Solving Euler Equation for Gas Dynamics by Computational Methods for Riemann Problem

Final Project

Course: MEEN 489/689 Special topic: Computational Fluid Dynamics

Prepared by,

Yatharth Kishor Vaishnani

327005677

Date: 12/12/2018

TABLE OF CONTENTS

1 Introduction

A major objective in computational fluid dynamics is the numerical solution of the Euler equations, which are a set of quasilinear hyperbolic equations governing adiabatic and inviscid flow. From the mathematical point of view, Euler equations are notably hyperbolic conservation equations in the case without external field. The conservation form emphasizes the mathematical interpretation of the equations as conservation equations through a control volume fixed in space, and is the most important for these equations also from a numerical point of view.

In this study, different finite volume methods are used to numerically analyze the compressible 1-D Euler equations for gas dynamics for the Riemann problem. A Riemann problem consists of an initial value problem composed of conservation equation together with piecewise data having a single discontinuity. The Godunov scheme, Runge-Kutta method and MUSCL scheme are used to numerically capture the discontinuity generated by the Sod shock tube problem and compared with the analytical solution. The MATLAB code for all the methods as well as for the analytical solution (reference mentioned in the code) are included in the Appendices (section 5).

Table 1 Different cases considered for study of computational methods for steady state 2-D heat diffusion

Case No.	Description	Grid points (N)	Time step (sec)	CFL Number ($C = {dt}/{dx}$)
1	Wave Propagation	100	0.0001	0.1
2	Sensitivity analysis	Varying from 100 to 1000	Varying from 0.0001 to 0.00001	0.1
5	Time complexity	Varying from 100 to 1000	Varying from 0.0001 to 0.00001	0.1

2 Method of Solution

For the case of 1-D unsteady gas dynamics, without heat transfer and without body forces, the governing equation for the system can be written as the Euler's equation. In general, the Euler equations in conservation vector form, reduces to

$$\frac{\partial \vec{Q}}{\partial t} + \frac{\partial \vec{E}(\vec{Q})}{\partial x} = 0 \tag{1}$$

where,

$$\vec{Q} = \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix}, \vec{E} = \begin{pmatrix} \rho u \\ p + \rho u^2 \\ Eu + pu \end{pmatrix} \tag{2}$$

where, $\vec{Q}$ is a vector of states and $\vec{E}(\vec{Q})$ is a vector of fluxes.

Equation (1) represents the conservation of mass, momentum and energy in one dimension. There are thus three equation and four variables – density (ρ), velocity (u), pressure (p) and total energy (E). In order to close the system, the equation of state is required, which is given by,

$$E = \frac{1}{2}\rho u^2 + \frac{p}{\gamma - 1} \tag{3}$$

where, γ is the ratio of specific heats at constant pressure and constant volume (here, γ = 1.4) for the fluid.

For solving the 1-D Euler's equation with different computational methods, the problem of Sod shock tube is considered, is a common test for the accuracy of computational fluid dynamics (CFD) codes. The test consists of a one-dimensional Reimann problem with the following initial condition of the above Euler's equation –

$$\vec{Q} = \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 2.5 \end{pmatrix} \qquad in\ (0,0.5) \tag{4}$$

$$\vec{Q} = \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix} = \begin{pmatrix} 0.125 \\ 0 \\ 0.25 \end{pmatrix} \qquad in\ (0.5,1) \tag{5}$$

The final time for the computational calculation is T = 0.1644 sec. The boundary conditions are considered of Diritchlet type (i.e., the conserved quantities take on the values specified by the initial conditions at either boundary). The solution with different CFD codes are obtained with 100 cells and CFL number 0.04.

For solving the Eq. (1) (i.e., combined Eq. (2) and (3)), there are several computational methods available. In this study, the solution is obtained by using Godunov scheme (1st order accurate), Runge-Kutta method (with artificial dissipation) and Godunov with MUSCL (Monotonic Upwind

Scheme for Conservation Laws). The results obtained by these methods are compared with the benchmark solutions for visualization of the computational accuracy.

2.1 Godunov Scheme

The Godunov scheme is conservative numerical scheme, suggested by S. K. Godunov in 1959. The Godunov scheme solves the exact, or approximate Riemann problems as a conservative Finite-Volume Method at each inter-cell boundary. Godunov scheme involves three basic steps in the algorithm - Reconstruct a piecewise polynomial function, Evolve the hyperbolic equation exactly (or approximately) with this initial data, Average this function over each grid cell to obtain new cell averages. Which is why it is also called as REA algorithm.

The steps for calculating the Euler equation as described before with the use of Godunov scheme are as below:

Step-1: Construct the $\vec{Q}_i^n$ and $\vec{E}(\vec{Q})_i^n$ vectors at n time level as the piece-wise constant approximation

In the Godunov scheme, we are assuming the state values $\{\bar{Q}_i^n\}$ as the piece-wise average values (zero order polynomials) of the actual state values at node $i, \{\vec{Q}_i^n\}$ as shown in the Fig. (1).

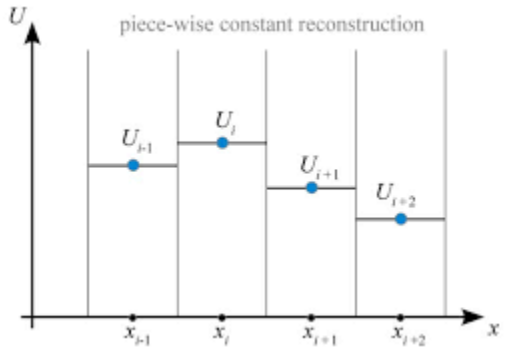

Figure 1 Piece-wise average values over the cell i

And the fluxes $\{\vec{E}(\vec{Q})_i^n\}$ are calculated at the cell boundaries $\left\{\vec{E}(\vec{Q})_{i+1/2}^n, \vec{E}(\vec{Q})_{i-1/2}^n\right\}$ from the cell average values of the state $\{\bar{Q}_i^n\}$.

Since, the piecewise constant approximation is an average of the solution over the cell of size Δx, the spatial error is of order of $O(\Delta x)$, and hence the resulting scheme will be first-order accurate in space.

Step-2: Obtain the solution for the Riemann problem

By using the approximate flux function for Riemann solver (e.g., Godunov flux, Enquist-Osher flux and Lax-Friedrich flux), calculate the solution for Riemann problem at cell boundaries.

Let's use the Lax-Friedrich flux function as shown in the Eq. (6)

$$f_i^n = \frac{1}{2}\left[\vec{E}(\vec{Q})^n_{i+1/2} + \vec{E}(\vec{Q})^n_{i-1/2}\right] - \frac{1}{2}|\alpha|(\bar{Q}^n_{i+1} - \bar{Q}^n_i) \quad (6)$$

where,

$$\alpha = \max\left|\frac{\partial E}{\partial Q}\right| \text{ (maximum eigen-value of the Jacobian matrix)} \quad (7)$$

Step-3: Update the solution at n+1 level from $\vec{Q}_i^{n+1}$

From the approximate flux function, the values of the state at the n+1 time level can be calculated as the Eq. (8).

$$\vec{Q}_i^{n+1} = \vec{Q}_i^n - \frac{\Delta t}{\Delta x}\left(f^n_{i+1/2} - f^n_{i-1/2}\right) \quad (8)$$

The state variables obtained after the step 3 are averaged over each cell defining a new piece-wise constant approximation resulting from the wave propagation during the time interval Δt.

2.2 Runge-Kutta Method (with Artificial Dissipation)

A numerical scheme commonly used to solve initial value problems for Ordinary Differential Equations is the Runge-Kutta (RK) method. This scheme essentially utilizes the weighted average of several solutions over the interval Δt in order to improve the accuracy of the solution. These methods were developed around 1900 by the German mathematicians Carl Runge and Martin Kutta. In computational analysis, Runge-Kutta methods are a family of implicit and explicit iterative methods. The most widely known method from this family is fourth order accurate Runge-Kutta method. Runge-Kutta schemes are typically expressed in explicit form. Implicit RK methods are computationally expensive and are rarely used. And as they are usually expressed in explicit form, these schemes are easy to program.

Runge-Kutta methods possess better stability criteria than comparable explicit schemes. For the linear hyperbolic equations, the stability requirement of explicit formulations is c ≤ 1. From the stability analysis of fourth order explicit RK method, with central differencing, it can be shown that the stability criteria is c ≤ 2√2. Moreover, it has higher order of accuracy i.e., fourth order.

However, the primary disadvantages of this scheme includes more computational time requirement per step and difficulty in error estimations.

A modified fourth order RK method for solving the Euler equation of the problem statement can be used as below:

Step-1: Construct the Q and E vectors as shown in the section 2.1

Step-2: Calculate the Q at (n+1) time step from the four step iterative method

$$\vec{Q}_i^1 = \vec{Q}_i^n \quad (9)$$

$$\vec{Q}_i^2 = \vec{Q}_i^n - \frac{\Delta t}{4}\left(\frac{\partial \vec{E}(\vec{Q})}{\partial x}\right)_i^1 \quad (10)$$

$$\vec{Q}_i^3 = \vec{Q}_i^n - \frac{\Delta t}{3}\left(\frac{\partial \vec{E}(\vec{Q})}{\partial x}\right)_i^2 \quad (11)$$

$$\vec{Q}_i^4 = \vec{Q}_i^n - \frac{\Delta t}{2}\left(\frac{\partial \vec{E}(\vec{Q})}{\partial x}\right)_i^3 \quad (12)$$

$$\vec{Q}_i^{n+1} = \vec{Q}_i^n - \Delta t\left(\frac{\partial \vec{E}(\vec{Q})}{\partial x}\right)_i^4 \quad (13)$$

where, a central difference approximation of second-order accuracy is used for the convective term $\frac{\partial \vec{E}(\vec{Q})}{\partial x}$, which can be shown as

$$\frac{\partial \vec{E}(\vec{Q})}{\partial x} = \frac{E_{i+1}^n - E_{i-1}^n}{2\Delta x} \quad (14)$$

Here, the presence of central difference scheme will produce the ringing in the results for a hyperbolic type of equation. To reduce the oscillations to an acceptable level, a damping term must be added.

$$D = e_1\left(\frac{\partial^2 Q}{\partial x^2}\right) - e_2\left(\frac{\partial^4 Q}{\partial x^4}\right) = e_1\left(\frac{Q_{i-1}^n - 2Q_i^n + Q_{i+1}^n}{\Delta x^2}\right) - e_2\left(\frac{Q_{i-2}^n - 4Q_{i-1}^n + 6Q_i^n - 4Q_{i+1}^n + Q_{i+2}^n}{\Delta x^4}\right) \quad (15)$$

where, e_1 and e_2 are small values. As shown in the Eq. (15), the damping term is calculated and added to the vector of state at n+1 time step after the Eq. (13).

$$Q_i^{n+1} = Q_i^{n+1} + D \quad (16)$$

Step-3: Update the solution at n+1 level from Q_i^{n+1} (as shown in section 2.1).

We should note here that the four step Runge-Kutta method is using the second order flux function (central difference scheme). Thus, overall accuracy of the scheme in space is of second order.

2.3 Godunov Scheme with MUSCL

MUSCL scheme is a Finite Volume method that can provide highly accurate numerical solutions for a given system, even in cases where the solutions exhibit shocks, discontinuities, or large gradients. MUSCL stands for Monotonic Upwind Scheme for Conservation Laws. This scheme was developed by Van Leer in 1979.

The MUSCL scheme is in general an idea to use the reconstructed states, derived from cell-averaged states instead of the piecewise constant approximation of Godunov's scheme, obtained from the previous time step. For example, the MUSCL scheme calculates the slope limited, reconstructed left and right states for each cell and are further used to calculate the fluxes at the cell boundaries. These fluxes are then used as input to Riemann solver, following which the solutions are averaged and used to advance the solution in time.

The MUSCL scheme can be explained as below to solve the Euler equation as described in the problem statement as below:

Step-1: Construct the $\vec{Q}_i^n$ and $\vec{E}(\vec{Q})_i^n$ vectors at n time level as the piece-wise linear approximation

In the Godunov scheme, we were assuming the values of the state were averaged using a piece-wise constant values, whereas in MUSCL scheme, the 1st order polynomial is used to calculate the values of state at cell boundaries as shown in the Fig. (2)

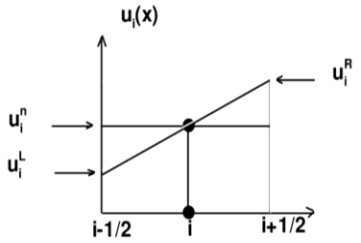

Figure 2 Piece-wise linear values over the cell i

And the fluxes $\{\vec{E}(\vec{Q})_i^n\}$ are calculated at the cell boundaries $\left\{\vec{E}(\vec{Q})_{i+1/2}^n, \vec{E}(\vec{Q})_{i-1/2}^n\right\}$ from the values of the state $\{\vec{Q}_i^n\}$ at cell boundaries $\left\{\vec{Q}_{i+1/2}^n, \vec{Q}_{i-1/2}^n\right\}$.

The slope of the 1st order polynomial is calculated by using the slope limiter functions and eventually approximating the value of the state vector to 2nd order.

$$\bar{Q}_i^n = \frac{1}{\Delta x}\int_{x_{(i-1/2)}}^{x_{(i+1/2)}} \vec{Q}^n(x)dx \tag{17}$$

$$\tilde{Q}_i^n(x) = \bar{Q}_i^n + O(\Delta x^2) \tag{18}$$

For this, we need to approximate the slope of the polynomial to 1st order and the state vector at any x $\{\tilde{Q}_i^n(x)\}$ can be written as

$$\tilde{Q}_i^n(x) = \bar{Q}_i^n + s_i(x - x_i),\ where\ x \in \left(x_{i-1/2}, x_{i+1/2}\right) \tag{19}$$

The slope is calculated from the minmod function as a slope limiter,

$$s_i = minmod\left\{\frac{\bar{Q}_{i+1}^n - \bar{Q}_i^n}{\Delta x}, \frac{\bar{Q}_i^n - \bar{Q}_{i-1}^n}{\Delta x}\right\} \tag{20}$$

where,

$$minmod(a,b) = \begin{cases} sign(a)\cdot \min(|a|,|b|), & a\cdot b > 0 \\ 0, & a\cdot b \le 0 \end{cases} \tag{21}$$

After calculating the values of state and flux vector at the cell boundaries $(i+1/2), (i-1/2)$ for the lower bound and the upper bound, we proceed further to calculate the approximate flux function to solve the Riemann problem.

Step-2: Obtain the solution for the Riemann problem

As shown in the section 2.1, by using values of state and flux vectors at each cell boundary, the Lax-Friedrich flux function is calculated as shown in the Eq. (14)

$$f^{n}_{i+1/2} = \frac{1}{2}\left[\vec{E}(\vec{Q})^{R}_{i+1/2} + \vec{E}(\vec{Q})^{L}_{i+1/2}\right] - \frac{1}{2}|\alpha|\left(\vec{Q}^{R}_{i+1/2} - \vec{Q}^{L}_{i+1/2}\right) \quad (22)$$

where, the superscript R and L are for right and left values of the vectors at cell boundary $(i + 1/2)$.

Step-3: Update the solution at n+1 level from $\vec{Q}_i^{n+1}$

From the approximate flux function, the values of the state at the n+1 time level can be calculated as the Eq. (8).

The state variables obtained after the step 3 are averaged over each cell defining a new piece-wise linear approximation resulting from the wave propagation during the time interval Δt.

3 Discussion of Results

3.1 Wave Propagation

The propagation of wave in the Sod's shock tube problem is calculated using the Godunov scheme, Runge-Kutta method with artificial dissipation and Godunov with MUSCL scheme. The primary results of these methods are shown in the Fig. (3), (4) and (5) along with the initial values and the analytical solution for the velocity, density and pressure terms.

For the solutions presented in this section, the value of $\Delta x = 0.001\ m$ and $\Delta t = 0.0001\ s$ is considered to generate them for each scheme.

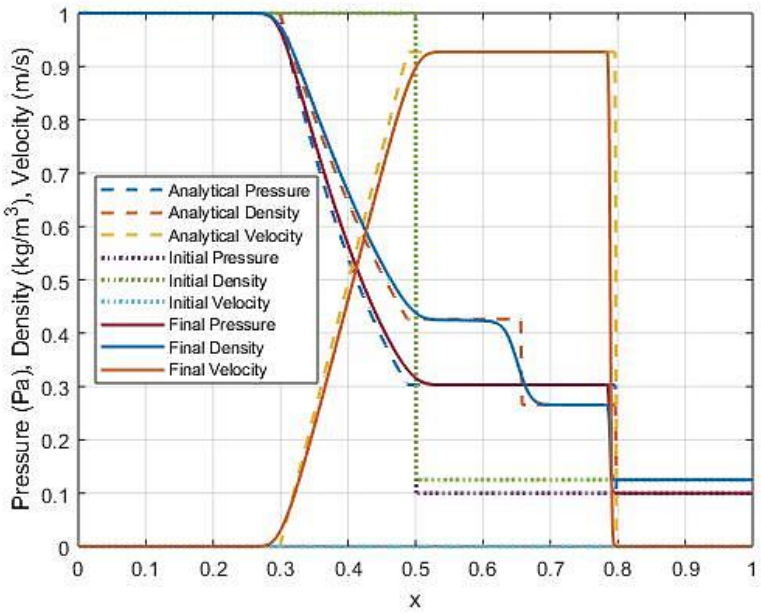

Figure 3 Godunov Scheme solution

As can be seen in the Fig. (3), (4) and (5), all these methods provide acceptable results when compared to the analytical solution of the problem. From the visual perspective, it can be noted that the MUSCL scheme is having the least dissipation compared to other two methods and thus provide quite similar results to the analytical solution.

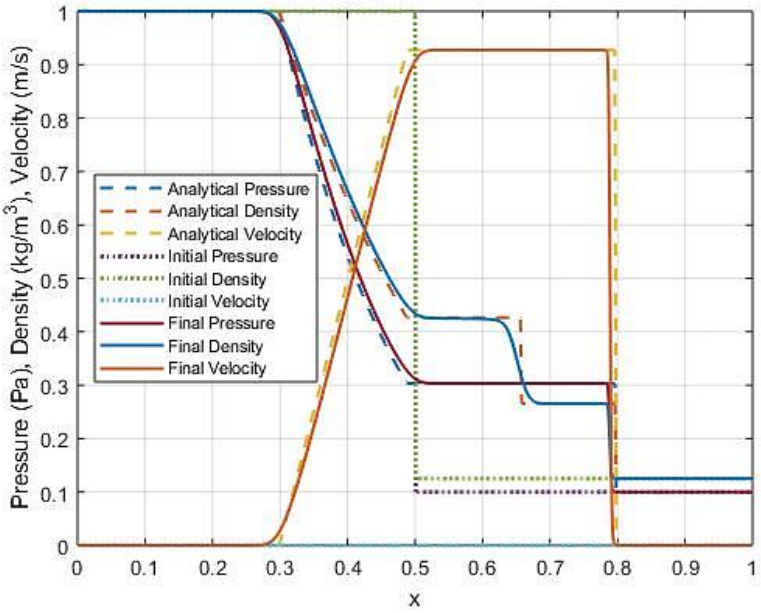

Figure 4 Runge-Kutta method with $e_1 = 0.05$

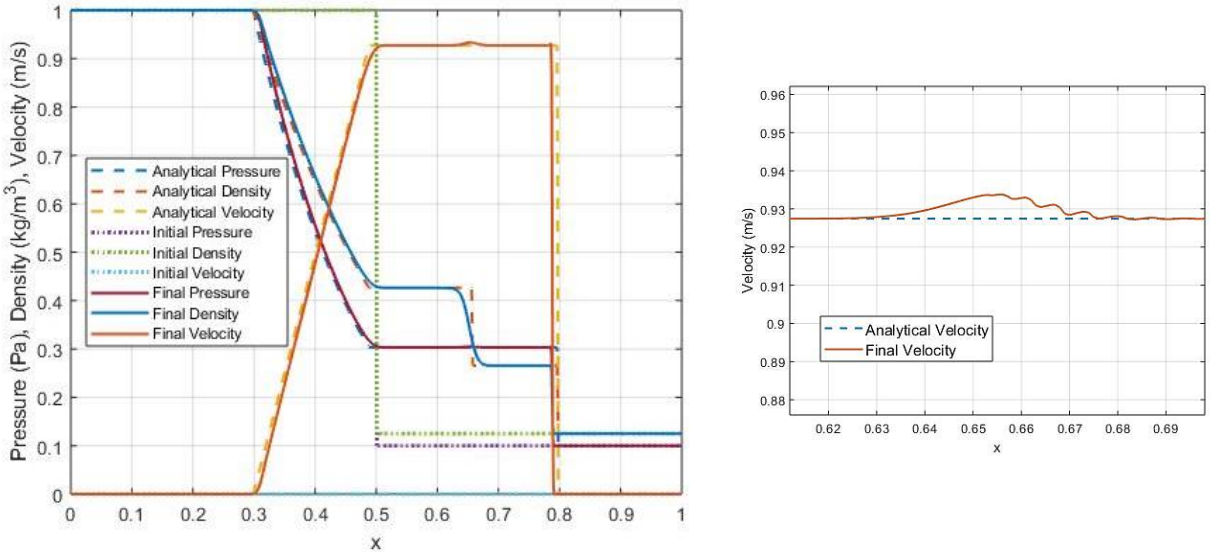

Figure 5 (a) Godunov scheme with MUSCL, (b) Slight oscillation in velocity profile

As we can see in the Fig. (3), (4) and (5), the propagation of the wave in the medium is effectively captured by all three methods. Here, the Godunov scheme is first order accurate, Runge-Kutta is second order accurate, and Godunov with MUSCL is second order accurate. Compared to the Godunov and Runge-Kutta method, MUSCL scheme is more accurate at the discontinuities. As we are adding a second order dissipation (with $e_1 = 0.05$) and which is why the scheme is becoming more dissipative. Although, there is a slight oscillation observed at $x = 0.65\ m$ in the MUSCL scheme, which is in acceptable terms.

3.2 Sensitivity Analysis

The sensitivity analysis for the grid and the time step is done for all three methods as shown in the Fig. (6), (7) and (8). Here, for the grid sensitivity analysis, due to the limitation of the CFL number on these explicit methods, the value of time step is changed for each scheme such as to keep the value of CFL number constant. The value of CFL number is fixed as 0.1 for all three methods and the results for the velocity profile are compared and presented in the Fig. (6), (7) and (8).

From the results shown, it can be concluded that for the grid with $\Delta x = 0.001\ m$ all three schemes becomes almost independent of the grid size. As it can be observed from the Fig. (8) that the MUSCL scheme is more sensitive to the grid size compared to the other schemes. The oscillation in the velocity profile with coarse grid size is the notation for this.

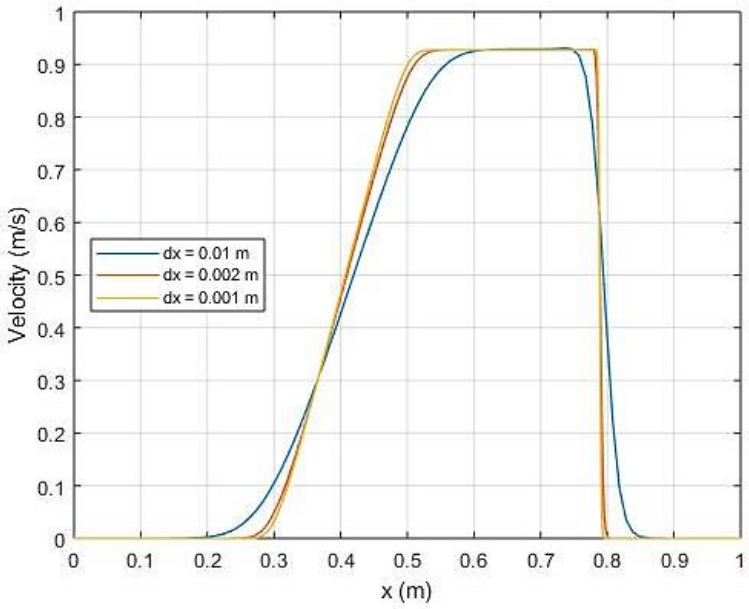

Figure 6 Grid sensitivity for Godunov scheme

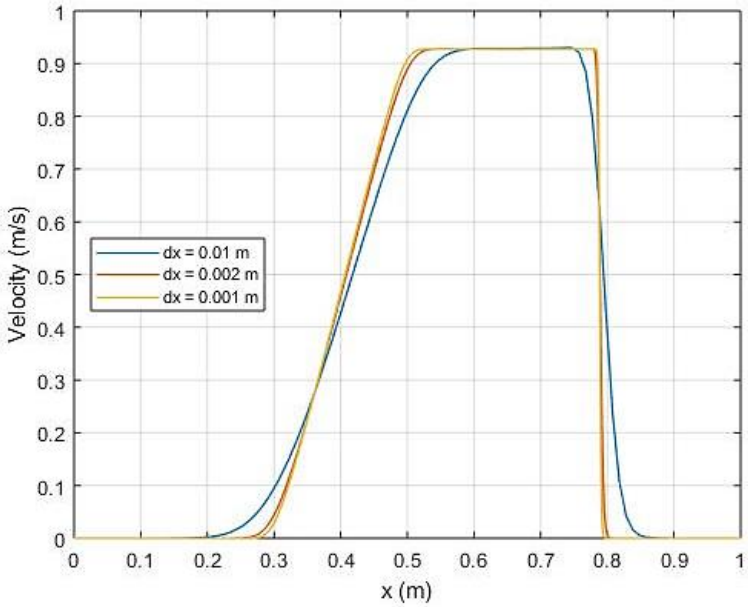

Figure 7 Grid sensitivity for Runge-Kutta method

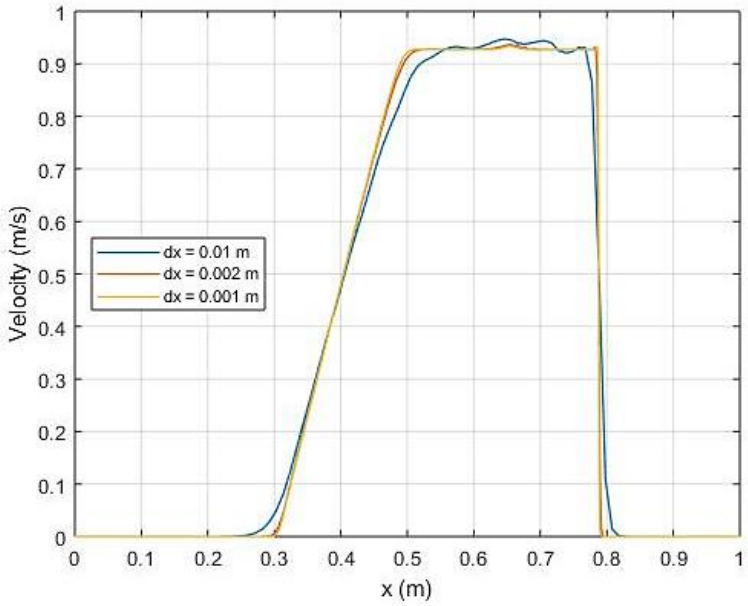

Figure 8 Grid sensitivity for MUSCL

3.3 Computational Cost and Accuracy

The complexity of the algorithm can be explained in terms of the computational time and space which are required for the algorithm to compute the given problem.

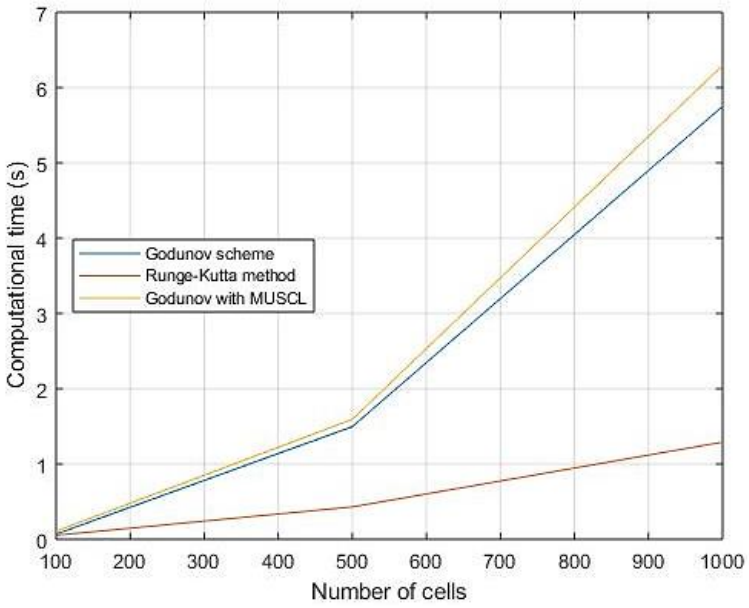

Figure 9 Computational time requirement for different methods

The computational time is calculated for all the methods by an in-built MATLAB function called "tic-toc", which measures the computational time required by the CPU for the given algorithm to run. The values obtained from the function is presented in the Fig. (9) as a graph for different number of cells in the domain. Here also, due to the requirement of these explicit algorithms, the value of the CFL number is kept constant at 0.1 for different grid sizes. Thus, with increase of grid size, the time step iterations are also increased proportionally and giving non-linear dependency of the computational time over the number of cells. From the Fig. (9), we can conclude that the Godunov and MUSCL scheme are highly time costly as compared to the Runge-Kutta method for higher size of grid.

For the space complexity, the first order accurate Godunov scheme is quite efficient compared to the Runge-Kutta method and MUSCL scheme, as we do not have to save the intermediate iterative values of state vector and flux vector in Godunov scheme. Also, compared to Runge-Kutta method, the MUSCL scheme is more costly in memory allocation.

For the accuracy of the computational results, the MUSCL scheme is more accurate compared to both the schemes and provides the most accurate capturing of the discontinuities over the domain for Riemann problems.

4 Summary and Conclusion

The computational methods – Godunov scheme, Runge-Kutta with artificial dissipation and Godunov with MUSCL schemes are used for the analysis of the Euler's equation in 1-D for the case of Sod's shock tube. Further analysis is done for the grid sensitivity and the computational complexity of the respective algorithms. We can conclude the following points form the study:

- All three methods provide accurate wave propagation compared to the benchmark results for the Sod's shock tube problem. For capturing the discontinuities, the MUSCL scheme is more accurate compared to the Runge-Kutta (with artificial dissipation) and Godunov scheme. For Runge-Kutta method, the acceptable results lies between the accuracy of the discontinuity and the amount of oscillations in the results.
- The sensitivity analysis concludes that the grid size considered for the results ($\Delta x = 0.001\ m$) is independent from the sensitivity towards the results of velocity profile. Also, the Godunov with MUSCL scheme is quite sensitive to the grid.
- From the comparison of all three methods for the computational complexity, the Runge-Kutta method is most efficient in terms of time required for the approximately same accurate results for given grid size. As both the Godunov scheme and Godunov with MUSCL scheme require the calculation of approximate flux function separately at each time step.

5 Appendices

5.1 Analytical Solution

```
% Exact solution of Riemann problem

% Theory in Section 10.2 of:

%   P. Wesseling: Principles of Computational Fluid Dynamics
%   Springer, Heidelberg, 2000 ISBN 3-5453-0. XII, 642 pp.
% See http://ta.twi.tudelft.nl/nw/users/wesseling/cfdbook.html

% This program is called by Euler schemes

% Functions called: f

global  PRL  CRL MACHLEFT  gamma  pleft  pright  rholeft  rhoright  uleft...
    uright  tend  lambda        % lambda = dt/dx

gammab = 1/(gamma - 1); gam1 = gamma-1;

% Assumed structure of exact solution
%
%    \         /      |con |       |s|
%     \   f   /       |tact|       |h|
% left \  a  /  state |disc| state |o| right
% state \ n /    2    |cont|   3   |c| state
%   1    \ /          |tinu|       |k|   4
%         |           |ity |       | |

PRL = pright/pleft;
cright = sqrt(gamma*pright/rhoright); cleft = sqrt(gamma*pleft/rholeft);
CRL = cright/cleft;
MACHLEFT = (uleft - uright)/cleft;

p34 = fzero('f',3);      % p34 = p3/p4
p3 = p34*pright;    alpha = (gamma+1)/(gamma-1);
rho3 = rhoright*(1+alpha*p34)/(alpha+p34);
rho2 = rholeft*(p34*pright/pleft)^(1/gamma);
u2 = uleft-uright+(2/(gamma-1))*cleft*...
    (1-(p34*pright/pleft)^((gamma-1)/(2*gamma)));
c2 = sqrt(gamma*p3/rho2);
spos = 0.5 + ...        % Shock position
    tend*cright*sqrt((gamma-1)/(2*gamma) + (gamma+1)/(2*gamma)*p34)+...
    tend*uright;

conpos = 0.5 + u2*tend + tend*uright;   % Position of contact discontinuity
pos1 = 0.5 + (uleft - cleft)*tend;  % Start of expansion fan
pos2 = 0.5 + (u2+uright-c2)*tend;   % End of expansion fan
xx = 0:0.002:1;
pexact = zeros(size(xx)); uexact= zeros(size(xx)); rhoexact =
zeros(size(xx));
machexact = zeros(size(xx));  cexact = zeros(size(xx));
```

```
for i = 1:length(xx)
  if xx(i) <= pos1
    pexact(i) = pleft;    rhoexact(i) = rholeft;
    uexact(i) = uleft;    cexact(i)   = sqrt(gamma*pexact(i)/rhoexact(i));
    machexact(i) = uexact(i)/cexact(i);
  elseif xx(i) <= pos2
    pexact(i) = pleft*(1+(pos1-xx(i))/(cleft*alpha*tend))^(2*gamma/(gamma-
1));
    rhoexact(i) = rholeft*(1+(pos1-xx(i))/(cleft*alpha*tend))^(2/(gamma-1));
    uexact(i) = uleft + (2/(gamma+1))*(xx(i)-pos1)/tend;
    cexact(i) = sqrt(gamma*pexact(i)/rhoexact(i));
    machexact(i) = uexact(i)/cexact(i);
  elseif xx(i) <= conpos
    pexact(i) = p3;           rhoexact(i) = rho2;
    uexact(i) = u2+uright;    cexact(i)   =
sqrt(gamma*pexact(i)/rhoexact(i));
    machexact(i) = uexact(i)/cexact(i);
  elseif xx(i) <= spos
    pexact(i) = p3;    rhoexact(i) = rho3;    uexact(i) = u2+uright;
    cexact(i) = sqrt(gamma*pexact(i)/rhoexact(i));
    machexact(i) = uexact(i)/cexact(i);
  else
    pexact(i) = pright;    rhoexact(i) = rhoright;
    uexact(i) = uright;    cexact(i)   = sqrt(gamma*pexact(i)/rhoexact(i));
    machexact(i) = uexact(i)/cexact(i);
  end
end
entroexact = log(pexact./rhoexact.^gamma);

subplot(2,3,1),         plot(xx,rhoexact)
subplot(2,3,2),     plot(xx,uexact)
subplot(2,3,3),     plot(xx,pexact)
subplot(2,3,4),     plot(xx,machexact)
subplot(2,3,5),         plot(xx,entroexact)
```

5.2 Godunov Scheme

```
%Gadunov scheme
clear
L = 1;
dx = 0.001;
x = (0:dx:L);
N = (L/dx)+1;
T = 0.1644;
dt = 0.0001;
n = (T/dt);
C = dt/dx;
gamma = 1.4;

%Initial conditions
for i = 1:N
    if x(i) <= 0.5
        rho(i) = 1;
        u(i) = 0;
        p(i) = 2.5*(gamma - 1);
```

```
    else
        rho(i) = 0.125;
        u(i) = 0;
        p(i) = 0.25*(gamma - 1);
    end
end
ET(:) = ((1/2).*rho(:).*(u(:).^2)) + (p(:)/(gamma - 1));
plot(x,p,':',x,rho,':',x,u,':','LineWidth',2)
hold on

%Time step calculations
for i = 1:n

    %1) Construct Q and E vectors

    Q(1,:) = rho(:);
    Q(2,:) = rho(:).*u(:);
    Q(3,:) = ET(:);

    E(1,:) = rho(:).*u(:);
    E(2,:) = E(1,:).*u(1,:) + p(1,:);
    E(3,:) = ET(:).*u(:) + p(:).*u(:);

    %2) Calculate Q(n+1) using Gadunov method
    Q1 = Q;
    flux = zeros(3,N);
    for k = 1:N-1
        c = sqrt(gamma*p(k)/rho(k));
        A = [abs(Q(2,k)) abs(Q(2,k)+c) abs(Q(2,k)-c)];
        al = max(A);
        flux(:,k) = 0.5*(E(:,k) + E(:,k+1)) - 0.5*(al)*(Q1(:,k+1) - Q1(:,k));
    end
    for k = 2:N-1
        Q(:,k) = Q1(:,k) - C*(flux(:,k) - flux(:,k-1));
    end

    %3) Update solution (rho,u,p)
    rho(:) = Q(1,:);
    u(1,:) = Q(2,:)./rho(1,:);
    ET(:) = Q(3,:);
    p(1,:) = (ET(1,:) - ((1/2).*rho(1,:).*(u(1,:).^2)))*(gamma - 1);

end

plot(x,p,x,rho,x,u,'LineWidth',1.5)
grid on
xlabel('x')
ylabel('Pressure (Pa), Density (kg/m^3), Velocity (m/s)')
legend({'Initial Pressure','Initial Density','Initial Velocity','Final
Pressure','Final Density','Final Velocity'},'Fontsize',8,'Location','west')
```

5.3 Runge-Kutta Method with Artificial Dissipation

```
clear
L = 1;
dx = 0.001;
x = (0:dx:L);
N = (L/dx)+1;
T = 0.1644;
dt = 0.0001;
n = (T/dt);
C = dt/dx;
gamma = 1.4;

%Initial conditions
for i = 1:N
    if x(i) <= 0.5
        rho(i) = 1;
        u(i) = 0;
        p(i) = 2.5*(gamma - 1);
    else
        rho(i) = 0.125;
        u(i) = 0;
        p(i) = 0.25*(gamma - 1);
    end
end
ET(:) = ((1/2).*rho(:).*(u(:).^2)) + (p(:)/(gamma - 1));

plot(x,p,':',x,rho,':',x,u,':','LineWidth',2)
hold on

%Time step calculations
for i = 1:n

    %1) Construct Q and E vectors

    Q(1,:) = rho(:);
    Q(2,:) = rho(:).*u(:);
    Q(3,:) = ET(:);

    E(1,:) = rho(:).*u(:);
    E(2,:) = E(1,:).*u(1,:) + p(1,:);
    E(3,:) = ET(:).*u(:) + p(:).*u(:);

    %2) Calculate Q(n+1) using RK method
    Ex = zeros(1,N);
    Q1 = Q;
    for j = 1:3
        for l = 1:4
            for k = 2:N-1
                Ex(j,k) = 0.5*(E(j,k+1) - E(j,k-1))/dx;
            end
            Q(j,:) = Q1(j,:) - (1/(5-l))*dt*(Ex(j,:));

            rho(:) = Q(1,:);
            u(1,:) = Q(2,:)./rho(1,:);
```

```
            ET(:) = Q(3,:);
            p(1,:) = (ET(1,:) - ((1/2).*rho(1,:).*(u(1,:).^2)))*(gamma - 1);

            E(1,:) = rho(:).*u(:);
            E(2,:) = E(1,:).*u(1,:) + p(1,:);
            E(3,:) = ET(:).*u(:) + p(:).*u(:);
        end
        Q(j,:) = Q(j,:) + Damp(Q1(j,:),N,0.05,0);
    end

    %3) Update solution (rho,u,p)
    rho(:) = Q(1,:);
    u(1,:) = Q(2,:)./rho(1,:);
    ET(:) = Q(3,:);
    p(1,:) = (ET(1,:) - ((1/2).*rho(1,:).*(u(1,:).^2)))*(gamma - 1);

end

plot(x,p,x,rho,x,u,'LineWidth',1.5)
grid on
xlabel('x')
ylabel('Pressure (Pa), Density (kg/m^3), Velocity (m/s)')
legend({'Initial Pressure','Initial Density','Initial Velocity','Final
Pressure','Final Density','Final Velocity'},'Fontsize',8,'Location','west')
```

5.4 Godunov Scheme with MUSCL

```
clear
L = 1;
dx = 0.001;
x = (0:dx:L);
N = (L/dx)+1;
T = 0.1644;
dt = 0.0001;
n = (T/dt);
C = dt/dx;
gamma = 1.4;

%Initial conditions
for i = 1:N
    if x(i) <= 0.5
        rho(i) = 1;
        u(i) = 0;
        p(i) = 2.5*(gamma - 1);
    else
        rho(i) = 0.125;
        u(i) = 0;
        p(i) = 0.25*(gamma - 1);
    end
end
ET(:) = ((1/2).*rho(:).*(u(:).^2)) + (p(:)/(gamma - 1));
plot(x,p,':',x,rho,':',x,u,':','LineWidth',2)
hold on

%Time step calculations
```

```
for i = 1:n

    %1) Construct Q and E vectors

    Q(1,:) = rho(:);
    Q(2,:) = rho(:).*u(:);
    Q(3,:) = ET(:);

    E(1,:) = rho(:).*u(:);
    E(2,:) = E(1,:).*u(1,:) + p(1,:);
    E(3,:) = ET(:).*u(:) + p(:).*u(:);

    %2) Calculate Q(n+1) using MUSCL scheme
    %2A) Construct Q(i+0.5) and Q(i-0.5)
    Qold = Q;
    QL = zeros(3,N);
    QR = zeros(3,N);
    s = zeros(3,N);

    for k = 2:N-1
        for j = 1:3
            s(j,k) = minmod((Qold(j,k+1) - Qold(j,k))/dx,(Qold(j,k) -
Qold(j,k-1))/dx);
        end
    end
        %storing QL(i+0.5) at QL(i) and QR(i-0.5) at QR(i-1)
    for k = 1:N-1
        QL(:,k) = Qold(:,k) + s(:,k).*(0.5*dx);
    end
    for k = 2:N
        QR(:,k-1) = Qold(:,k) - s(:,k).*(0.5*dx);
    end

    %2B) Construct Flux at (i+0.5) and (i-0.5)
    rhoL(:) = QL(1,:);
    rhoR(:) = QR(1,:);
    uL(1,:) = QL(2,:)./rho(1,:);
    uR(1,:) = QR(2,:)./rho(1,:);
    ETL(:) = QL(3,:);
    ETR(:) = QR(3,:);
    pL(:) = (ETL(:) - ((1/2).*rhoL(:).*(uL(:).^2)))*(gamma - 1);
    pR(:) = (ETR(:) - ((1/2).*rhoR(:).*(uR(:).^2)))*(gamma - 1);

    EL(1,:) = rho(1,:).*uL(1,:);
    ER(1,:) = rho(1,:).*uR(1,:);
    EL(2,:) = EL(1,:).*uL(1,:) + pL(1,:);
    ER(2,:) = ER(1,:).*uR(1,:) + pR(1,:);
    EL(3,:) = ET(1,:).*uL(1,:) + pL(1,:).*uL(1,:);
    ER(3,:) = ET(1,:).*uR(1,:) + pR(1,:).*uR(1,:);

    flux = zeros(3,N);
    for k = 1:N-1
        c = sqrt(gamma*p(k)./rho(k));
        A = [abs(Qold(2,k)) abs(Qold(2,k)+c) abs(Qold(2,k)-c)];
        al = max(A);
```

```
        for j=1:3
            flux(j,k) = 0.5*(ER(j,k) + EL(j,k)) - 0.5*(al)*(QR(j,k) -
QL(j,k));
        end
    end


    %2C) Calculate Q at n+1 time
    for k = 2:N-1
        Q(:,k) = Qold(:,k) - C*(flux(:,k) - flux(:,k-1));
    end

    %3) Update solution (rho,u,p)
    rho(:) = Q(1,:);
    u(1,:) = Q(2,:)./rho(1,:);
    ET(:) = Q(3,:);
    p(1,:) = (ET(1,:) - ((1/2).*rho(1,:).*(u(1,:).^2)))*(gamma - 1);

end

plot(x,p,x,rho,x,u,'LineWidth',1.5)
grid on
xlabel('x')
ylabel('Pressure (Pa), Density (kg/m^3), Velocity (m/s)')
legend({'Initial Pressure','Initial Density','Initial Velocity','Final
Pressure','Final Density','Final Velocity'},'Fontsize',8,'Location','west')
```

5.5 Sensitivity and Time Complexity Analysis

```
%Godunov with MUSCL
clear
a = 0;
for N = 100:100:1000
    a = a + 1;
    b(a) = N;
    L = 1;
    C = 0.1;
    T = 0.1644;
    gamma = 1.4;
    dx = L/N;
    x = linspace(0,L,N);
    dt = C*dx;
    n = (T/dt);
    %Initial conditions
    for i = 1:N
        if x(i) <= 0.5
            rho(i) = 1;
            u(i) = 0;
            p(i) = 2.5*(gamma - 1);
        else
            rho(i) = 0.125;
            u(i) = 0;
            p(i) = 0.25*(gamma - 1);
        end
        ET(i) = ((1/2).*rho(i).*(u(i).^2)) + (p(i)/(gamma - 1));
```

```
    end


    %Time steps
    tic
    for i = 1:n

        %1) Construct Q and E vectors

        Q = zeros(3,N);
        Q(1,:) = rho(:);
        Q(2,:) = rho(:).*u(:);
        Q(3,:) = ET(:);

        E = zeros(3,N);
        E(1,:) = rho(:).*u(:);
        E(2,:) = E(1,:).*u(1,:) + p(1,:);
        E(3,:) = ET(:).*u(:) + p(:).*u(:);

        %2) Calculate Q(n+1) using MUSCL scheme
        %2A) Construct u(i+0.5) and u(i-0.5)
        Qold = Q;

        QL = zeros(3,N);
        EL = zeros(3,N);
        rhoL = zeros(1,N);
        uL = zeros(1,N);
        ETL = zeros(1,N);
        pL = zeros(1,N);

        QR = zeros(3,N);
        ER = zeros(3,N);
        rhoR = zeros(1,N);
        uR = zeros(1,N);
        ETR = zeros(1,N);
        pR = zeros(1,N);
        s = zeros(3,N);

        for k = 2:N-1
            for j = 1:3
                s(j,k) = minmod((Qold(j,k+1) - Qold(j,k))/dx,(Qold(j,k) -
Qold(j,k-1))/dx);
            end
        end
            %storing QL(i+0.5) at QL(i) and QR(i-0.5) at QR(i-1)
        for k = 1:N-1
            QL(:,k) = Qold(:,k) + s(:,k).*(0.5*dx);
        end
        for k = 2:N
            QR(:,k-1) = Qold(:,k) - s(:,k).*(0.5*dx);
        end

        %2B) Construct Flux at (i+0.5) and (i-0.5)
        rhoL(:) = QL(1,:);
        rhoR(:) = QR(1,:);
```

```
        uL(1,:) = QL(2,:)./rho(1,:); %u1 is saving values of u(i-0.5) in
u1(1,:) and u(i+0.5) in u1(2,:)
        uR(1,:) = QR(2,:)./rho(1,:);
        ETL(:) = QL(3,:);
        ETR(:) = QR(3,:);
        pL(:) = (ETL(:) - ((1/2).*rhoL(:).*(uL(:).^2)))*(gamma - 1);
        pR(:) = (ETR(:) - ((1/2).*rhoR(:).*(uR(:).^2)))*(gamma - 1);

        EL(1,:) = rho(1,:).*uL(1,:);
        ER(1,:) = rho(1,:).*uR(1,:);
        EL(2,:) = EL(1,:).*uL(1,:) + pL(1,:);
        ER(2,:) = ER(1,:).*uR(1,:) + pR(1,:);
        EL(3,:) = ET(1,:).*uL(1,:) + pL(1,:).*uL(1,:);
        ER(3,:) = ET(1,:).*uR(1,:) + pR(1,:).*uR(1,:);

        flux = zeros(3,N); %flux1 is F(i-0.5) and flux2 is F(i+0.5)
        for k = 1:N-1
            c = sqrt(gamma*p(k)./rho(k));
            A = [abs(Qold(2,k)) abs(Qold(2,k)+c) abs(Qold(2,k)-c)];
            al = max(A);
            for j=1:3
                flux(j,k) = 0.5*(ER(j,k) + EL(j,k)) - 0.5*(al)*(QR(j,k) -
QL(j,k));
            end
        end


        %2C) Calculate Q at n+1 time
        for k = 2:N-1
            Q(:,k) = Qold(:,k) - C*(flux(:,k) - flux(:,k-1));
        end

        %3) Update solution (rho,u,p)
        rho(:) = Q(1,:);
        u(1,:) = Q(2,:)./rho(1,:);
        ET(:) = Q(3,:);
        p(1,:) = (ET(1,:) - ((1/2).*rho(1,:).*(u(1,:).^2)))*(gamma - 1);

    end
    t(a) = toc;
%{
    plot(x,u,'LineWidth',1)
    grid on
    xlabel('x (m)')
    ylabel('Velocity (m/s)')
    hold on
%}
end
%hold off
%legend({'dx = 0.01 m','dx = 0.002 m','dx = 0.001
m'},'Fontsize',8,'Location','west')
figure(1)
plot(b,t)
grid on
xlabel('Number of cells')
ylabel('Computational time (s)')
```

```
legend({'Godunov scheme','Runge-Kutta method','Godunov with
MUSCL'},'Fontsize',8,'Location','west')
hold off
```