One Dimensional Euler's Equations

Computational Physics

Adam Reyes

November 18, 2013

1 Introduction

WeThe one-dimensional partial differential equation of hydrodynamics we wish to solve is

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = 0 \tag{1.1}$$

Here U and F are given by the vectors

$$\mathrm{U} = \begin{pmatrix} \rho \\ \rho v \\ E \end{pmatrix} \quad \mathrm{F} = \begin{pmatrix} \rho v \\ \rho v^2 + P \\ (E+P)v \end{pmatrix} \tag{1.2}$$

where for U ρ is the density of the gas, ρv is the momentum density and E is the energy density, with $E = \frac{1}{2}\rho v^2 + \epsilon$, with ϵ being the thermal energy density. For F, ρv is the flux of mass density, and P is the pressure.

The pressure is to be given by an equation of state (EOS), and in general $P = P(\rho, \epsilon)$. For and ideal gas the EOS is

$$P = \rho\epsilon(\gamma - 1) \tag{1.3}$$

where γ is the adiabatic index, which we take to be 1.4.

To solve the differential equation (**??**) we first integrate in x using the Mid-point method and get

$$\frac{dU_i}{dt} = -\frac{F_{i+1/2} - F_{i-1/2}}{\Delta x} \tag{1.4}$$

We can then integrate forward in time using the forward Euler method to obtain

$$U^{n+1} = U^n - \frac{\Delta t}{\Delta x} \cdot \frac{F_{i+1/2} - F_{i-1/2}}{\Delta x}. \tag{1.5}$$

Here the fluxes are given at the half-space steps. In order to obtain these fluxes we employ an HLL Riemann Solver. The HLL flux is given as

$$F^{HLL} = \frac{\alpha^+ F^L + \alpha^- F^R - \alpha^+\alpha^-(U^R - U^L)}{\alpha^+ + \alpha^-} \tag{1.6}$$

Here the superscripts L and R refer to the spatial points i to the left and right of the half step of interest. the $\alpha^\pm$ terms refer to the max of the plus and minus wavespeeds and 0.

There is a stability condition constraining the size of the time steps that can be taken. This is given by the Courant-Friedrich-Levy condition:

$$\Delta t < \frac{\Delta x}{MAX(\alpha^\pm)} \tag{1.7}$$

2 Sod Shock Tube

The first system we will solve is that of the sod shock tube. The initial conditions are as follows:

1. there is some interface at $x = 0$ initially dividing two regions.
2. at $t = 0$ the gas is stationary
3. to the left of the boundary the density and pressure are 1. To the right the density is 0.125 and the pressure 0.1.

The code was run for $t = 0.25$.

2.1 Results

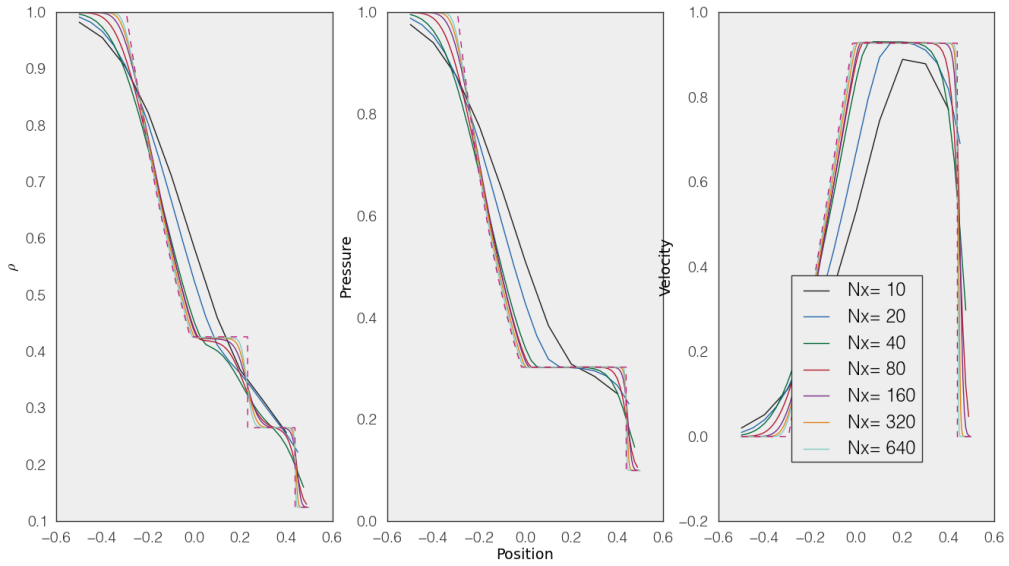

Figure 2.1: Plots of density (ρ), pressure and velocity at various resolutions, with the analytic solution given as the dashed lines.

We can see in Figure 2.1 that the method seems to capture the characteristics of the analytic solution, and there are the appropriate shocks in all three primitive variables. Using the data from the analytic solution we can then make a plot of the accumulated error of the method at these various resolutions, as seen in 2.2.

In doing so we get sort of a strange result. The method begins converging with a log-log slope less than one, but then begins to accumulate more error as resolution is increased, until it starts to converge again with a slope of exactly one. Since the method used in 1.5 was only first order, we expect that the slope be less than one, but there certainly does not seem to be a reason for the middle region.

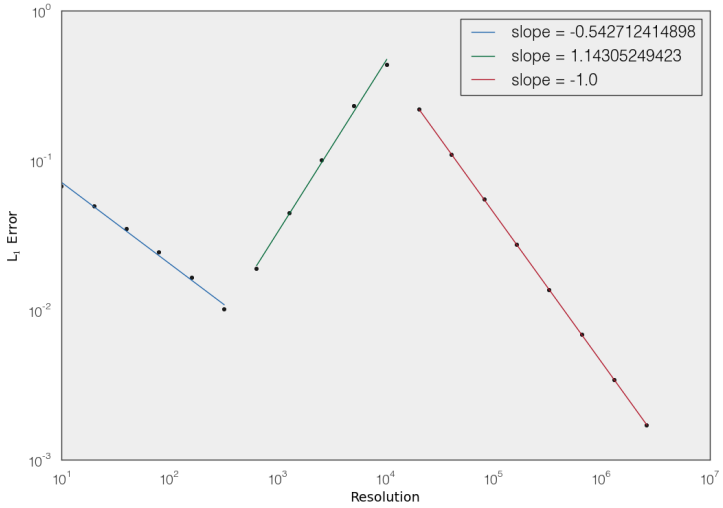

Figure 2.2: plot of the L_1 error against the resolution on a log-log scale with linear fits in three regions

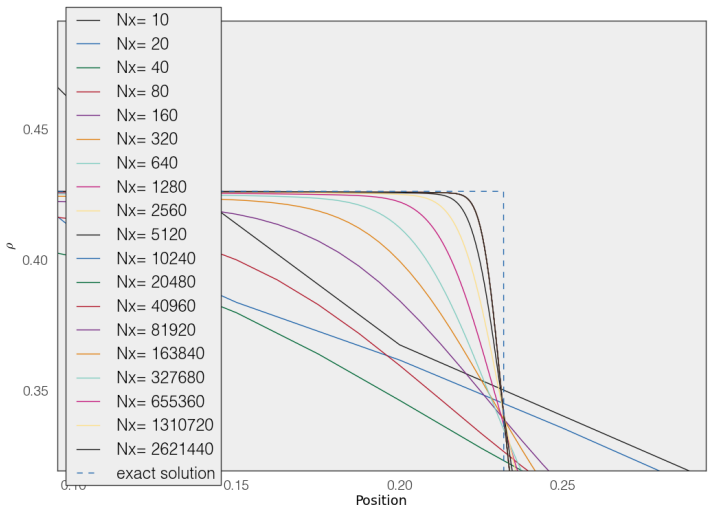

Figure 2.3: density plot at various resolution enlarged around a shock, with the analytic solution shown in the dashed line

In Figure 2.3 we can see the behavior of the computed solutions around a shock. We can notice that even at a resolution on the order of 10^6 the computed solutions do not seem to fully capture the characteristics we are after. This leads me to conclude that there is some error in the implementation of the method, but have been unable to identify it. Still it is strange that the code eventually seems to seems to converge with a slope of one at very high resolution. Because of hardware limitations it is not practical to run the code at even higher resolutions but it would be interesting to see if the trend continues.

3 Isentropic Wave

We now consider an Isentropic wave, where the initial conditions are specified by the following equations:

$$\rho(x) = \rho_{ref}(1 + \alpha f(x)) \tag{3.1}$$
$$P(x) = K\rho^{\gamma} \tag{3.2}$$
$$v(x) = \frac{2}{\gamma - 1}\sqrt{\gamma K}(\rho(x)^{\frac{\gamma-1}{2}} - \rho_{ref}^{\frac{\gamma-1}{2}}) \tag{3.3}$$

Again γ is the adiabatic index. K, α and ρ_{ref} are arbitrary constants. $f(x)$ is any smooth function, for this assignment I use a gaussian of the form

$$f(x) = e^{-ax^2} \tag{3.4}$$

Since the system begins with the velocity being different across the space, regions will move faster than others until eventually there is a shock.

3.1 Results

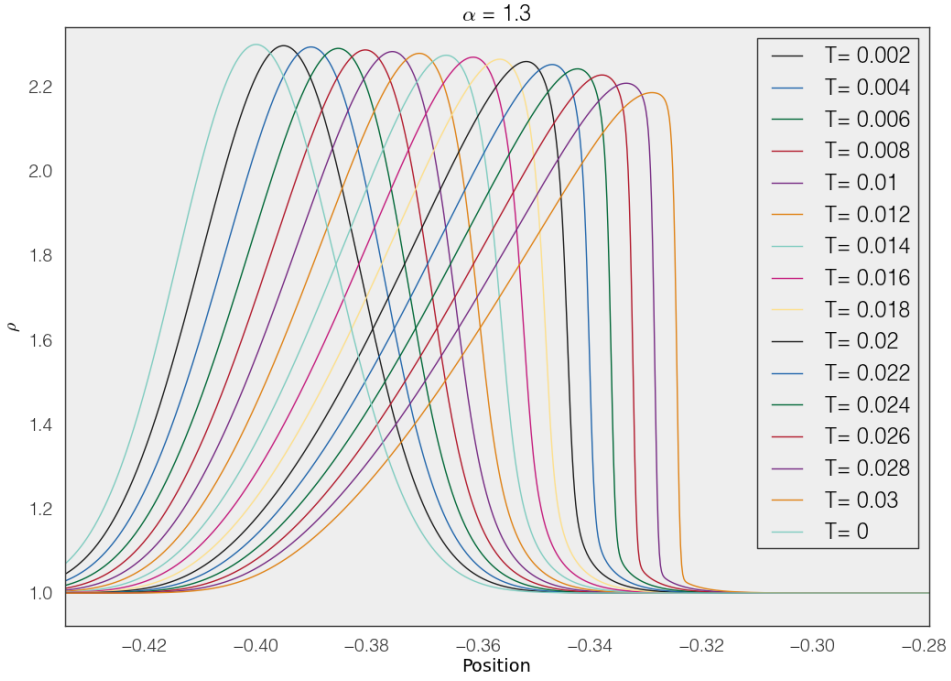

Figure 3.1: mass density plotted at various time steps with $\alpha = 1.3$ with 2000 spatial zones.

We can see in Figure 3.1 the computed solutions taken with 2000 zones at various times. We can see that the initial wave is of a gaussian profile and some where near t = 0.02 the wave begins to shock, and as time goes on it diffuses and spreads behind the shock.

Before the time of the shock we can expect that equations 3.2 and 3.3 to still hold true. However, there is already some diffusion inherent in the method, which is minimized by taking higher resolution, so we would expect that before the shock the difference in the computed solutions from 3.2 and 3.3 to go like the diffusion from the method, and then to change drastically at the time of the shock. We see exactly this in Figure 3.2.

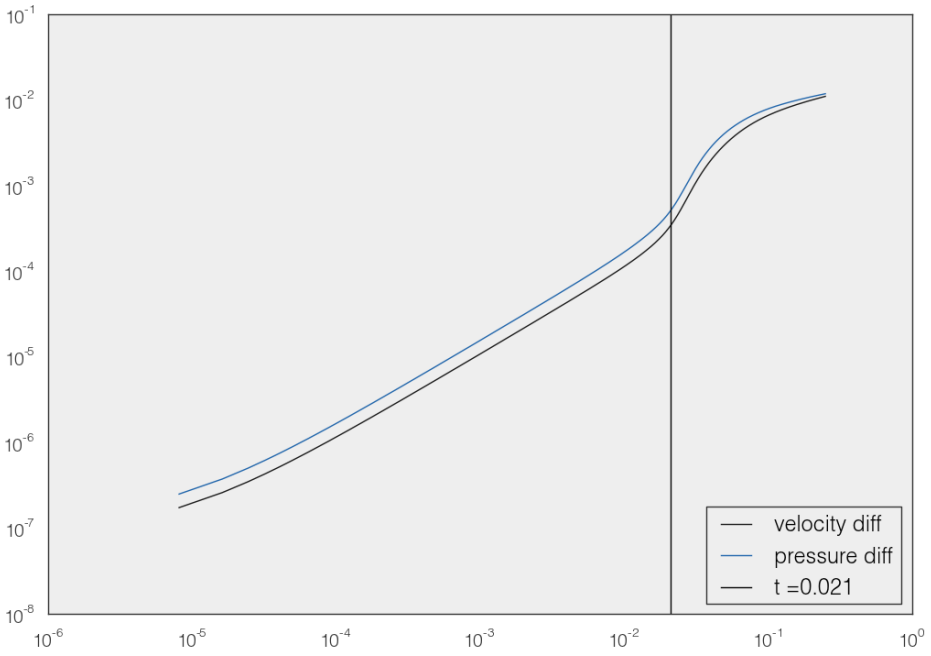

Figure 3.2: differences in pressure and velocity between the computed values and equations 3.2 and 3.3. Vertical line shows rough estimate of the shock at t = 0.021

4 Source Code

Listing 1: Sod Shock Tube

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

double gam = 1.4;

void initnxN(double *** A, int N, int n){
  *A = (double**)malloc(n*sizeof(double*));
  int i, j;
  for (i=0;i<3;++i){
    (*A)[i] = (double*)malloc(N*sizeof(double));
    for (j=0; j<3;++j){
      (*A)[i][j] = 0.;
    }
  }
}
void initialize_system(double ** U, double * rho, double * P, double * V, int N){
  double v = 0.; double dx = 1./(double)N; double x = -0.5-dx; int i = 0;
  int NT = N + 2;
  while(x<0){
    rho[i] = 1.;
    P[i] = 1.;
    x += dx; ++i;
  }
  while(x<.5+dx){
    //printf("%f\n", x);
    rho[i] = 0.125;
    P[i] = 0.1;
    x+= dx; ++i;
  }
  int count = 0;
  for(i=0;i<NT;++i){
    V[i] = 0;
    U[0][i] = rho[i];
    U[1][i] = rho[i]*v;
    U[2][i] = pow(U[1][i],2)/(2*U[0][i]) + P[i]/(gam - 1);
    //printf("i %d start %f  %f   %f\n", i,U[0][i],U[1][i],U[2][i]);
  }
}
void Reset_BC(double ** U, int NT){
  U[0][0] = 1.;
  U[1][0] = 0.;
  U[2][0] = 2.5;

  U[0][NT-1] = .125;
  U[1][NT-1] = 0.;
  U[2][NT-1] = 0.25;
}
void getcol(double **U, int j, double * col){
  int i;for(i=0;i<3;++i){
    col[i] = U[i][j];
  }
  //printf("j %d column %f  %f   %f\n", j,U[0][j],U[1][j],U[2][j]);
}
double Pressure(double* Ui){
  double Et = Ui[2] - pow(Ui[1],2)/(2*Ui[0]);
  double P = Et*(gam-1);
  //printf("%f  %f  %f  %f\n", Ui[0],Ui[1],Ui[2], P);
```

```
  return(P);
}
void make_P(double ** U, int NT, double * Press){
  Press[0] = 1.;
  int i;for(i=1;i<NT-1;++i){
    double Ui[3]; getcol(U, i, Ui);
    Press[i] = Pressure(Ui);
    //printf("Ui_%d %f  %f P=%f\n", i, Ui[0], Ui[1], Press[i]);
  }
  Press[NT] = 0.1;
  //printf("%f\n", Press[NT]);
}
double Velocity(double * Ui){
  double v = Ui[1]/Ui[0];
  //printf("%f   %f   %f\n", Ui[1], Ui[0], v);
  return(v);
}
double C_s(double P, double rho){
  double c = sqrt(gam*P/rho);
  //printf("%f  %f  %f\n", P, rho, c);
  return(c);
}
double lambdapm(double v, double c_s, int pm){
  double lam = v + (double)pm *c_s;
  //printf("%f  %f\n",v, c_s);
  return(lam);
}
double Max(double* a, int N){
  double max = a[0];
  int i; for(i=0;i<N;++i){
    if(max<a[i]) max = a[i];
  }
  return(max);
}
double alphapm(double* Ui, double* Uimo, int pm, double  Pi, double Pimo){
  double vi = Velocity(Ui); double ci = C_s(Pi, Ui[0]);
  double lami = lambdapm(vi, ci, pm);
  double vimo = Velocity(Uimo); double cimo = C_s(Pimo, Uimo[0]);
  double lamimo = lambdapm(vimo, cimo, pm);
  double tobemaxed[3] = {0, (double)pm*lami, (double)pm*lamimo};
  double alpha = Max(tobemaxed, 3);
  //printf("%f  %f\n", lami, lamimo);
  return(alpha);
}
void Flux(double * Ui, double * Fi, double P){
  Fi[0] = Ui[1];
  Fi[1] = pow(Ui[1], 2)/Ui[0] + Pressure(Ui);
  Fi[2] = (Ui[2] + Pressure(Ui))*Velocity(Ui);
}
void Fi_mh(double * Ui, double * Uimo, double * Fimh, double * alphas, double Pi,
     double Pimo){
  double Fi[3], Fimo[3];//double  Fhll[3];
  Flux(Ui, Fi, Pi); Flux(Uimo, Fimo, Pimo);
  double alphap = alphapm(Ui, Uimo, 1, Pi, Pimo);
  double alpham = alphapm(Ui, Uimo, -1, Pi, Pimo);
  //printf("alpha=%f  %f  Pi = %f\n", alphap, alpham, Pi);
  alphas[0] = alphap; alphas[1] = alpham;
  int i; for(i=0;i<3;++i){
    Fimh[i] = (alphap*Fimo[i] + alpham*Fi[i] - alphap*alpham*(Ui[i]-Uimo[i]))/(
        alphap+alpham);
  }
  //printf("Fimh = %f\n", Fimh[0], Fimh[1], Fimh[2]);
}
```

```
void make_F(double ** U, double ** F, double **alpha, int NT, double * P){
  double alphas[2];
  F[0][0] = 0.; F[1][0] = 1.; F[2][0] = 0.;
  int i,j; for(i=1;i<NT;++i){
    double Ui[3], Uimo[3]; getcol(U, i, Ui); getcol(U, i-1, Uimo);
    double Fimh[3]; Fi_mh(Ui, Uimo, Fimh, alphas, P[i], P[i-1]);
    for(j=0;j<2;++j){
      if(i>0 && i<NT){
  alpha[j][i-1]=alphas[j];
      }
      //printf("alphas %f   %f\n", alpha[j][i], alphas[j]);
    }
    for(j=0;j<3;++j){
      if(i>0 && i<NT){
      F[j][i] = Fimh[j];
      }
    }
    // printf("Fi_%d = %f  %f  %f\n",i, F[0][i],F[1][i],F[2][i]);
  }
}
double get_step(double maxalpha, double dx){
  double dt = dx/(10*maxalpha);
  //dt = 0.00001;
  return(dt);
}
double get_maxalpha(double ** alpha, int NT){
  double max = fabs(alpha[0][0]); //printf("here is the first one%f\n", max);
  int i,j; for(i=0;i<NT-2;++i){
    for(j=0;j<2;++j){
      if(max<fabs(alpha[j][i])) max = fabs(alpha[j][i]);
      //printf("other alphas %f\n", alpha[j][i]);
    }
  }
}
double advance_system(double ** U, double dx , double t, int NT, double * P){
  double ** alpha; initnxN(&alpha, NT-2, 2);
  make_P(U, NT, P);
  double ** F; initnxN(&F, NT, 3); make_F(U, F, alpha, NT, P);
  double maxalpha = get_maxalpha(alpha, NT);
  double dt = get_step(maxalpha, dx);
  double Un[3];
  //printf(" column %f  %f   %f  F[1][1] = %f\n", U[0][0],U[1][0],U[2][0], F
      [1][1]);
  double L;
  int i,j; for(i=0;i<NT;++i){
    for(j=0;j<3;++j){
      L = - dt*(F[j][i+1]-F[j][i])/dx;
      Un[j] = U[j][i] + L;
      //printf("j=%d  L = %f\n", j, L);
      U[j][i] = Un[j];
    }
    //printf("i %d column %f  %f   %f\n", i,U[0][i],U[1][i],U[2][i]);
  }
  //printf("%f\n", dt);
  Reset_BC(U, NT);
  for(i=0;i<NT;++i){
    //printf("i %d column %f  %f   %f\n", i,U[0][i],U[1][i],U[2][i]);
  }
  t += dt;
  return(t);
}
void write(double ** U, FILE * fid, int NT, double dx){
  double x = -0.5;
```

```
  int i;for(i=1;i<NT-1;++i){
    fprintf(fid, "%e    %e   %e   %e\n", x, U[0][i], U[1][i], U[2][i]);
    x+= dx;
  }
}
void getRow(double ** U, int NT, double * thing, int row){
  int count = 1;
  int i;for(i=1;i<NT-1;++i){
    thing[i-1] = U[row][i];
    count+=1;
  }
}

int main(int argc, char ** argv){
  // ./a.out Nx
  FILE * fid1, * fid2;
  fid1 = fopen("initial.dat", "w");
  double T = 0.25; double t = 0; double tn;
  double L = 1.;   int Nx = atoi(argv[1]); double dx = L/(double)Nx;int NT = Nx +
      2;
  double **U; double rho[Nx+2]; double P[Nx+2];double V[Nx+2];
  initnxN(&U, Nx+2, 3);
  initialize_system(U, rho, P, V, Nx);
  write(U, fid1, NT, dx);
  while(t<T){
    tn = advance_system(U, dx, t, NT, P); t = tn; //printf("t=%f\n", t);
    //printf("i 21 column %f  %f  %f", U[0][21], U[1][21], U[2][21]);
    //break;
    //printf("E = %f\n", U[2][1]);
     }
  getRow(U, NT, rho, 0);
  fid2 = fopen("final.dat", "w");
  write(U, fid2, NT, dx);
}
```

Listing 2: Sod Shock Tube - Plotting and Error

```
import scipy as sp
import numpy as np
import matplotlib.pyplot as plt
import pdb
import os
#pauls columns
#x rho v p ?<-dontworry about this
def L(x, a, c, N):
    con = (len(a)/len(c))
    err = 0
    dx = 1./float(N)
    for i in range(len(c)-1):
        diff = ((c[i] - a[int((i)*con)]))
        err += np.abs(diff*dx)

    return(err)
def makeprim(rho, mom, E):
    gam = 1.4
    eps = E - mom**2 / (2*rho)
    P = eps*(gam - 1)
    V = mom/rho
    data = [P, V]
    return(data)
Ns, L1 = [], []
filename = 'final.dat'
anal = 'solution.dat'
```

```
data = np.loadtxt(anal)
xa = data[:,0]
va = data[:,2]
rhoa = data[:,1]
Pa = data[:,3]
Nx = 5
count = 1
plt.figure()
figrho = plt.subplot(131)
plt.ylabel(r'$\rho$')
figP = plt.subplot(132)
plt.ylabel(r'Pressure')
plt.xlabel('Position')
figV = plt.subplot(133)
plt.ylabel(r'Velocity')
while(count < 8):
    Nx *= 2
    cmnd = './prob1 ' + str(Nx)
    os.system(cmnd)
    data = np.loadtxt(filename)
    lab = 'Nx= ' + str(Nx)
    x = data[:,0]
    rho = data[:,1]
    mom = data[:,2]
    E = data[:,3]
    [P, V] = makeprim(rho, mom, E)
    #print(L(x, Pa, P, Nx))
    L1.append(L(x, rhoa, rho, Nx))
    Ns.append(float(Nx))
    figrho.plot(x,rho, label = lab)
    figP.plot(x, P)
    figV.plot(x, V, label = lab)
    count +=1

figrho.plot(xa,rhoa,'--' ,label = 'exact solution')
figP.plot(xa, Pa, '--')
figV.plot(xa, va, '--')


figV.legend(loc = 'center', bbox_to_anchor = (0.5, 0.3))
'''
fit1, Ns1 = [], []
fit2, Ns2 = [], []
fit3, Ns3 = [], []
N1 = Ns[0:6]
L11 = L1[0:6]
N2 = Ns[6:11]
N3 = Ns[11:19]
L12 = L1[6:11]
L13 = L1[11:19]


m1, b1 = np.polyfit(np.log10(N1), np.log10(L11), 1)
m2, b2 = np.polyfit(np.log10(N2), np.log10(L12), 1)
m3, b3 = np.polyfit(np.log10(N3), np.log10(L13), 1)
print(m1,b1)
print( m2, b2)
print(m3, b3)
for n in N1:
    fit1.append((10**b1 *n**m1))
    Ns1.append(n)
for n in N2:
    fit2.append((10**b2 *n**m2))
```

```
    Ns2.append(n)
for n in N3:
    fit3.append((10**b3 *n**m3))
    Ns3.append(n)

lab1 = 'slope = '+str(m1)
lab2 = 'slope = '+str(m2)
lab3 = 'slope = '+str(m3)
plt.figure()
plt.plot(Ns, L1, 'o')
plt.plot(Ns1, fit1, label = lab1)
plt.plot(Ns2, fit2, label = lab2)
plt.plot(Ns3, fit3, label = lab3)
plt.yscale('log')
plt.xscale('log')
plt.ylabel(r'L$_1$ Error')
plt.xlabel('Resolution')
plt.legend()
'''
plt.show()
```

Listing 3: Isentropic Wave

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

double gam = 1.4;
double alph = 1.3;

void initnxN(double *** A, int N, int n){
  *A = (double**)malloc(n*sizeof(double*));
  int i, j;
  for (i=0;i<3;++i){
    (*A)[i] = (double*)malloc(N*sizeof(double));
    for (j=0; j<3;++j){
      (*A)[i][j] = 0.;
    }
  }
}
double f(double x){
  return(exp(-pow((x+0.4)*50,2)));
}
double density(double x){
  return(1+alph*f(x));
}
double Pressurei(double rho){
  double P = pow(rho, gam);
  return(P);
}
double Velocityi(double  rho){
  double gama = 2./(gam-1.);
  double v = gama*sqrt(gam) * (pow(rho,1/gama) - 1);
  return(v);
}
void initialize_system(double ** U, double * rho, double * P, double * V, int NT)
    {
  double v = 0.; double dx = 1./(double)NT; double x = -0.5-dx; int i = 0;
  while(x<.5+dx){
    rho[i] = density(x);
    P[i] = Pressurei(rho[i]);
    V[i] = Velocityi(rho[i]);
```

```
    x += dx; ++i;
  }
  for(i=0;i<NT;++i){
    U[0][i] = rho[i];
    U[1][i] = rho[i]*V[i];
    U[2][i] = pow(U[1][i],2)/(2*U[0][i]) + P[i]/(gam - 1);
  }
  //printf("last guy=%f\n", U[0][NT-2]);
}
void Reset_BC(double ** U, int NT){
  double Ul[3], Ur[3];
  int i;for(i=0;i<3;++i){
    Ul[i] = U[i][1];
    Ur[i] = U[i][NT-2];
    U[i][0] = Ur[i];
    U[i][NT-1] = Ul[i];
  }
}
void getcol(double **U, int j, double * col){
  int i;for(i=0;i<3;++i){
    col[i] = U[i][j];
  }
  //printf("j %d column %f  %f   %f\n", j,U[0][j],U[1][j],U[2][j]);
}
double Pressure(double* Ui){
  double Et = Ui[2] - pow(Ui[1],2)/(2*Ui[0]);
  double P = Et*(gam-1);
  //printf("%f  %f  %f  %f\n", Ui[0],Ui[1],Ui[2], P);
  return(P);
}
void make_P(double ** U, int NT, double * Press){
  Press[0] = 1.;
  int i;for(i=1;i<NT-1;++i){
    double Ui[3]; getcol(U, i, Ui);
    Press[i] = Pressure(Ui);
    //printf("Ui_%d %f  %f P=%f\n", i, Ui[0], Ui[1], Press[i]);
  }
  Press[NT] = 0.1;
  //printf("%f\n", Press[NT]);
}
double Velocity(double * Ui){
  double v = Ui[1]/Ui[0];
  //printf("%f   %f   %f\n", Ui[1], Ui[0], v);
  return(v);
}
double C_s(double P, double rho){
  double c = sqrt(gam*P/rho);
  //printf("%f  %f  %f\n", P, rho, c);
  return(c);
}
double lambdapm(double v, double c_s, int pm){
  double lam = v + (double)pm *c_s;
  //printf("%f  %f\n",v, c_s);
  return(lam);
}
double Max(double* a, int N){
  double max = a[0];
  int i; for(i=0;i<N;++i){
    if(max<a[i]) max = a[i];
  }
  return(max);
}
double alphapm(double* Ui, double* Uimo, int pm, double  Pi, double Pimo){
```

```
  double vi = Velocity(Ui); double ci = C_s(Pi, Ui[0]);
  double lami = lambdapm(vi, ci, pm);
  double vimo = Velocity(Uimo); double cimo = C_s(Pimo, Uimo[0]);
  double lamimo = lambdapm(vimo, cimo, pm);
  double tobemaxed[3] = {0, (double)pm*lami, (double)pm*lamimo};
  double alpha = Max(tobemaxed, 3);
  //printf("%f  %f\n", lami, lamimo);
  return(alpha);
}
void Flux(double * Ui, double * Fi, double P){
  Fi[0] = Ui[1];
  Fi[1] = pow(Ui[1], 2)/Ui[0] + P;
  Fi[2] = (Ui[2] + P)*Velocity(Ui);
}
void Fi_mh(double * Ui, double * Uimo, double * Fimh, double * alphas, double Pi,
     double Pimo){
  double Fi[3], Fimo[3];//double  Fhll[3];
  Flux(Ui, Fi, Pi); Flux(Uimo, Fimo, Pimo);
  double alphap = alphapm(Ui, Uimo, 1, Pi, Pimo);
  double alpham = alphapm(Ui, Uimo, -1, Pi, Pimo);
  alphas[0] = alphap; alphas[1] = alpham;
  int i; for(i=0;i<3;++i){
    Fimh[i] = (alphap*Fimo[i] + alpham*Fi[i] - alphap*alpham*(Ui[i]-Uimo[i]))/(
        alphap+alpham);
  }
}
void make_F(double ** U, double ** F, double **alpha, int NT, double * P){
  double alphas[2];

  int i,j; for(i=1;i<NT;++i){
    double Ui[3], Uimo[3]; getcol(U, i, Ui); getcol(U, i-1, Uimo);
    double Fimh[3]; Fi_mh(Ui, Uimo, Fimh, alphas, P[i], P[i-1]);
    for(j=0;j<2;++j){
      if(i>0 && i<NT-1){
  alpha[j][i-1]=alphas[j];
      }
    }
    for(j=0;j<3;++j){
      if(i>0 && i<NT){
      F[j][i] = Fimh[j];
      }
    }
  }
  for(j=0;j<3;++j){
    F[j][0] = F[j][NT-1];
    F[j][NT] = F[j][1];
  }
}
double get_step(double maxalpha, double dx){
  double dt = dx/(10*maxalpha);
  //dt = 0.00001;
  return(dt);
}
double get_maxalpha(double ** alpha, int NT){
  double max = fabs(alpha[0][0]); //printf("here is the first one%f\n", max);
  int i,j; for(i=0;i<NT-2;++i){
    for(j=0;j<2;++j){
      if(max<fabs(alpha[j][i])) max = fabs(alpha[j][i]);
      //printf("other alphas %f\n", alpha[j][i]);
    }
  }
  return(max);
}
```

```
double advance_system(double ** U, double dx , double t, int NT, double * P){

  double ** alpha; initnxN(&alpha, NT-2, 2);
  make_P(U, NT, P);
  double ** F; initnxN(&F, NT+1, 3); make_F(U, F, alpha, NT, P);
  double maxalpha = get_maxalpha(alpha, NT);
  double dt = get_step(maxalpha, dx);
  double Un[3];
  double L;
  int i,j; for(i=0;i<NT-1;++i){
    for(j=0;j<3;++j){
      L = - dt*(F[j][i+1]-F[j][i])/dx;
      Un[j] = U[j][i] + L;
      //printf("j=%d  L = %f\n", j, L);
      U[j][i] = Un[j];
    }
    //printf("i %d column %f  %f   %f\n", i,U[0][i],U[1][i],U[2][i]);
  }
  Reset_BC(U, NT);
  double tn;
  return(t+dt);
}
void write(double * P, FILE * fid, int NT, double dx){
  double x = -0.5;
  int i;for(i=0;i<NT-2;++i){
    fprintf(fid, "%e    %e    %d\n", x, P[i], i);
    //printf("%d %f\n", i,  P[i]);
    x+= dx;
  }
}
void getRow(double ** U, int NT, double * thing, int row){
  int count = 1;
  int i;for(i=1;i<NT-1;++i){
    thing[i-1] = U[row][i];
    //printf("%d %f\n", i-1, U[row][i]);
    count+=1;
  }
  //printf("before i write=%f NT-2=%d\n", thing[NT-3], NT-3);
}
double Entropy(double ** U, int NT, double dx, double * sumv, double * sump){
  double vsum = 0;
  double psum = 0;
  int i;for(i=1;i<NT-1;++i){
    double Ui[3]; getcol(U, i, Ui);
    double P = Pressure(Ui);
    double V = Velocity(Ui);
    double Pi = Pressurei(Ui[0]);
    double Vi = Velocityi(Ui[0]);
    vsum += dx*fabs(V - Vi);
    psum += dx*fabs(P - Pi);
  }
  *sumv = vsum; *sump = psum;
}

int main(int argc, char ** argv){
  // ./a.out Nx
  FILE * fid1, * fid2, * fid3;
  fid1 = fopen("initial2.dat", "w");
  fid3 = fopen("entropy.dat", "w");
  double T = atof(argv[1]); double t = 0;
  double L = 1.;   int Nx = 5000; double dx = L/(double)Nx;int NT = Nx + 2;
  double **U; double rho[Nx+2]; double P[Nx+2];double V[Nx+2];
  initnxN(&U, Nx+2, 3);
```

```
  initialize_system(U, rho, P, V, NT);
  //printf("i write=%f\n", U[0][NT-2]);
  double dense[Nx]; getRow(U, NT, dense, 0);
  write(dense, fid1, NT, dx);


  //advance_system( U , dx , t , NT , P );
  //printf("t=%f\n", t);
  //sleep(1);

  while(t<T){
    double tn = advance_system(U, dx, t, NT, P);
    double sumv, sump;
    Entropy(U, NT, dx, &sumv, &sump);
    fprintf(fid3, "%e   %e   %e\n", t, sumv, sump);
    //printf("t=%f\n", tn);
    //sleep(1);
    t = tn;
    //printf("i 21 column %f  %f  %f", U[0][21], U[1][21], U[2][21]);
    //break;

    }
  fid2 = fopen("final2.dat", "w");
  getRow(U, NT, dense, 0);
  write(dense, fid2, NT, dx);

  return(0);
}
```