周吕文 201128000718065
物理学院 20110308 班
计算报告
4/15/2012

一维激波管问题的数值计算

1 引言

激波管是一根两端封闭, 中间用一薄膜隔开成两部分的柱形长管, 薄膜两侧分别充满低压和高压气体. 薄膜瞬时突然破裂, 气体将从高压端冲向低压端, 同时在管内形成激波, 稀疏波和接触间断等复杂波系.

历史上第一根激波管是由法国化学家 P. 维埃耶在 19 世纪末为研究矿井中的爆炸制成. “激波管” 一词最早出现在 1946 年美国 W. 布利克尼的研究报告中. 激波管早期主要应用于燃烧, 爆炸和非定常波运动的研究以及压力传感器的标定等. 1950 年以来, 由于研制洲际导弹和核武器的需要, 激波管得到了蓬勃发展. 激波管结构简单, 使用方便而且价格低廉, 能提供范围宽广的实验参量, 因此得到广泛的应用. 例如, 在空气动力学, 气体物理学, 化学动力学和航空声学的研究中都广泛地使用激波管. 近来, 激波管又开始在气体激光, 环境科学和能源科学的研究中发挥作用. 为满足导弹, 核武器等的发展需要, 研制出了多种多样的激波管, 并产生了诸如激波风洞等多种新型实验装置.

2 一维激波管问题

一维激波管问题, 又称一维 Riemann 问题, 是一个非常经典的算例. 其中包括激波, 膨胀波, 接触间断等典型的物理现象, 如图1所示. 在 $t = 0$ 时, 由一隔膜两侧气体静止, 但压力, 密度不同. 在本文中隔膜两

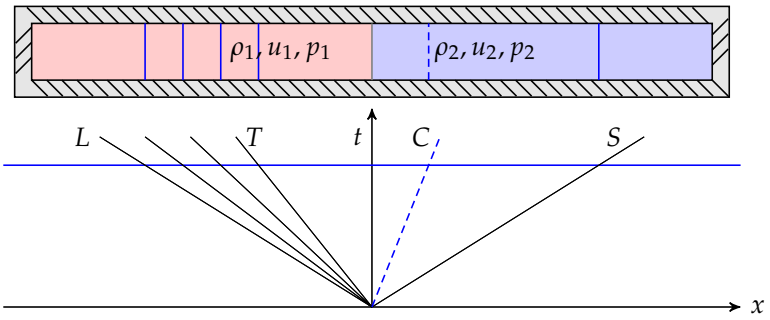

图 1: 一维 Riemann 问题示意图

侧气体的压力, 密度分别为

$$\rho_1 = 0.125 \quad p_1 = 1.0 \quad u_1 = 0$$
$$\rho_2 = 1.000 \quad p_2 = 1.0 \quad u_2 = 0$$

其中, ρ, p, u 均为无量纲量. 在 $t = 0$ 时刻, 隔膜瞬时突然破裂, 左侧高压气体开始向右侧流动, 在右侧形成激波 OS, 在左侧的 OL 和 OT 之间形成膨胀波, 其中 OL 为膨胀波的前缘, OT 为膨胀波的后缘. 在激波和膨胀波之间有一接触间断 OC, 即在 OC 两侧, 气体的压力, 速度相等, 但密度不同.

3 物理模型

设一维激波管问题中气体是理想气体. 一维激波管问题在数学上可以用一维可压缩无黏气体 Euler 方程组来描述. 在在直角坐标系下无量纲的一维 Euler 方程组为:

$$\frac{\partial \mathbf{w}}{\partial t} + \frac{\partial \mathbf{f}}{\partial x} = 0,\ 0 \le x \le 1.0 \tag{1}$$

其中

$$\mathbf{w} = \begin{bmatrix} \rho \\ \rho u \\ E \end{bmatrix},\ \mathbf{f} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ (E + p)u \end{bmatrix}$$

这里 ρ 是流体的密度, u 是流体的速度, p 是流体的压力, E 是流体单位体积总能:

$$E = \left(e + \frac{1}{2}u^2\right)$$

其中 e 为比内能 (单位质量物质的内能), 对于理想气体有

$$e = \frac{p}{(\gamma-1)\rho} \implies p = (\gamma-1)\rho e = (\gamma-1)\Big[E - \frac{1}{2}\rho u^2\Big]$$

对于理想气体, 式 (1) 的 Jacobian 系数矩阵为

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{1}{2}(\gamma-3)u^3 & (3-\gamma)u & \gamma-1 \\ u\big(\frac{1}{2}(\gamma-1)u^2 - H\big) & H-(\gamma-1)u^2 & \gamma u \end{bmatrix}$$

其中 H 为总比焓 $H = \frac{H+p}{\rho} = \frac{c^2}{\gamma-1} + \frac{1}{2}u^2$. 声速为

$$c = \sqrt{\frac{\gamma p}{\rho}} = \sqrt{\frac{\gamma}{\rho}(\gamma-1)(E - \frac{1}{2}\rho u^2)}$$

矩阵 $\mathbf{A}$ 的特征值为

$$\lambda_1 = u - c, \quad \lambda_2 = u, \quad \lambda_3 = u + c$$

对应的特征向量为

$$\mathbf{e_1} = \begin{bmatrix} 1 \\ u-c \\ H-uc \end{bmatrix}, \ \mathbf{e_2} = \begin{bmatrix} 1 \\ u \\ \frac{1}{2}u^2 \end{bmatrix}, \ \mathbf{e_3} = \begin{bmatrix} 1 \\ u+c \\ H+uc \end{bmatrix}$$

一维 Riemann 问题具有精确解. 因此可以用精确解来校验本文后面的数值算法的正确性和精确度. 对于本文的具体问题

- 初始条件: $\rho(x > 0.5) = 0.125$, $p(x > 0.5) = 1.0$, $u(x > 0.5) = 0$; $\rho(x < 0.5) = 1.0$, $p(x < 0.5) = 1.0$, $u(x < 0.5) = 0$.
- 边界条件: 在 $x = 0$ 和 $x = 1$ 处为反射边界条件, $u(x = 0) = u(x = 1) = 0$.

这个具体的算例在时间 $t = 0.20$ 的精确如图2所示.

4 计算方案

为得到一维激波管问题的数值解, 本文分别用两种差分格式对式 (1) 进行逼近. 本文要讨论的第一种差分格是 Lax 差分格式, 它在时间和空间上都是一阶精度; 本文要讨论的第二种差分格式是蛙跳差分格式. 它在时间和空间上都是二阶精度.

4.1 Lax 差分格式

Lax 差分格式是计算流体力学中提出和应用最早的, 非常著名的守恒型差分格式. 它在计算流体力学发展初期得到广泛应用, 在历史上曾起过十分重要的作用. Lax 单步差分格式的差分方程为

$$\mathbf{w}_j^{n+1} = \frac{1}{2}(\mathbf{w}_{j+1}^n + \mathbf{w}_{j-1}^n) - \frac{1}{2}r(\mathbf{f}_{j+1}^n - \mathbf{f}_{j-1}^n)$$

其中 $r = \Delta t/\Delta x$. 将上式作简单变型

$$\mathbf{w}_j^{n+1} = \mathbf{w}_j^n - \frac{\Delta t}{\Delta x}\frac{\mathbf{f}_{j+1}^n - \mathbf{f}_{j-1}^n}{2} + \frac{1}{2}(\mathbf{w}_{j+1}^n - 2\mathbf{w}_j^n - \mathbf{w}_{j-1}^n)$$

实际上, 上式等同于以下微分方程

$$\frac{\partial \mathbf{w}}{\partial t} + \frac{\mathbf{f}}{\partial x} = -\frac{1}{2}\frac{\partial^2 w}{\partial t^2}\Delta t + \frac{1}{2}\frac{\partial^2 w}{\partial x^2}\frac{\Delta x^2}{\Delta t} + O(\Delta t^2, \Delta x^2)$$

截断误差为 $O(\Delta t, \Delta x^2/\Delta t)$. 在时间和空间上都是一阶精度. Lax 差分格式是一个非线性守恒型差分格式. 对它进行线性化后, 可得到它的稳定性条件 $|\mathbf{w}|_{\max}\Delta t/\Delta x \le 1$.

以往计算实践表明, 由于 Lax 差分格式的黏性较大, 计算精度不高, 在计算激波时, 激波会被拉宽和抹平, 一般都不采用 Lax 差分格式计算激波.

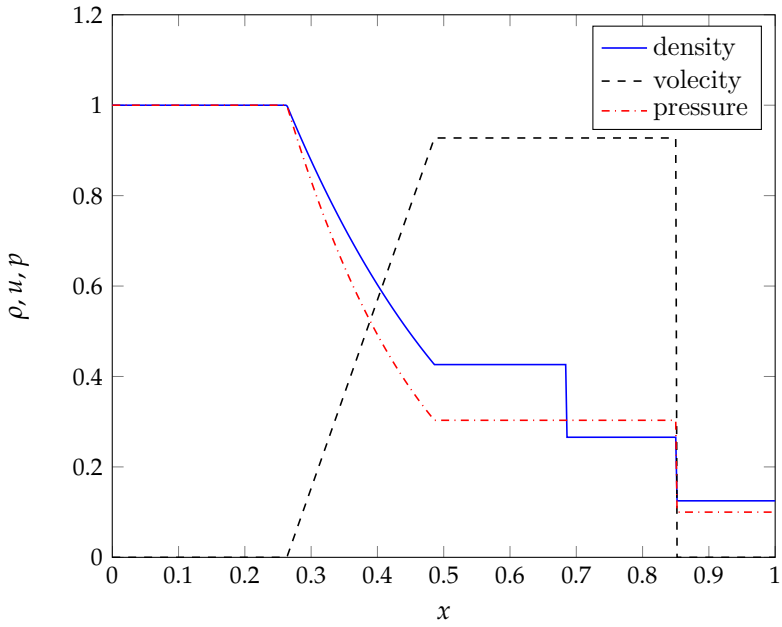

图 2: 一维激波管问题在 $t = 0.20$ 时的精确解: 密度 ρ, 压强 p, 速度 u 空间分布

4.2 蛙跳差分格式

蛙跳差分格式在时间方向和空间方向都用中心差分. 其差分方程为

$$\mathbf{w}_j^{n+1} = \mathbf{w}_j^{n-1} - r\left(\mathbf{f}_{j+1}^n - \mathbf{f}_{j-1}^n\right)$$

其中 $r = \Delta t/\Delta x$. 通过代入各项的泰勒展开可得

$$\frac{\partial \mathbf{w}}{\partial t} + \frac{\mathbf{f}}{\partial x} = -\frac{1}{6}\left(\frac{\partial^3 \mathbf{w}}{\partial t^3}\Delta t^2 + \frac{\partial^3 \mathbf{f}}{\partial x^3}\Delta x^2\right) + O(\Delta t^4, \Delta x^4)$$

截断误差为 $O(\Delta t^2, \Delta x^2)$, 蛙跳差分格式在时间和空间上都是二阶精度. 从截断误差可以看出蛙跳差分格式没有耗散项, 即零耗散, 但却存在较强的色散作用.

蛙跳格式是一个时间方向三层的格式. 在计算中, 需要知道 0,1 两个时间步的初值. 因此, 在初始时刻需要估算出第一个步的结果, 本文用 Lax 估出一步的结果, 再采用蛙跳差分格式对初始状态和第一步结果进行迭代.

4.3 人工黏性滤波方法

上面讨论的两种差分格式: Lax 在时间和空间上都是一阶精度, 具有数值耗散效应, 因此不用添加人工粘性; 蛙跳差分格式在时间和空间上都是二阶精度, 零耗散, 却存在较强的色散作用. 在用蛙跳差分格式对一维激波的计算中, 发现该格式非常不稳定, 因此在计算激波时, 必须采用人工黏性滤波方法:

$$\overline{\mathbf{w}}_j^n = \mathbf{w}_j^n + \frac{1}{2}\eta\left(\mathbf{w}_{j+1}^n - 2\mathbf{w}_j^n + \mathbf{w}_{j-1}^n\right)$$

为了在激波附近人工黏性起作用, 而在光滑区域人工黏性为零, 需要引入一个与密度 (或者压力) 相关的开关函数:

$$\mathrm{sw} = \left|\frac{|\rho_{i+1} - \rho_i| - |\rho_i - \rho_{i-1}|}{|\rho_{i+1} - \rho_i| + |\rho_i - \rho_{i-1}|}\right|$$

从上式可以看出, 在光滑区域, 密度变化很缓, 因此 sw 值也很小; 而在激波附近密度变化很陡, 值就很大. 带有开关函数的前置人工黏性滤波方法为:

$$\overline{\mathbf{w}}_j^n = \mathbf{w}_j^n + \frac{1}{2}\eta \cdot \mathrm{sw} \cdot \left(\mathbf{w}_{j+1}^n - 2\mathbf{w}_j^n + \mathbf{w}_{j-1}^n\right)$$

其中参数 η 往往需要通过实际试算来确定. 通过尝试, 发现 η 简单地取 1.0 即可得到效果较好的激波.

5 数值计算流程

采用 Lax 和蛙跳差分格式计算一维激波管问题流程相似, 附录 A 程序中的 Lax 和蛙跳差分格式只是作为一个共同主程序框架下的两个函数. 这里给出蛙跳差分格式计算一维激波管问题程序的流程图, 如图3. 需要注意的是, 在用蛙跳差分格式计算时, 第一步需要用 Lax 差分格式估算, 这一过程被封装在 subroutine: leapfrog 中, 因此图3中并没有体现这一过程.

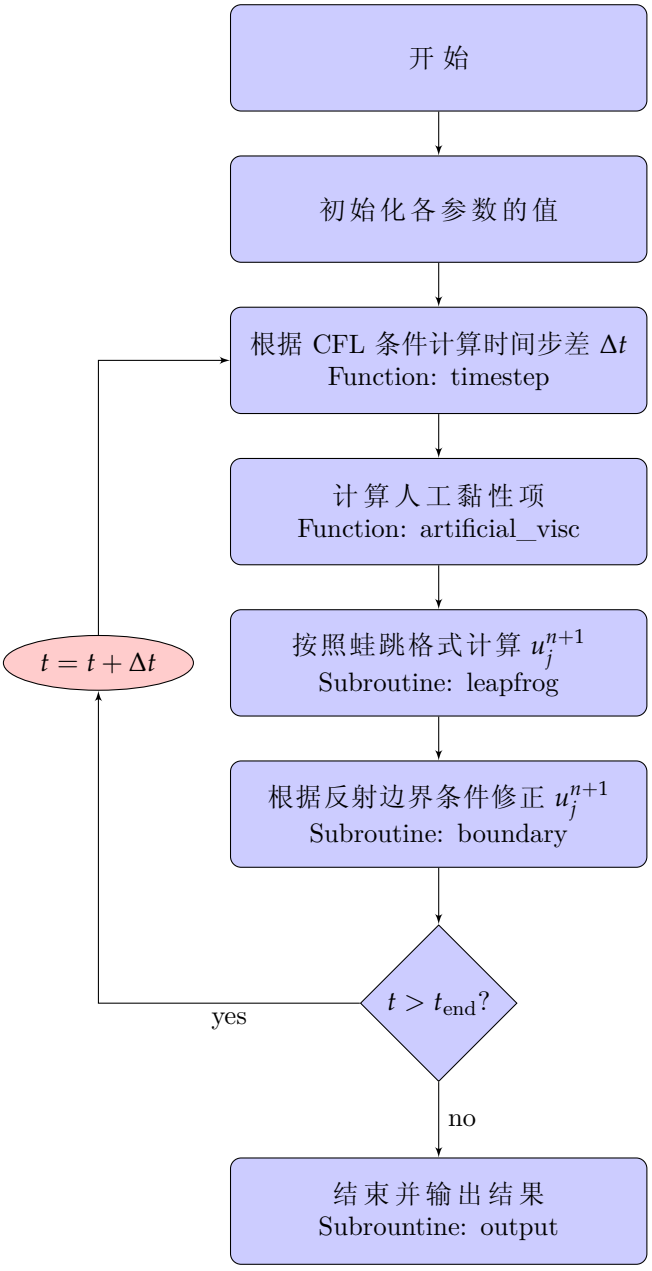

图 3: 采用蛙跳差分格式计算一维激波管问题程序流程图

6 计算结果

数值计算采用 Fortran90 语言编写程序, MatLab 作后处理 (作图), 具体代码见附录. 计算中网格数取 10^4(本文网格数取的较大, 取 500 即可), CFL 取 0.5, 计算总时间为 0.2. 计算得到在 $T = 0.2$ 时刻的密度, 速度和压力分布.

6.1 LAX 差分格式

由 LAX 差分格式得到的一维激波管问题的密度 ρ, 压强 p, 速度 u 空间分布如图4所示. 从图中可以看出, 与精确解相比 (图2), 在激波面和间断面处由于数值耗散效应的作用, 具有被拉宽的现象.

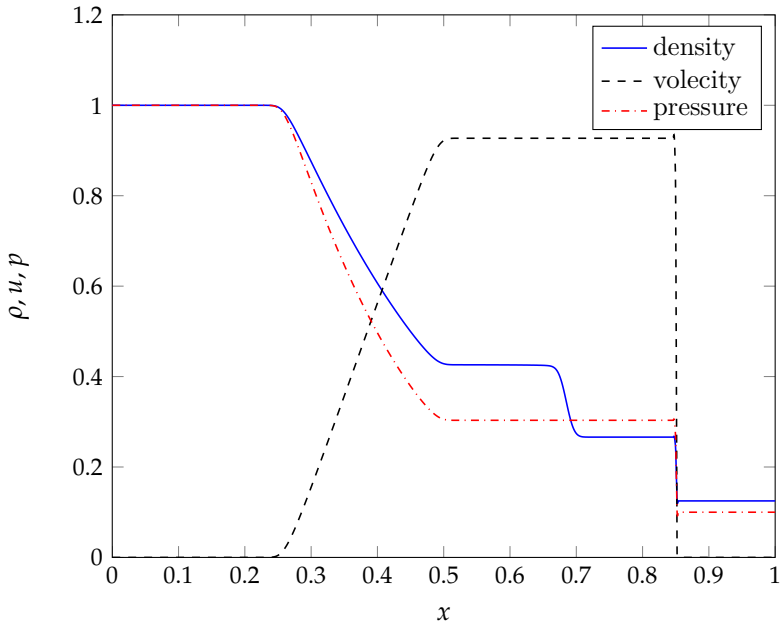

图 4: 采用 Lax 差分格式得到的一维激波管问题的密度 ρ, 压强 p, 速度 u 空间分布

图5是 Lax 差分格式得到的密度 ρ, 压强 p, 速度 u 场.

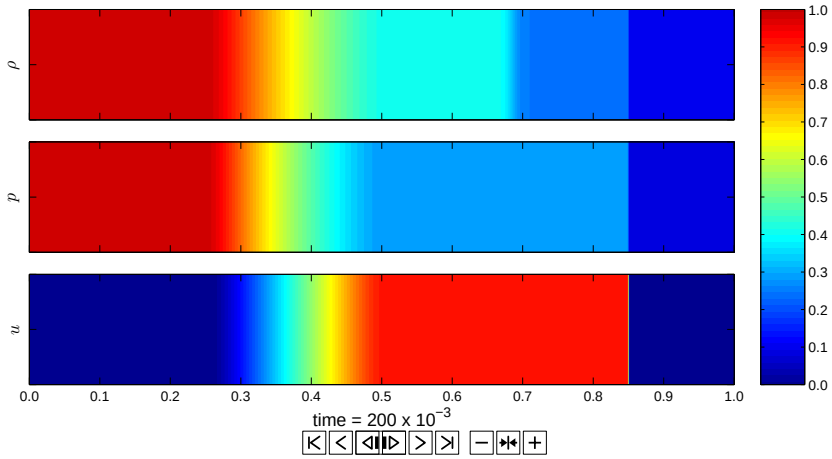

图 5: 采用 Lax 差分格式得到的密度 ρ, 压强 p, 速度 u 场

6.2 蛙跳差分格式

计算中发现, 在不采用人工黏性情况下, 当 Δx 非常小时 (网格点取的较大), 蛙跳差分格式极不稳定, 具有较强的色散, 甚至使程序无法运行 (当去掉蛙跳格式的人工黏性项后, 网格数必需取小至 1000 以内, 否则程序出错). 为了消除数值色散效应, 需要添加人工黏性, 图6是采用人工黏性的蛙跳差分格式得到的一维激波管问题的密度 ρ, 压强 p, 速度 u 空间分布. 从图中可以看出, 经过添加人工粘性后可以明显看到色散现象消失. 与精确解相比 (图2), 激波处的间断跳跃现象和激波位置模拟都比较准确. 图7是蛙跳差分格式得到的密度 ρ, 压强 p, 速度 u 场.

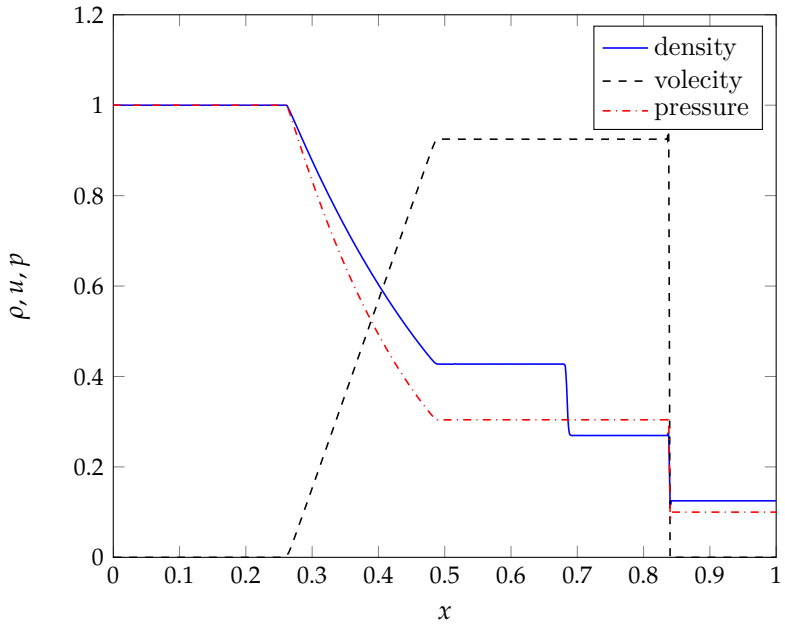

图 6: 采用蛙跳差分格式得到的一维激波管问题的密度 ρ, 压强 p, 速度 u 空间分布

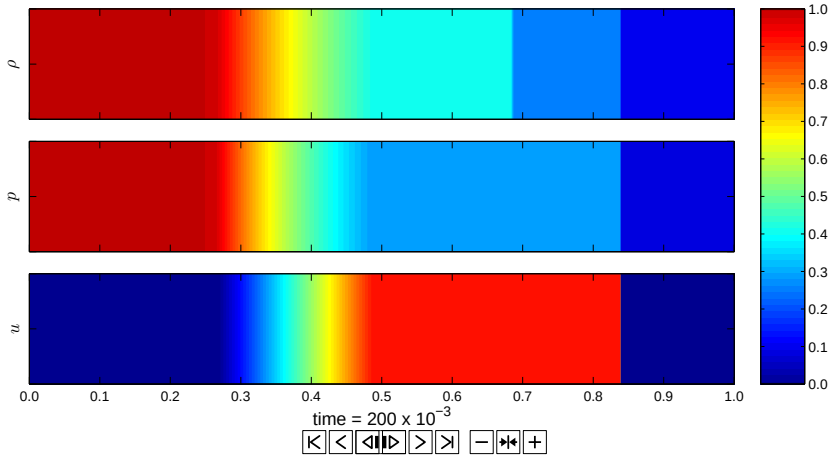

图 7: 用蛙跳差分格式得到的密度 ρ, 压强 p, 速度 u 场

7 结论

采用两种不同差分格式所得到的一维激波管问题的计算结果基本与精确解吻合. Lax 差分格式不需要加入人工黏性就能捕捉激波. 但由于数值耗散, 使计算得到的激波面并不是很陡, 精度并不是很高. 而蛙跳差分格式需要添加人工黏性项, 在人工黏性项辅助下, 能很好地捕捉激波, 计算得到的激波面很陡, 很窄, 计算激波精度是很高的. 采用带开关函数的前置人工滤波法能消除激波附近的非物理振荡, 计算效果很好.

从两种差分格式的结果都可以看出通过激波后气体的密度, 压力和速度都是增加的; 在压力分布中存在第二个台阶, 表明在这里存在一个接触间断, 在接触间断两侧压力是有间断的, 而密度和速度是相等的. 这个计算结果正确地反映了一维激波管问题的物理特性, 并被激波管实验所验证.

参考文献

[1] 张德良, 计算流体力学教程. 高等教育出版社. 北京, 2011.

[2] Shock tube. http://en.wikipedia.org/wiki/Shock_tube.

[3] Bernd Freytag, Naples. Introduction to Numerical Hydrodynamics.
www.astro.uu.se/ bf/course/numhd_course_20100124.pdf

[4] The Wave Equation and Staggered Leapfrog.
http://math.mit.edu/classes/18.086/2006/am53.pdf

[5] Justin Hudson. A Review on the Numerical Solution of the 1D Euler Equations.
http://eprints.ma.man.ac.uk/150/01/covered/MIMS_ep2006_9.pdf

附录

A shocktube.f90

```
! ============================================================
!   One-dimensional shock tube solved by Lax and leapflog scheme
! ------------------------------------------------------------
! At t = 0:
!       +----------------------
!       |                      |
!       |    rho1  u1  p1      |    rho2  u2  p2
!       |                      ----------------------
!       +--------------------+---------------------+--------> X
!       0                   L/2                    L
!
! Solve the Riemann problem. At both ends, the tube is closed
! so u = 0. Equations:
!       d(rho)  /dt + d(rho*u)      /dx = 0
!       d(rho*u)/dt + d(rho*u*u + p)/dx = 0
!       d(E)    /dt + d(u*(rho*E+p))/dx = 0
!
! ============================================================
! !!!!!!!!!!!!!!!!!!!!!!!! Output !!!!!!!!!!!!!!!!!!!!!!!!!!
! "solution.dat" = Data file containing for each grid point,
!                  coordinate, density, velocity, pressure,
!                  in the following format:
!
!              x(1)        rho(1)        u(1)        p(1)
!              x(2)        rho(2)        u(2)        p(2)
!               .            .            .           .
!               .            .            .           .
!              x(n)        rho(n)        u(n)        p(n)
!
! Use the matlab program, data2figure.m, to plot the solutions.
! ============================================================
! !!!!!!!!!!!!!!  READ THIS BEFORE YOU START !!!!!!!!!!!!!!!!!!
!
! W in the program is the matrix of conservative varibles: rho,
! rho*u and E stored as columns in the matrix W
!          _                                          _
!         |    rho_1        (rho*u)_1          E_1      |
!         |    rho_2        (rho*u)_2          E_2      |
!   W =   |      .              .                .       |
!         |      .              .                .       |
!         |_   rho_n        (rho*u)_n          E_n     _|
!
! F in the program is the matrix of flux: rho*u, rho*u**2+p and
! u*(rho*E+p) stored as columns in the matrix F
!          _                                          _
!         |  (rho*u)_1   (rho*u**2+p)_1   (u*(rho*E+p))_1  |
!         |  (rho*u)_2   (rho*u**2+p)_2   (u*(rho*E+p))_2  |
!   F =   |    .               .               .           |
!         |    .               .               .           |
!         |_ (rho*u)_n   (rho*u**2+p)_n   (u*(rho*E+p))_n _|
!
```

```
! rho: Density,  u: Velocity,  p : Pressure
!
! ================================================================
! !!!!!!!!!!!!!!!!  Record of revisions !!!!!!!!!!!!!!!!!!!!!!!!
!
!        Date        Programmer        Description of change
!       4/02/12      zhou lvwen            original code
!----------------------------------------------------------------
!

Program TubeShock

implicit none
! ================================================================
! !!!!!!!!!!!!!! Declare constant parameters !!!!!!!!!!!!!!!!!!!
integer, parameter    :: o     = 8    ! Double Precision
real(o), parameter    :: gamma = 1.4  ! Ratio of specific heats
real(o), parameter    :: L     = 1.0  ! Length of domain
real(o), parameter    :: CFL   = 0.5  ! CFL number for stability
real(o), parameter    :: Tend  = 0.2  ! final time
integer, parameter    :: n     = 10001! Number of grid points
real(o), parameter    :: dx = L/(n-1) ! Spatial step size
! ............... Define intial conditions ......................
! Right conditions:
real(o), parameter    :: p1    = 0.1  ! Pressure in right side
real(o), parameter    :: rho1  = 0.125! Density  at right side
real(o), parameter    :: u1    = 0.0  ! Velocity in right side
! Left conditions:
real(o), parameter    :: p2    = 1.0  ! Pressure in left  side
real(o), parameter    :: rho2  = 1.0  ! Density  at left  side
real(o), parameter    :: u2    = 0.0  ! Velocity in left  side

! ================================================================
! !!!!!!!!!! Declare variable types & definitions !!!!!!!!!!!!!!!
integer               :: i            ! loop index
integer               :: nsteps= 0    ! Number of time steps
real(o)               :: t     = 0.0  ! current time
real(o)               :: dt           ! time step
real(o),dimension(n)  :: x     = [(i*dx, i = 0, n-1)]
integer,dimension(n-2):: nc    = [(i, i= 2, n-1)]
! center point index  :: nc    = [2, 3, ..., n-1]
! left   point index  :: nc-1  = [1, 2, ..., n-2]
! right  point index  :: nc+1  = [3, 4, ..., n  ]
real(o),dimension(n)  :: p            ! Pressure corresponding x
real(o),dimension(n)  :: rho          ! Density  corresponding x
real(o),dimension(n)  :: u            ! Velocity corresponding x
real(o),dimension(n)  :: E            ! energy   corresponding x
real(o),dimension(n,3):: W            ! conservative varibles
real(o),dimension(n,3):: Wbefore      ! for leapfrog
real(o),dimension(n,3):: F            ! flux
real(o)               :: eta   = 1.0  ! Art_visc coefficient
character             :: scheme*10    ! Difference scheme
integer               :: process = 0  ! process index

! Initial conditions
p  (1:(n+1)/2) = p2  ;  p  ((n+3)/2:n) = p1  ;
rho(1:(n+1)/2) = rho2;  rho((n+3)/2:n) = rho1;
```

```
u  (1:(n+1)/2) = u2  ;  u  ((n+3)/2:n) = u1  ;
E = p/((gamma-1)*rho) + 0.5*u**2

! Prompted to select a difference scheme in the terminal
call Initialization(scheme)

Do while (t<Tend)

   ! Determine time step by CFL condition
   dt = timestep(CFL, dx)

   ! code the variables
   call u2WF(rho, u, p, E, W, F)

   !Add artificial viscosity
   W = W + artificial_visc(rho, W, eta)

   ! Use lax or leapfrog to update the solution
   if (scheme =='lax') then
      call Lax(W, F, dt)
   else if (scheme=='leapfrog') then
      call leapfrog(W, Wbefore, F, dt, nsteps)
   end if

   ! boundary conditions: u(0,t) = u(L,t) = 0
   call boundary(u)

   ! Decode the variables
   call W2u(W, rho, u, p, E)

   ! update the time and time steps
   t = t + dt
   nsteps = nsteps + 1

   ! draw the program running process bar
   call processBar(t, Tend, process)

End do



call output(t)

! End of program
! ************************************************************
! ************************************************************

contains
! Below is a list of subroutines and functions used in the main
! program.

! ************************************************************
subroutine Initialization(scheme)
! Print some information on the termianl about the program and
! Prompted to select a difference scheme in the terminal
implicit none
character,intent(inout):: scheme*10   ! Difference scheme
```

```
write(*,*), "------- One-dimensional shock tube problem -------"
write(*,*), " Copyrhigt by Zhou Lvwen: zhou.lv.wen[at]gmail.com"
write(*,*), "----------------- intial conditions ---------------"
write(*,*), '-Right conditions: '
write(*, '("   rho = ",f6.3, "  u = ",f6.3, "  p = ",f6.3)')&
                    rho1,            u1,            p1
write(*,*), '-Left  conditions: '
write(*, '("   rho = ",f6.3, "  u = ",f6.3, "  p = ",f6.3)')&
                    rho2,            u2,            p2
write(*,'(" -Length of domain:", f6.2)') , L
write(*,*)
write(*,*), "------------- computational parameters -----------"
write(*,*), 'The recommended parameters will be used'
write(*,'(" -CFL number for stability :", f5.2)') CFL
write(*,'(" -Number of grid points    :", i5  )') n
write(*,'(" -Final time               :", f5.2)') Tend
write(*,*)
write(*,*), "----------------- Difference scheme --------------"
1 write(*,'(A,$)'),  &
             ' Please select Difference scheme (lax/leapfrog): '
read(*,'(A)') scheme
select case (scheme)
   case ('lax')
       write(*,*), 'You select lax scheme'
   case ('leapfrog')
       write(*,*), 'You select leap-forg scheme'
   case default
       write(*,*), 'please input "lax" or "leapfrog"'; goto 1
end select
write(*,*)

end subroutine Initialization
! ------------------------------------------------------------

! ************************************************************
function timestep(CFL, dx) result(dt)
! Determine dt by CFL condition: dt = CFL * dx / max_wavespeed
!                                   CFL <= 1.0
!
implicit none
real(o), intent(in)   :: CFL          ! CFL number for stability
real(o), intent(in)   :: dx           ! Spatial step size
real(o)               :: dt           ! time step
real(o)               :: umax         ! maximal absolute speed
real(o),dimension(n)  :: c            ! speed of sound

! Find the speed of sound
  c = sqrt(gamma*p/rho);

! Compute maximal absolute value of the characteristic speeds
  umax = maxval(abs(u+c));

! Determine the time step using the CFL condition
  dt = CFL*dx/umax
end function timestep
```

```
! ------------------------------------------------------------


! ************************************************************
function artificial_visc(rho, W, eta) result(Q)
! Artificial dissipative flux
!     Q = 1/2 * eta * sw * [ W(i+1) - 2W(i) + W(i-1) ]
!
! where sw is density switch:
!         |  |rho(i+1) - rho(i)| - |rho(i) - rho(i-1)|  |
!    sw = | ---------------------------------------- |
!         |  |rho(i+1) - rho(i)| + |rho(i) - rho(i-1)|  |
!

implicit none
real(o),intent(in),dimension(n)  :: rho
real(o),intent(in),dimension(n,3):: W
real(o),intent(in)               :: eta! Art_visc coefficient
real(o),dimension(n,3)           :: Q  ! Artificial viscidity
real(o),dimension(n-2)           :: dur, dul
real(o),dimension(n-2,3)         :: sw ! density switch


dur = dabs( rho(nc+1)-rho(nc  ) )
dul = dabs( rho(nc  )-rho(nc-1) )
sw(:,1) = dabs((dur - dul) / (dur+dul + 1e-5))
sw(:,2) = sw(:,1);  sw(:,3) = sw(:,1)
Q = 0
Q(nc, :) = 0.5*sw*eta*(W(nc+1,:)-2*W(nc,:)+W(nc-1, :))
end function artificial_visc

! ------------------------------------------------------------


! ************************************************************
subroutine u2WF(rho, u, p, E, W, F)
! Code the variables:
!                     rho, u, p, E    ==>    W, F
implicit none

real(o),intent(in)    :: p(n)         ! Pressure corresponding x
real(o),intent(in)    :: rho(n)       ! Density  corresponding x
real(o),intent(in)    :: u(n)         ! Velocity corresponding x
real(o),intent(in)    :: E(n)         ! energy   corresponding x
real(o),intent(out)   :: W(n,3)       ! conservative varibles
real(o),intent(out)   :: F(n,3)       ! flux

! Code the variables
  W(:,1) = rho  ;  W(:,2) = rho*u       ;  W(:,3) = rho*E
  F(:,1) = rho*u;  F(:,2) = rho*u**2 + p;  F(:,3) = u*(rho*E+p)

end subroutine u2WF
! ------------------------------------------------------------

! ************************************************************
subroutine W2u(W, rho, u, p, E)
! Decode the variables:
```

```
!                          W    ==>   rho, u, p, E
implicit none
real(o),intent(in)    :: W(n,3)        ! conservative varibles
real(o),intent(out)   :: p(n)          ! Pressure corresponding x
real(o),intent(out)   :: rho(n)        ! Density  corresponding x
real(o),intent(out)   :: u(n)          ! Velocity corresponding x
real(o),intent(out)   :: E(n)          ! energy   corresponding x

  rho = W(:,1)
  u   = W(:,2)/rho
  E   = W(:,3)/rho
  p   = (gamma-1)*rho*(E-0.5*u**2)
end subroutine W2u
! ------------------------------------------------------------

! ************************************************************
subroutine Lax(W, F, dt)
! Lax scheme for system of 1D conservation laws:
!
!  W(j,n+1) = + 1/2*  [W(j+1,n) + W(j-1,n)]
!             - 1/2*r*[F(j+1,n) + F(j-1,n)]
!  r = dt/dx
!
implicit none
real(o),intent(in)    :: F(n,3)        ! flux
real(o),intent(in)    :: dt            ! time step
real(o),intent(inout) :: W(n,3)        ! conservative varibles


  W(nc,:) = 0.5*( W(nc+1,:)+W(nc-1,:) )                    &
                           - dt/(2*dx) * ( F(nc+1,:)-F(nc-1,:) )

end subroutine Lax
! ------------------------------------------------------------

! ************************************************************
subroutine leapfrog(W, Wbefore, F, dt, nsteps)
! Lax scheme for system of 1D conservation laws:
!
!  W(j,n+1) = W(j, n-1) - r*[F(j+1,n) - F(j-1,n)]
!  r = dt/dx
!
implicit none
real(o),intent(in)    :: F(n,3)        ! flux
real(o),intent(in)    :: dt            ! time step
integer,intent(in)    :: nsteps        ! Number of time steps
real(o),intent(inout) :: Wbefore(n,3) ! conservative varibles
real(o),intent(inout) :: W(n,3)        ! conservative varibles
real(o)               :: temp(n,3)     ! temp of W_{t}

if (nsteps == 0) then
   Wbefore = W
   call Lax(W, F, dt)
else
  temp = W
  W(nc,:) = Wbefore(nc,:) - dt/dx * ( F(nc+1,:)-F(nc-1,:) )
  Wbefore = temp
```

```
end if

end subroutine leapfrog
! ------------------------------------------------------------

! ************************************************************
subroutine boundary(u)
! boundary conditions for the shock tube. should be implemented
! The boundary conditions are:
!        u(0,t)=u(L,t) = 0  => rho*u = 0 at both ends
! rho should be extrapolated at both ends
implicit none
real(o),intent(inout) :: u(n)          ! Velocity corresponding x

  u([1, n]) = 0;

end subroutine boundary
! ------------------------------------------------------------


! ************************************************************
subroutine output(t)
!  Write output data file
!
!  Output:  Data file "solution.dat" containing for each point
!           the following:
!           coordinate, density, velocity, pressure
implicit none
real(o), intent(in)   :: t             ! current time

open(unit = 8, file='solution.dat', status = 'replace')
write(8, '("%%%%%%%%", " 1D shock tube problem ", "%%%%%%%%%")')
write(8,*) '%'
write(8, '("% % % % % % % ", " conditions ", " % % % % % % %")')
write(8,'("% Right conditions             :")')
write(8,'("%   rho = ", f5.2, "  u = ", f5.2, "  p = ", f5.2)')&
                   rho1,            u1,             p1
write(8,'("% Left  conditions             :")')
write(8,'("%   rho = ", f5.2, "  u = ", f5.2, "  p = ", f5.2)')&
                   rho2,            u2,             p2
write(8,'("% Length of domain             :", f5.2)') L
write(8,*) '%'
write(8, '("% % % % % % % ", " parameter ", "% % % % % % % %")')
write(8,'("% CFL number for stability     :", f5.2)') CFL
write(8,'("% Number of grid points        :", i5  )') n
write(8,'("% time                         :", f5.2)') t
write(8,'("% Difference scheme            :", A   )') scheme
write(8,*) '%'
write(8, '("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")')
write(8,*)
write(8, '("%", 6x, "x", 8x, "rho", 8x, "u", 8x, "p")')
write(8, '(4f10.4)') (x(i), rho(i), u(i), p(i), i=1, n)
close(8)

write(*,*),  "Done! The flow data are writen to 'solution.dat' "
end subroutine output
! ------------------------------------------------------------
```

```
! ************************************************************
subroutine processBar(t, Tend, process)
!  draw the program running process bar
implicit none
real(o), intent(in)    :: t             ! current time
real(o), intent(in)    :: Tend          ! final time
integer, intent(inout):: process

if (nsteps ==1 ) then
   write(*,*), 'Running process:'
   write(*,*), '-------20%-------40%-------60%-------80%------100%'
   write(*,'(A,$)') ' '
elseif (int(t/(Tend/50)) == 50) then
   write(*,'(A,$)') '>|'
   write(*,*)
elseif (int(t/(Tend/50)) > process) then
   write(*,'(A,$)') '='
   process = int(t/(Tend/50))
end if
end subroutine processBar
! ------------------------------------------------------------

End program Tubeshock
```

B data2figure.m

```
% Get the data of the 1D shock tube solutions and show as a
% figure. the data: "solution.dat" = Data file containing for
% each grid point, coordinate, density, velocity, pressure,
% in the following format:
%
%              x(1)        rho(1)        u(1)        p(1)
%              x(2)        rho(2)        u(2)        p(2)
%               .             .            .          .
%               .             .            .          .
%              x(n)        rho(n)        u(n)        p(n)


M = load('solution.dat');
x   = M(:,1);
rho = M(:,2);
u   = M(:,3);
p   = M(:,4);

plot(x, rho, '-', x, u, '--', x, p, '-.', 'linewidth',1.5)
xlabel('$x$', 'interpreter','latex','fontsize', 15)
ylabel('$\rho$, $u$, $p$', 'interpreter','latex','fontsize', 15)
legend('density', 'volecity', 'pressure',0)
```