Этот код реализует численное решение задачи о распаде разрыва (Sod shock tube problem) методом конечных объемов. Давайте разберем основные этапы и компоненты кода:

SodaEyler

clear all

close all

clc

N\_grid = 5;

for z = 1 : N\_grid

% начальные данные сетки

N\_(z) = 10\*2^(z-1);

N = N\_(z);

a = 0;

b = 1;

gamma = 1.4;

h = (b - a) / N;

x = a + h/2:h:b - h/2;

t = 0;

t\_end = 0.2;

K = 0.1;

tetha = 0.5;

tau = 1e-5;

% начальные условия задачи сода

for i = 1:N

if x(i) <= 0.5

rho(i) = 1;

P(i) = 1;

u(i) = 0;

else

rho(i) = 0.125;

P(i) = 0.1;

u(i) = 0;

end

end

% запись вектора консервативных переменных и потоков

e = P ./ (rho .\* (gamma - 1));

U = [rho

rho .\* u

rho .\* e + (rho .\* u.^2) ./ 2];

Здесь создается цикл для различных размеров сетки (разное количество ячеек). В каждой итерации цикла вычисляются начальные условия для задачи о распаде разрыва.

while t <= t\_end

% деконструкция консерв. переменных

% значения в ячейках

UGCell = [U(:,1) U(:,:) U(:,N)];

dUdx = zeros(3, N);

% аппроксимсация производной

for j = 1:3

Um = zeros(3, N);

for i = 2:N

Um(1, i) = tetha \* 1/h \* (UGCell(j, i) - UGCell(j, i-1));

Um(2, i) = tetha \* 1/(2 \* h) \* (UGCell(j,i+1) - UGCell(j, i-1));

Um(3, i) = tetha \* 1/h \* (UGCell(j, i+1) - UGCell(j, i));

end

for i = 1:N

if (Um(1, i) > 0 && Um(2, i) > 0 && Um(3, i) > 0)

dUdx(j, i) = min([Um(1, i) Um(2, i) Um(3, i)]);

end

if (Um(1, i) < 0 && Um(2, i) < 0 && Um(3, i) < 0)

dUdx(j, i) = max([Um(1, i) Um(2, i) Um(3, i)]);

if dUdx(j,i) < 0

dUdx(j, i) = 0.0;

end

end

end

end

for j = 1:3

% граничные условия слева

Ur(j, 1) = U(j, 1) - h/2 \* dUdx(j, 1);

Ul(j, 1) = U(j, 1);

for i = 2:N

Ur(j, i) = U(j, i) - h/2 \* dUdx(j, i);

Ul(j, i) = U(j, i-1) + h/2 \* dUdx(j, i-1);

end

% граничные условия справа

Ul(j, N+1) = U(j, N) + h/2 \* dUdx(j, N);

Ur(j, N+1) = U(j, N);

end

for i = 1:N+1

Fr(1, i) = Ur(2, i);

Fr(2, i) = 1/2 \* (3 - gamma) \* (Ur(2, i)^2 / Ur(1, i)) + (gamma - 1) \* Ur(3, i);

Fr(3, i) = gamma \* (Ur(2, i) / Ur(1, i)) \* Ur(3, i) - (1/2) \* (gamma - 1) \* (Ur(2, i)^3 / Ur(1, i)^2);

Fl(1, i) = Ul(2, i);

Fl(2, i) = 1/2 \* (3 - gamma) \* (Ul(2, i)^2 / Ul(1, i)) + (gamma - 1) \* Ul(3, i);

Fl(3, i) = gamma \* (Ul(2, i) / Ul(1, i)) \* Ul(3, i) - (1/2) \* (gamma - 1) \* (Ul(2, i)^3 / Ul(1, i)^2);

end

% скорость распространения возмущений

speedL = Ul(2,:) ./ Ul(1,:);

speedR = Ur(2,:) ./ Ur(1,:);

Pl = (gamma - 1) \* (Ul(3,:) - 0.5 \* (Ul(2,:).^2 / Ul(1,:)));

Pr = (gamma - 1) \* (Ur(3,:) - 0.5 \* (Ur(2,:).^2 / Ur(1,:)));

cl = ((gamma .\* Pl) ./ Ul(1,:)).^(1/2);

cr = ((gamma .\* Pr) ./ Ur(1,:)).^(1/2);

for i = 1:N+1

% модуль скоростей возмущений

amax(i) = max([abs(speedL(i)+cl(i)) abs(speedL(i)-cl(i)) abs(speedR(i)+cr(i)) abs(speedR(i)-cr(i))]);

end

Flux = zeros(3,N+1);

% значения потоков на границах ячеек

for j = 1:3

Flux(j, :) = (Fr(j, :) + Fl(j, :)) ./ 2 - amax ./ 2 .\* (Ur(j, :) - Ul(j, :));

end

Происходит обновление консервативных переменных на новом временном слое с использованием метода Эйлера.

% значения консервативных переменных на новом временном слое

for j = 1:3

for i = 1:N

dUdt(j, i) = (Flux(j, i+1) - Flux(j, i));

end

end

% разностная аппроксимация выражения по времени методом Эйлера

% temp = dUdt .\* tau / h;

U = U - dUdt .\* tau / h;

% скорость

vel = U(2,:) ./ U(1,:);

% давление

P = (gamma - 1) \* (U(3,:) - 0.5 \* (U(2,:).^2 ./ U(1,:)));

% Чтобы не уйти за рамки по времени

if t + tau > t\_end

tau = t\_end - t;

else

tau = (K \* h) / max(amax);

end

% шаг по времени

t = t + tau;

end

% load Result.txt

%

% figure(2);

% plot(Result(:,1),Result(:,2),'k:','linewidth',2); % ѕлотность

% hold on

% figure(3);

% plot(Result(:,1),Result(:,4),'k:','linewidth',2); % ?авление

% hold on

% figure(1);

% plot(Result(:,1),Result(:,3),'k:','linewidth',2); % Cкорость

% hold on

% grid on;

Норма сравнивается с аналитическим решением задачи, и результаты сохраняются для последующего анализа сходимости численного метода.

data = analyticSod(t\_end,N);

norma(z) = h\*sum(abs(U(1,:)-data.rho'));

end

for z = 1 :N\_grid-1

N\_\_(z) = N\_(z);

order(z) = log2(norma(z)/norma(z+1));

end

figure(5);

plot(N\_\_,order,'b-o','linewidth',2);

grid on;

hold on;

figure(4);

loglog(N\_,norma,'b-o','linewidth',2);

grid on;

hold on;

% построение решения

figure(1);

plot(x,vel,'bo','MarkerSize',4);

hold on;

plot(data.x,data.u,'k','LineWidth',2);

xlabel('пространство');

ylabel('скорость');

title('численное решение');

grid on

figure(2);

plot(x,U(1,:),'bo','MarkerSize',4);

hold on;

plot(data.x,data.rho,'k','LineWidth',2);

xlabel('пространство');

ylabel('плотность');

title('численное решение');

grid on

figure(3);

plot(x,P,'bo','MarkerSize',4);

hold on

plot(data.x,data.P,'k','LineWidth',2);

xlabel('пространство');

ylabel('давление');

title('численное решение');

grid on

% time = 0.165;

% data = analyticSod(time);

% figure(4);

% % subplot(2,2,1),

% plot(data.x,data.rho,'-b','LineWidth',2);

% xlabel('x (m)');

% ylabel('Density (kg/m^3)');

% title('Plot of Density vs Position');

% grid on;

% figure(5);

% % subplot(2,2,2),

% plot(data.x,data.P,'-g','LineWidth',2);

% xlabel('x (m)');

% ylabel('Pressure (Pa)');

% title('Plot of Pressure vs Position');

% grid on;

% % subplot(2,2,3),

% figure(6);

% plot(data.x,data.u,'-r','LineWidth',2);

% xlabel('x (m)');

% ylabel('Velocity (m/s)');

% title('Plot of Velocity vs Position');

% grid on;

% figure(7);

% % subplot(2,2,4),

% plot(data.x,data.e,'-k','LineWidth',2);

% xlabel('x (m)');

% ylabel('Specific Internal Energy (J/kg)');

% title('Plot of Internal Energy vs Position');

% grid on;

sodFunc.m

function y = sodFunc(P)

%defines function to be used in analytic\_sod

%Initial conditions

rho\_l = 1;

P\_l = 1;

u\_l = 0;

rho\_r = 0.125;

P\_r = 0.1;

u\_r = 0;

gamma = 1.4;

mu = sqrt( (gamma-1)/(gamma+1) );

y = (P - P\_r)\*(( ((1 - mu^2)^2)\*((rho\_r\*(P + mu\*mu\*P\_r))^-1) )^(0.5))...

- 2\*(sqrt(gamma)/(gamma - 1))\*(1 - power(P, (gamma - 1)/(2\*gamma)));

end

sodDemo.m

%A demo script file to show the use of analytic\_sod.m

time = 0.165;

data = analyticSod(time);

figure,

subplot(2,2,1),

plot(data.x,data.rho,'-b','LineWidth',2);

xlabel('x (m)');

ylabel('Density (kg/m^3)');

title('Plot of Density vs Position');

grid on;

subplot(2,2,2),

plot(data.x,data.P,'-g','LineWidth',2);

xlabel('x (m)');

ylabel('Pressure (Pa)');

title('Plot of Pressure vs Position');

grid on;

subplot(2,2,3),

plot(data.x,data.u,'-r','LineWidth',2);

xlabel('x (m)');

ylabel('Velocity (m/s)');

title('Plot of Velocity vs Position');

grid on;

subplot(2,2,4),

plot(data.x,data.e,'-k','LineWidth',2);

xlabel('x (m)');

ylabel('Specific Internal Energy (J/kg)');

title('Plot of Internal Energy vs Position');

grid on;

analyticSod.m

function [data] = analyticSod(t,n\_points)

%to solve Sod's Shock Tube problem

%reference: "http://www.phys.lsu.edu/~tohline/PHYS7412/sod.html"

% | | | | |

% | | | | |

% | | | | |

%\_\_\_|\_\_\_\_\_\_\_|\_\_\_|\_\_\_\_\_|\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

% x1 x2 x0 x3 x4

%

%input require: t (time)

if nargin < 1

%set default value

t = 0.2;

end

%Initial conditions

x0 = 0.5;

rho\_l = 1;

P\_l = 1;

u\_l = 0;

rho\_r = 0.125;

P\_r = 0.1;

u\_r = 0;

gamma = 1.4;

mu = sqrt( (gamma-1)/(gamma+1) );

%speed of sound

c\_l = power( (gamma\*P\_l/rho\_l),0.5);

c\_r = power( (gamma\*P\_r/rho\_r),0.5);

P\_post = fzero('sodFunc',pi);

v\_post = 2\*(sqrt(gamma)/(gamma - 1))\*(1 - power(P\_post, (gamma - 1)/(2\*gamma)));

rho\_post = rho\_r\*(( (P\_post/P\_r) + mu^2 )/(1 + mu\*mu\*(P\_post/P\_r)));

v\_shock = v\_post\*((rho\_post/rho\_r)/( (rho\_post/rho\_r) - 1));

rho\_middle = (rho\_l)\*power((P\_post/P\_l),1/gamma);

%Key Positions

x1 = x0 - c\_l\*t;

x3 = x0 + v\_post\*t;

x4 = x0 + v\_shock\*t;

%determining x2

c\_2 = c\_l - ((gamma - 1)/2)\*v\_post;

x2 = x0 + (v\_post - c\_2)\*t;

%start setting values

% n\_points = 1000; %set by user

%boundaries (can be set)

x\_min = 0;

x\_max = 1;

x = linspace(x\_min,x\_max,n\_points);

data.x = x';

data.rho = zeros(n\_points,1); %density

data.P = zeros(n\_points,1); %pressure

data.u = zeros(n\_points,1); %velocity

data.e = zeros(n\_points,1); %internal energy

for index = 1:n\_points

if data.x(index) < x1

%Solution b4 x1

data.rho(index) = rho\_l;

data.P(index) = P\_l;

data.u(index) = u\_l;

elseif (x1 <= data.x(index) && data.x(index) <= x2)

%Solution b/w x1 and x2

c = mu\*mu\*((x0 - data.x(index))/t) + (1 - mu\*mu)\*c\_l;

data.rho(index) = rho\_l\*power((c/c\_l),2/(gamma - 1));

data.P(index) = P\_l\*power((data.rho(index)/rho\_l),gamma);

data.u(index) = (1 - mu\*mu)\*( (-(x0-data.x(index))/t) + c\_l);

elseif (x2 <= data.x(index) && data.x(index) <= x3)

%Solution b/w x2 and x3

data.rho(index) = rho\_middle;

data.P(index) = P\_post;

data.u(index) = v\_post;

elseif (x3 <= data.x(index) && data.x(index) <= x4)

%Solution b/w x3 and x4

data.rho(index) = rho\_post;

data.P(index) = P\_post;

data.u(index) = v\_post;

elseif x4 < data.x(index)

%Solution after x4

data.rho(index) = rho\_r;

data.P(index) = P\_r;

data.u(index) = u\_r;

end

data.e(index) = data.P(index)/((gamma - 1)\*data.rho(index));

end

end