第三次大作业

一维Riemann问题数值模拟



姓名：李春蕾

学号：SY1804410

指导教师：宁方飞

# 有限体积法的控制方程

无源项的Euler方程为

采用有限体积法，在一个单元内将其离散化后得到

其中为单元某一面的面积，本问题一维情况下可以取为1。

其中I表示第I个单元，一维的情况只需要I即可，二维下标改为I, J

其中m表示该单元格的第m个表面，NF是表面的个数，一维情况下NF=1, 二维下NF=4。

各类对流项离散化格式的核心任务就是给出不同的。因为有限体积法的守恒性，离散越精确，右端项会越趋于0，所以右端代表的就是单元格内的残差。记为R。即

于是一维情况下公式可以改写为

而时间项离散的核心任务就是给出等号左端项的公式。

在该公式中，W是待求解的向量，称为流动变量，因为在有限体积法中它是守恒的，所以也称为守恒变量。Fc是单元面上的矢通量，代表了流出流出单元体的参数。

在一维情况下，它们分别代表

它们代表的是流场参数。

其中是密度, p是静压，u是速度， E是单位质量流体的总能，它被定义为

在一维情况下，u=v。e是单位质量内能，右端第二项则是单位质量动能。

其中H是单位质量流体总焓，它等于焓加动能。又因为焓定义为内能加上

所以

因为解向量有三个量，所以实际上只需要三个流场变量就可以唯一确定流场。我们选择

其中直接对应了W的前两个量，p和结合则可以对应总能。根据完全气体的相关公式，可以得到其转化关系如下：

显然W与F是可以一一对应的。也就是说它们可以相互转化。

# 对时间的离散：三阶显式荣格库塔法

不采用类似Lax-Wendroff，MacCormark这样时间空间共同离散的格式的话，时间项与空间项可以分开单独离散。

其中n代表第n时间步，W与R的上标表示荣格库塔法的第几步。每计算一步要进行一次空间离散，以更新残差R。

系数、、分别取0.1481， 0.4， 1。

CFL数取1.5。

# 时间间隔dt：当地时间步法

采用当地时间步法，每一个单元格有各自的(如荣格库塔法中所示).

但是由于只能取流场内最小的时间步作为向前推进的时间步，所以实际上在荣格库塔法中取统一的即可。

当地时间步的计算公式(在单元格I处)

其中与下面JST格式中的是同一个值，即当地Jacobian矩阵的特征值。

而流场的dt只能取最小值，即全局dt为

# 对对流项的离散:标量的JST格式

该问题下空间离散只有一个对流项。流通量的离散求解后的目的就是求解该单元格的残差R。对流项的离散分为5种：中心型格式，矢通量分裂格式，通量差分分裂格式，TVD格式，脉动分裂格式。

我们首先采用较简单的中心型格式。其中最常用的是JST格式。

我们采用标量的JST格式。

JST格式最主要的部分就是在对流项中减去一个人工粘性D。即

其中是单元格临近的两个单元格中心的流动变量的算术平均

是人工粘性，可以由周围单元格的W以及Jacobian矩阵的特征值计算出来：

其中是两个系数，而特征值同W一样，取临近两个单元格特征值的平均值

则与声速c有关。

系数是根据压力感知的开关函数给定的。

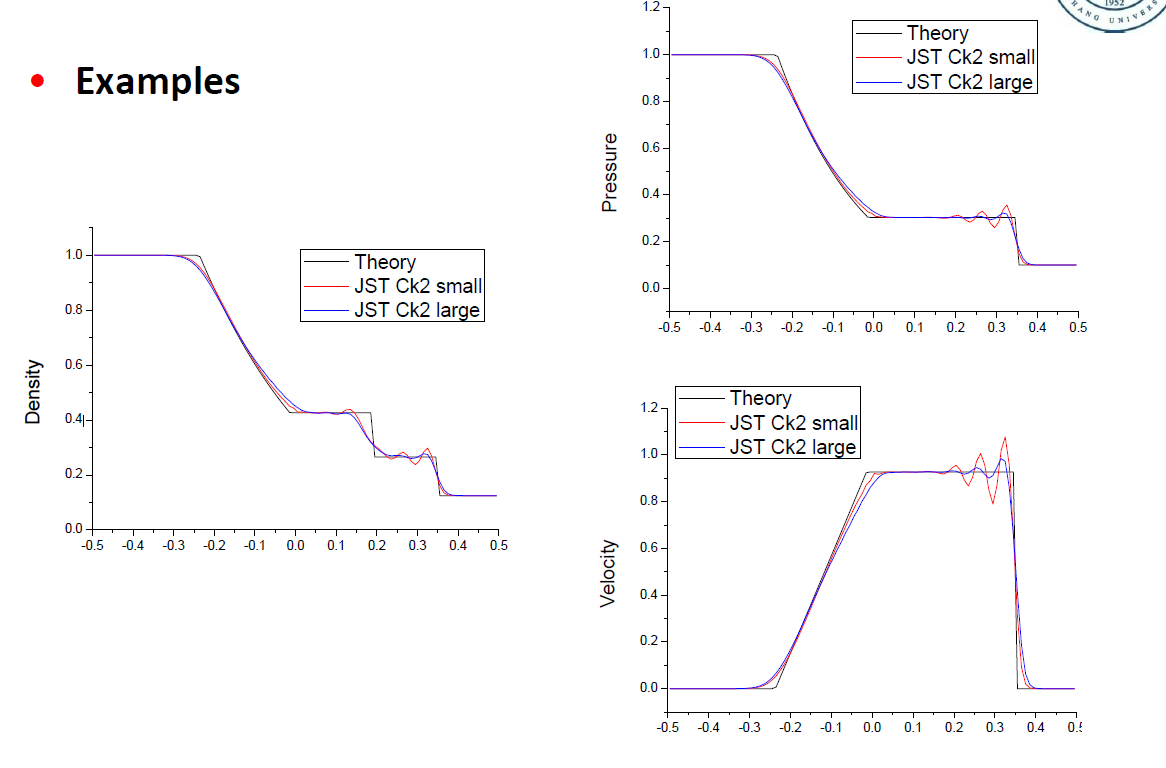
其中k2=0.5, 1/64<=k4<=1/128， 这里取k4=0.01

# 结果与精确解的比较与准确性分析

C:\Users\Administrator\Desktop\结果.eps结果如下图所示。

采用了1000个网格离散。时间到0.1s停止。可以看到，在速度的第二次间断处有轻微的震荡，出现色散。除此之外，在其他的间断处有圆角，证明出现了数值耗散。

精确解如下：



# 源程序：C++

该代码主要分为三个主要的函数：主函数控制两层循环，外层是时间循环，内层是空间循环。循环主体为TIME\_DIS函数，可以看到在头文件中将其宏定义为RungeKuttaTime函数。RungeKuttaTime函数内包含三步，每一步都先进行空间离散，只更新残差后更新流动变量W。空间离散为CONV函数，在头文件中宏定义为scalarJSTConv函数，利用标量的JST法计算当地的残差。

内部流场计算完以后，由于JST法利用了前一个点和后两个点作为模板，所以第一个点和最后两个点是边界点，在空间循环之后简单外推即可，从结果可以看到，波动尚未传到边界，因此边界的准确性不是那么重要。

下面是源代码

## 主函数main.C

#include "main.H"

#include <ctime>

#include<cstdlib>

double calDtGlobal(double W[][3]);

int main()

{

double W[maxSpace + 1][3] , R[maxSpace + 1][3] ;

init(W);

double dtGlobal;

int timeStep = 0;

for (double t = 0; t <= stopTime ; t += dtGlobal)

{

dtGlobal=calDtGlobal(W);

for (int I = 1; I <= maxSpace-2; I++)

{

TIME\_DIS(W, dtGlobal, R, I);

}

zeroGradBC(W);

timeStep++;

cout << "dtGlobal= " << dtGlobal;

cout << "\t time step = " << timeStep;

cout << "\t time = " << t << endl;

}

print(W);

return 0;

}

## 头文件 main.H

#ifndef main\_H

#define main\_H

#include<cmath>

#include<iostream>

using namespace std;

#define CONV scalarJSTConv

// #define CONV MacCormackConv

// #define CONV RoeConv

//#define TIME\_DIS EulerFTime

#define TIME\_DIS RungeKuttaTime

// declear the constants //

// \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* //

const int maxSpace= 1000;

const double physicalSpace = 2;

const double dx = physicalSpace / maxSpace;

const int maxTime = 100000;

const double stopTime = 0.1;

const double CFL=1.5;

const double GAMMA = 1.4;

// declear the inline funcs //

// \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* //

inline double p(const double W[])

{

double u = W[1]/W[0];

return (GAMMA-1) \* W[0]\* ( W[2]/W[0] - 0.5\* u \* u );

}

inline double max(const double a, const double b)

{

return a>b?a:b;

}

//计算声速c

inline double c(const double W[3])

{

return sqrt( GAMMA \* p(W)/W[0] );

}

inline double lambda(const double W[3])

{

return fabs(W[1]/W[0]) + c(W);

}

inline double WToRho(const double vec[3])

{

return vec[0];

}

inline double WToVelocity(const double vec[3])

{

return vec[1]/vec[0];

}

// define the func //

// \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* //

void CONV(const double W[][3], const double dtLocal, double R[][3], const int I);

void TIME\_DIS(double W[][3], double dtLocal, double R[][3], const int I);

void zeroGradBC(double W[][3]);

void init(double W[][3]);

double localTime(const double W[][3],const int I);

void WToF(double const W[3], double F[3]);

void print( const double W[][3]);

void printW(const double W[][3]);

double minOverScalarField(double scalarField[]);

#endif

## 时间离散荣格库塔法 RungeKuttaTime.C

#include"main.H"

#include<cstdlib>

//一阶精度的三阶显式RungeKutta法

void RungeKuttaTime(double W[][3], double dt, double R[][3], const int I)

{

const double alpha1=0.1481, alpha2=0.4, alpha3=1;

//先定义W0,用于保存原始的W

double W0[3];

for (int k = 0; k < 3; k ++)

{

W0[k] = W[I][k];

}

//后面每一步都先计算该单元格的残差, 后根据RK公式更新W

CONV(W, dt, R, I);

for (int k = 0; k < 3; k ++)

{

W[I][k] = W0[k] - alpha1 \* dt / dx \* R[I][k];

}

CONV(W, dt, R, I);

for (int k = 0; k < 3; k ++)

{

W[I][k] = W0[k] - alpha2 \* dt / dx \* R[I][k];

}

CONV(W, dt, R, I);

for (int k = 0; k < 3; k ++)

{

W[I][k] = W0[k] - alpha3 \* dt / dx \* R[I][k];

}

}

## 空间离散标量JST格式 scalarJSTConv.C

#include"main.H"

void scalarJSTConv(const double W[][3], const double dt, double R[][3], const int I)

{

const double k2=0.5, k4=0.01;

double D[3], Wf[3], F[3];

if(I>=1 && I<=maxSpace-2)

{

//先定义一些系数, Lambda和Ep2, Ep4, 用于计算D[k]

//其中Y与Y1是开关函数,用于切换Ep2和Ep4

const double Lambda = 0.5 \* (lambda(W[I]) + lambda(W[I + 1]));

const double p1=p(W[I+1]), p2=p(W[I + 2]), p0=p(W[I]), p\_1=p(W[I - 1]);

const double Y = fabs(p1- 2 \* p0 + p\_1)

/ (p1+ 2 \* p0 + p\_1);

const double Y1 = fabs(p2 - 2 \* p1+ p0)

/ (p2 + 2 \* p1+ p0);

const double Ep2 = k2 \* max(Y, Y1);

const double Ep4 = max(0, k4 - Ep2);

//然后计算人工黏性D, 计算face上所用的流动变量Wf(中心型简单算数平均)

for (int k = 0; k < 3; k++)

{

D[k] = Lambda

\* ( Ep2 \* (W[I + 1][k] - W[I][k])

- Ep4 \* (W[I + 2][k] - 3 \* W[I + 1][k] + 3 \* W[I][k] - W[I - 1][k])

);

Wf[k] = 0.5 \* (W[I][k] + W[I + 1][k]);

}

}

//face上的流动变量Wf转换为通量F

// WToF(Wf, F);

double p0=p(Wf);

double u=Wf[1]/Wf[0];

F[0] = Wf[1];

F[1] = Wf[0] \* u \* u + p0;

F[2] = (Wf[2] + p0) \* u;

//最后算出残差R

for (int k = 0; k < 3; k++)

{

R[I][k] = F[k] - D[k];

}

}

## 当地时间步 localTime.C

#include"main.H"

double localTime(const double W[][3], const int I)

{

return CFL\*dx/lambda(W[I]);

}

## 用于指定编译顺序的Makfile文件

CONV = scalarJSTConv

#CONV = MacCormackConv

#CONV = RoeConv

TIME\_DIS = RungeKuttaTime

#TIME\_DIS = EulerFTime

OBJS = init.o print.o ${CONV}.o main.o WToF.o localTime.o BC.o ${TIME\_DIS}.o minOverScalarField.o calDtGlobal.o

SRCS = init.C print.C ${CONV}.C main.C WToF.C localTime.C BC.C ${TIME\_DIS}.C minOverScalarField.C calDtGlobal.C

LIBS = -lm

EXE = out

CFLG = -g -Wall -std=c++11

start : $(OBJS) main.H

g++ -o ${EXE} ${CFLG} ${OBJS} ${LIBS}

localTime.o : localTime.C main.H

g++ ${CFLG} -c localTime.C

print.o : print.C main.H

g++ ${CFLG} -c print.C

init.o : init.C main.H

g++ ${CFLG} -c init.C

WToF.o : WToF.C main.H

g++ ${CFLG} -c WToF.C

calDtGlobal.o : calDtGlobal.C main.H

g++ ${CFLG} -c calDtGlobal.C

${CONV}.o : ${CONV}.C WToF.C main.H

g++ ${CFLG} -c ${CONV}.C

BC.o : BC.C main.H

g++ ${CFLG} -c BC.C

minOverScalarField.o : minOverScalarField.C main.H

g++ ${CFLG} -c minOverScalarField.C

${TIME\_DIS}.o : ${TIME\_DIS}.C main.H

g++ ${CFLG} -c ${TIME\_DIS}.C

main.o : $(SRCS) main.H

g++ ${CFLG} -c main.C

.PHONY:clean\_o clean\_data clean\_log clean\_all

clean\_o :

rm -rf \*.o

clean\_data :

rm -rf data/\*

clean\_log :

rm -rf log/\*

clean\_all :

rm -rf data/\* log/\* \*.o ${EXE}

## 其他辅助性函数

### 给定初场的init.C

#include"main.H"

void init(double W[][3])

{

cout<<"\nInitializing the fields..."<<endl;

double u1=0.0, rho1=1.0, p1=1.0e6;

double u2=0.0, rho2=0.125, p2=0.1e6;

for(int i=0; i< maxSpace/2; i++)

{

W[i][0]=rho1;

W[i][1]=u1\*rho1;

W[i][2]=p1/(GAMMA-1) + 0.5\*rho1\*u1\*u1;

}

for(int i=maxSpace/2; i<= maxSpace; i++)

{

W[i][0]=rho2;

W[i][1]=u2\*rho2;

W[i][2]=p2/(GAMMA-1) + 0.5\*rho2\*u2\*u2;

}

}

### 外推边界BC.C

#include "main.H"

// 0方向梯度边界条件, 边界上的点直接等于内部的点 //

// \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* //

void zeroGradBC(double W[][3])

{

for (int k = 0; k < 3; k++)

{

W[0][k] = W[1][k];

W[maxSpace - 1][k] = W[maxSpace - 2][k];

W[maxSpace][k] = W[maxSpace - 1][k];

}

}

### W与F相互转换的WToF.C

#include"main.H"

void WToF(double const W[3], double F[3])

{

double p0=p(W);

double u=W[1]/W[0];

F[0] = W[1];

F[1] = W[0] \* u \* u + p0;

F[2] = (W[2] + p0) \* u;

}

### 打印流场的print.C

#include"main.H"

#include<fstream>

#include<iomanip>

void print(const double W[][3])

{

ofstream foutAll("data/result.dat");

foutAll.setf(ios::left);

foutAll.precision(3);

cout<<"\nprint the rho, u, p, in the data/result.dat, first column is distance"<<endl;

foutAll<<"x"<<'\t'<<"rho"<<'\t'<<"u"<<'\t'<<"p"<<endl;

for (int i = 0; i <= maxSpace; i++)

{

foutAll<<i\*dx<<'\t'<<W[i][0]<<'\t'<<W[i][1]/W[i][0]<<'\t'<<p(W[i])<<endl;

}

}

void printW(const double W[][3])

{

ofstream fout("data/vecterW.dat");

cout<<"\nprint W[][3]\n";

for (unsigned i =0 ; i <=maxSpace ; i++)

{

fout<<W[i][0]<<"\t"<<W[i][1]<<'\t'<<W[i][2]<<endl;

}

}

### 找流场最小值的minOverScalarField.C

#include"main.H"

double minOverScalarField(double scalarField[])

{

double minVal;

for (int i = 0; i <=maxSpace ; i++)

{

if (minVal<scalarField[i])

{

minVal=scalarField[i];

}

}

return minVal;

}

### 找流场全局dt的calDtGlobal.C

#include"main.H"

double calDtGlobal(double W[][3])

{

double min=1e10;

double dtLocal;

for (int I = 0; I <=maxSpace ; I++)

{

dtLocal=localTime(W,I);

if(dtLocal<min)

{

min=dtLocal;

}

}

return min;

}