ((1 / 3) * F_n(j + 2, :)) - ((7 / 6) * F_n(j + 1, :)) + ...
        ((11 / 6) * F_n(j, :));
622 Fh_n_c(2, :) = ((-1) * (1 / 6) * F_n(j + 1, :)) + ((5 / 6) * F_n(j, :)) + ...
        ((1 / 3) * F_n(j - 1, :));
623 Fh_n_c(3, :) = ((1 / 3) * F_n(j, :)) + ((5 / 6) * F_n(j - 1, :)) - ((1 / ...
        6) * F_n(j - 2, :));
624
```

```
625 % omega_p' * Fh_p_c
626 omega_n_c = omega_n';
627
628 % Sum up all the stencils with weight to obtain F- (j - 1/2)
629 for k = 1 : 3
630 Fh_n(j, k) = (omega_n_c(k, :) * Fh_n_c(:, k));
631 end
632
633 end
634
635 xs_new = xs + 1;
636 xt_new = xt - 1;
637
638 for j = xs_new : xt_new
639 Fx_p(j, :) = (Fh_p(j, :) - Fh_p(j - 1, :)) / dx;
640 Fx_n(j, :) = (Fh_n(j + 1, :) - Fh_n(j, :)) / dx;
641 Fx(j, :) = Fx_p(j, :) + Fx_n(j, :);
642 end
643
644 end
645
646 end
647
648 end
649
650
651 % Function Module: Cal. minmod(a, b)
652 % Or minmod(a, b) = 0.5 * (sign(a) + siagn(b)) * min(abs(a), abs(b))
653 % a, b is 1-D array with same length
654
655 function Q = Cal_Minmod(a, b)
656
657 Np = length(a);
658 Q = zeros(1, Np);
659
660 for i = 1 : Np
661 if ((a(i) * b(i)) > 0)
662 if (abs(a(i)) > abs(b(i)))
663 Q(i) = b(i);
```

```
664  else
665  Q(i) = a(i);
666  end
667  else
668  Q(i) = 0;
669  end
670  end
671
672
673  end
674
675  function [data] = analytic_sod(t)
676  %to solve Sod's Shock Tube problem
677  %reference: "http://www.phys.lsu.edu/¬tohline/PHYS7412/sod.html"
678  %    |        |    |      |          |
679  %    |        |    |      |          |
680  %    |        |    |      |          |
681  %____|________|____|______|__________|_______________
682  %    x1       x2   x0     x3         x4
683  %
684  %input require: t (time)
685  if nargin < 1
686  %set default value
687  t = 0.2;
688  end
689  %Initial conditions
690  x0 = 0;
691  rho_l = 1;
692  P_l = 1;
693  u_l = 0;
694
695  rho_r = 0.125;
696  P_r = 0.1;
697  u_r = 0;
698
699  gamma = 1.4;
700  mu = sqrt( (gamma-1)/(gamma+1) );
701
702  %speed of sound
```

```
703 c_l = power( (gamma*P_l/rho_l),0.5);
704 c_r = power( (gamma*P_r/rho_r),0.5);
705
706 P_post = fzero('sod_func',pi);
707 v_post = 2*(sqrt(gamma)/(gamma - 1))*(1 - power(P_post, (gamma - ...
        1)/(2*gamma)));
708 rho_post = rho_r*(( (P_post/P_r) + mu^2 )/(1 + mu*mu*(P_post/P_r)));
709 v_shock = v_post*((rho_post/rho_r)/( (rho_post/rho_r) - 1));
710 rho_middle = (rho_l)*power((P_post/P_l),1/gamma);
711
712 %Key Positions
713 x1 = x0 - c_l*t;
714 x3 = x0 + v_post*t;
715 x4 = x0 + v_shock*t;
716 %determining x2
717 c_2 = c_l - ((gamma - 1)/2)*v_post;
718 x2 = x0 + (v_post - c_2)*t;
719
720 %start setting values
721 n_points = 1000;     %set by user
722 %boundaries (can be set)
723 x_min = -0.5;
724 x_max = 0.5;
725
726 x = linspace(x_min,x_max,n_points);
727 data.x = x';
728 data.rho = zeros(n_points,1);   %density
729 data.P = zeros(n_points,1); %pressure
730 data.u = zeros(n_points,1); %velocity
731 data.e = zeros(n_points,1); %internal energy
732
733 for index = 1:n_points
734 if data.x(index) < x1
735 %Solution b4 x1
736 data.rho(index) = rho_l;
737 data.P(index) = P_l;
738 data.u(index) = u_l;
739 elseif (x1 ≤ data.x(index) && data.x(index) ≤ x2)
740 %Solution b/w x1 and x2
```

```
741  c = mu*mu*((x0 - data.x(index))/t) + (1 - mu*mu)*c_l;
742  data.rho(index) = rho_l*power((c/c_l),2/(gamma - 1));
743  data.P(index) = P_l*power((data.rho(index)/rho_l),gamma);
744  data.u(index) = (1 - mu*mu)*( (-(x0-data.x(index))/t) + c_l);
745  elseif (x2 ≤ data.x(index) && data.x(index) ≤ x3)
746 %Solution b/w x2 and x3
747  data.rho(index) = rho_middle;
748  data.P(index) = P_post;
749  data.u(index) = v_post;
750  elseif (x3 ≤ data.x(index) && data.x(index) ≤ x4)
751 %Solution b/w x3 and x4
752  data.rho(index) = rho_post;
753  data.P(index) = P_post;
754  data.u(index) = v_post;
755  elseif x4 < data.x(index)
756 %Solution after x4
757  data.rho(index) = rho_r;
758  data.P(index) = P_r;
759  data.u(index) = u_r;
760  end
761  data.e(index) = data.P(index)/((gamma - 1)*data.rho(index));
762  end
763  end
764
765  function y = sod_func(P)
766 %defines function to be used in analytic_sod
767 %Initial conditions
768  rho_l = 1;
769  P_l = 1;
770  u_l = 0;
771
772  rho_r = 0.125;
773  P_r = 0.1;
774  u_r = 0;
775
776  gamma = 1.4;
777
778  mu = sqrt( (gamma-1)/(gamma+1) );
779
```

```
780  y = (P - P_r)*(( ((1 - mu^2)^2)*((rho_r*(P + mu*mu*P_r))^-1) )^(0.5)) ...
781  - 2*(sqrt(gamma)/(gamma - 1))*(1 - power(P, (gamma - 1)/(2*gamma)));
782  end
```