```cpp
#include <iostream>
#include <math.h>
#include <cmath>
#include <conio.h>
#include <vector>
#include <iomanip>
#include <algorithm>
#include <numeric>
#include <fstream>
using namespace std;
using std::cout;
//Global variables
//double p_init = 101325; // Pascals
//double T_init = 293.15; // Kelvin
//double rho_init = 1.204;// kg/m^3
//double u_init = 0;      //m/s
double Gamma = 1.4;
double R_univ = 287.085635359116; // R_air
const int i_max = 64;  // check with 32 cells
double i_max_duplicate = i_max;
double d_x = 2/(double(i_max));  //2/i_max+1
double CFL= 0.01;                  // for Euler explicit eqn use cfl <= 1
double x_min = -1;
double x_max = 1;
double range = x_max-x_min;
double nozzle_total_length = 2;  //meters
const int n_max = 200000;
double Width = 1;
double p_stag = 300000;
double T_stag = 600;
double rho_stag = p_stag/(R_univ*T_stag);

double primitive_variable_n[i_max+2][3] = {0};

double conserved_variable_n[i_max+2][3] = {0};
double conserved_variable_n_plus_1[i_max+2][3] = {0};

double psi_n[i_max+2] = {0};
double T_n[i_max+2] = {0};
double u_n[i_max+2] = {0};
double total_energy_n[i_max+2]= {0};
double total_enthalpy_n[i_max+2]= {0};
double p_n[i_max+2]= {0};
double rho_n[i_max+2]= {0};
double M_n[i_max+2]= {0};

double Vol[i_max+2]= {0};
double d1_plus_half[i_max+1]= {0};
double d2_plus_half[i_max+1]= {0};
double d3_plus_half[i_max+1]= {0};

double F1_plus_half[i_max+1] ={0};

double F2_plus_half[i_max+1] = {0};

double F3_plus_half[i_max+1] = {0};

double Source[i_max+2]= {0};

double M_minus_1 = 0 ;
double psi_bc_minus_1 = 0;
double T_minus_1 = 0;
double p_minus_1 = 0 ;
double rho_minus_1 = 0;
double a_minus_1 = 0;
double u_minus_1 = 0;
double total_enthalpy_minus_1 = 0;
double total_energy_minus_1 = 0;


double M_plus_2 = 0;
double psi_bc_plus_2 = 0;
double T_plus_2 = 0;
double p_plus_2 = 0;
double rho_plus_2 = 0;
double a_plus_2 = 0;
double u_plus_2 = 0;
double total_enthalpy_plus_2 = 0;
double total_energy_plus_2 = 0;


vector<double> artificial_dissipation_output;
double residual_eq1[i_max+1];
double residual_eq2[i_max+1];
double residual_eq3[i_max+1];

double L_1_norm_eq1[n_max];
double L_2_norm_eq1[n_max];
double L_inf_norm_eq1[n_max];

double L_1_norm_eq2[n_max];
double L_2_norm_eq2[n_max];
double L_inf_norm_eq2[n_max];

double L_1_norm_eq3[n_max];
double L_2_norm_eq3[n_max];
double L_inf_norm_eq3[n_max];


   double conv_eq1_L2[n_max] = {0};
   double conv_eq2_L2[n_max] = {0};
   double conv_eq3_L2[n_max] = {0};

   double conv_eq1_L1[n_max] = {0};
   double conv_eq2_L1[n_max] = {0};
   double conv_eq3_L1[n_max] = {0};

   double conv_eq1_Linf[n_max] = {0};
   double conv_eq2_Linf[n_max] = {0};
   double conv_eq3_Linf[n_max] = {0};


void calculate_boundary_conditions_from_print_variables(int n)
{
     M_n[0] = (2.0*M_n[1])- M_n[2];

     if (M_n[0] < (0.18999999999999995/10))
     {

      M_n[0] = 0.18999999999999995/10;

     }
     double psi_bc_0 = 1.0+ ((Gamma-1.0)/2.0)*M_n[0]*M_n[0];

     T_n[0] = T_stag / psi_bc_0;
     p_n[0] = (p_stag) / pow(psi_bc_0,(Gamma/(Gamma-1.0)));
     rho_n[0] = p_n[0]/(R_univ*T_n[0]);
     double a_0 = sqrt(Gamma*R_univ*T_n[0]);
     u_n[0] = M_n[0] * a_0;

     total_enthalpy_n[0] = (((Gamma*R_univ)/(Gamma-1.0))*T_n[0]) +
(u_n[0]*u_n[0]/2.0);
     total_energy_n[0] = total_enthalpy_n[0] - (p_n[0]/rho_n[0]);

// i_max+1 cell :

     M_n[i_max+1] = (2.0*M_n[i_max]) - M_n[i_max-1];

      if (M_n[i_max+1] < (0.18999999999999995/10))
     {

      M_n[i_max+1] = 0.18999999999999995/10;

     }
     double psi_bc_max_plus_1 = 1.0 + ((Gamma-
1.0)/2.0)*M_n[i_max+1]*M_n[i_max+1];
     T_n[i_max+1] = T_stag / psi_bc_max_plus_1;
     p_n[i_max+1] = p_stag / pow(psi_bc_max_plus_1,(Gamma/(Gamma-1)));
     rho_n[i_max+1] = p_n[i_max+1]/(R_univ*T_n[i_max+1]);
     double a_i_max_plus_1 = sqrt(Gamma*R_univ*T_n[i_max+1]);
     u_n[i_max+1] = M_n[i_max+1] * a_i_max_plus_1;

     total_enthalpy_n[i_max+1] = (((Gamma*R_univ)/(Gamma-1))*T_n[i_max+1]) +
(u_n[i_max+1]*u_n[i_max+1]/2.0);
     total_energy_n[i_max+1] = total_enthalpy_n[i_max+1] -
(p_n[i_max+1]/rho_n[i_max+1]);
}
double primitive_to_conserved_variable(int n)

{

for(int i = 0; i <= i_max+1; i++)
{
conserved_variable_n[i][0] = primitive_variable_n[i][0];
// if (conserved_variable_n[i][0] < (0.110323511064052582/1000))
//   {

//     conserved_variable_n[i][0] = 0.110323511064052582/1000;
//     cout<< " the 0th conserved variable has gone small and we are using the
limiter"<<endl;

//   }

conserved_variable_n[i][1] =
primitive_variable_n[i][0]*primitive_variable_n[i][1];
// if (conserved_variable_n[i][1] < (0.110323511064052582*57.252777099609375
)/1000)
//   {

//     conserved_variable_n[i][1] =
0.110323511064052582*57.252777099609375/1000;
//     cout<< " the first conserved variable has gone small and we are using
the limiter"<<endl;

//   }


conserved_variable_n[i][2] = (primitive_variable_n[i][2]/(Gamma-1.0)) +
0.5*(primitive_variable_n[i][0]*pow(primitive_variable_n[i][1],2));
// if (conserved_variable_n[i][2] <
(432172.919382919092*0.110323511064052582)/1000) // reduce the value for
energy to 432172.919382919092 if this limiter is giving problems.
//   {
```

```
//     conserved_variable_n[i][2] =
432172.919382919092*0.110323511064052582/1000;
//     cout<< " the second conserved variable has gone small and we are using
the limiter"<<endl;

//   }
    //U(:,3) = ( V(:,3)/(gamma - one) ) + half*V(:,1)*V(:,2)**2
}

return 0;
}

void update_domain_print_variables(int n)
// remember that these are still without boundary values.
{
   for(int i = 1; i <= i_max ; i++)
   {
rho_n[i] = primitive_variable_n[i][0];
u_n[i] = primitive_variable_n[i][1];
p_n[i] = primitive_variable_n[i][2];
T_n[i] = primitive_variable_n[i][2]/ ( primitive_variable_n[i][0]*R_univ);
M_n[i] = u_n[i]/sqrt(Gamma*R_univ*T_n[i]);
if (M_n[i]<0)
{
   cout<< " the Mach number has gone negative"<<endl;
}
total_energy_n[i] =
conserved_variable_n_plus_1[i][2]/conserved_variable_n_plus_1[i][0];
   }
}

double conserved_to_primitive_variable_to_print_variable(int n)

{
for(int i = 0; i <= i_max+1 ; i++)

{
primitive_variable_n[i][0] = conserved_variable_n_plus_1[i][0];
//  if (primitive_variable_n[i][0] < (0.110323511064052582/1000))
//   {

//     primitive_variable_n[i][0] = 0.110323511064052582/1000;
//     cout<< " the density has gone small and we are using the
limiter"<<endl;

//   }
```

```
primitive_variable_n[i][1]
=  conserved_variable_n_plus_1[i][1]/conserved_variable_n_plus_1[i][0];
 if (primitive_variable_n[i][1] < (57.252777099609375/1000))
  {

    primitive_variable_n[i][1] = 57.252777099609375/1000;
    cout<< " the velocity has gone negative and we are using the
limiter"<<endl;

  }
primitive_variable_n[i][2] = (Gamma-1.0)*conserved_variable_n_plus_1[i][2] -
(0.5*(Gamma-
1.0)*pow(conserved_variable_n_plus_1[i][1],2)/conserved_variable_n_plus_1[i][0
]);

// if (primitive_variable_n[i][2] < (6302.525390625/1000))
//   {

//     primitive_variable_n[i][2] = 6302.525390625/1000;
//     cout<< " the pressure has gone small and we are using the
limiter"<<endl;

//   }

//V(:,3) = (gamma - one)*U(:,3) - half*(gamma - one)*U(:,2)**2/U(:,1)

}
//calculate_boundary_conditions_from_print_variables(n);
return 0;
}


// setting geometry
double x_location(double i)
{
    double x;
    if (i == 0)
    {
    x = x_min - (i+0.5)*d_x;
    return x;
    }
    else if ( i == 0.5 )
    {
    x = x_min ;
    return x;
    }
    else
    {
   if (std::fmod(i,1) == 0)
   {
    x = x_min + (i-0.5)*d_x;
   return x;
    }
    else
    {
    x = x_min + (i-0.5)*d_x;
    return x;
    }
    }
}
double find_area(double i)
{
    // change how area is taken.
double x = x_location(i);
double area_x = 0;
area_x = 0.2 + 0.4 * ( 1 + sin ( M_PI * ( x - 0.5)));
//cout << " area at this "<< x << "  is = " << area_x<<endl;
return area_x;

}

double set_initial_condition_primitive_variable(int n) // checked manually
seems ok!
{
   if (n==0)
   {
   for (int i = 1; i <= i_max; i++)
   {
      //M_n[i] = (x_location(i)*0.9) + 1;
      M_n[i] = (x_location(i)*1.4) + 1.6;
      // M_n[i] = M_n[i-1] + 0.1;
      // M_n[0] = 0.1;
     double psi = 1 + (((Gamma-1.0)/2.0)*M_n[i]*M_n[i]);
     T_n[i] = T_stag/psi;
     p_n[i] = p_stag/pow(psi,(Gamma/(Gamma-1)));
     rho_n[i] = p_n[i]/(R_univ*T_n[i]);
     double a = sqrt(Gamma*R_univ*T_n[i]);
     u_n[i] = M_n[i] * a;
     total_enthalpy_n[i] = (((Gamma*R_univ)/(Gamma-1.0))*T_n[i]) +
(u_n[i]*u_n[i]/2.0);
     total_energy_n[i] = total_enthalpy_n[i] - (p_n[i]/rho_n[i]) ;

   //   cout << " Values at i = " << i <<endl;
   //   cout << "d_x"<<d_x<<endl;
   //   cout << " Values for x_location = " << x_location(i)<<
std::setprecision(14)<<endl;
   //   cout << " Area is = " << find_area(i)<<endl;
   //   cout << " Values for M_n = " << M_n[i]<<endl;
   //   cout << " Values for u_n = " << u_n[i]<<endl;
   //   cout << " Values for p_n = " << p_n[i]<<endl;
   //   cout << " Values for rho_n = "<< rho_n[i]<<endl;
   //   cout << " Values of total_enthaly_n" << total_enthalpy_n[i]<<endl;
   //   cout << " Values of total_energy_n" << total_energy_n[i]<<endl;
    primitive_variable_n[i][0] = rho_n[i];
    primitive_variable_n[i][1] = u_n[i];
    primitive_variable_n[i][2] = p_n[i];

   }

   }
return 0;
}

double set_initial_boundary_conditions_insetropic(int n)
{
   calculate_boundary_conditions_from_print_variables(n);

primitive_variable_n[0][0] = rho_n[0];
primitive_variable_n[0][1] = u_n[0];
primitive_variable_n[0][2] = p_n[0];

primitive_variable_n[i_max+1][0] = rho_n[i_max+1];
primitive_variable_n[i_max+1][1] = u_n[i_max+1];
primitive_variable_n[i_max+1][2] = p_n[i_max+1];

return 0;
}

double defect_test_initial_values(int n) // values for 10 cells + 2 ghost
cells
{

if ((M_n[0] - 0.018999999999999996 >= pow(10,-10)) \
|| (rho_n[0] - 1.7413262797928382 >= pow(10,-10))\
||(u_n[0] - 9.3300355351865267 >= pow(10,-10))\
||(p_n[0] - 299924.20231370459 >= pow(10,-10)))
{
cout<<"check the initial condition, the first ghost cell is a
problem"<<endl;
}

if((M_n[i_max+1] - 1.9899999999999998 >= pow(10,-10))\
||(rho_n[i_max+1] - 0.40513653635616093 >= pow(10,-10))\
||(u_n[i_max+1] - 730.0070000457597 >= pow(10,-10))\
||(p_n[i_max+1] - 38942.270715840248 >= pow(10,-10)))
```

```
{
cout<<"check the initial condition the last ghost cell is a problem"<<endl;
}

return 0;
}

// double set_boundary_conditions_normal_shock()
// {
// p_n[i_max+1] = 120000;
// return 0;
// }

    void defect_test_extra_ghost_cells(int n)
    {
      if (n == 0)
      {
if((M_plus_2 - 2.16999999999999904 >= pow(10,-10))\
||(rho_plus_2 - 0.331480879643353987 >= pow(10,-10))\
||(u_plus_2 - 764.72466233891646 >= pow(10,-10))\
||(p_plus_2 - 29404.9992095327907 >= pow(10,-10))\
||(total_energy_plus_2 - 514171.854351441318 >= pow(10,-10)))
{
cout<<"check the initial condition the extra ghost cell on the right is a
problem"<<endl;
}

if((M_minus_1 - 0.189999999999999995/100 >= pow(10,-10))\
||(rho_minus_1 - 1.7416374625490771 >= pow(10,-10))\
||(u_minus_1 - 0.933036897512747776>= pow(10,-10))\
||(p_minus_1 - 299999.24190123158 >= pow(10,-10))\
||(total_energy_minus_1 - 430628.57740408147 >= pow(10,-10)))
{
cout<<"check the initial condition the extra ghost cell on the left is a
problem"<<endl;
}
      }
    }

double set_extra_ghost_cells(int n)
{
   // 0 minus 1 cell :
   M_minus_1 = (2 * M_n[0]) - M_n[1] ;
if (M_minus_1 < (0.189999999999999995/100))
     {

      M_minus_1= 0.189999999999999995/100;
     }
     psi_bc_minus_1 = 1.0+ ((Gamma-1.0)/2.0)*M_minus_1*M_minus_1;

     T_minus_1 = T_stag /psi_bc_minus_1;
     p_minus_1 = (p_stag) / pow(psi_bc_minus_1,(Gamma/(Gamma-1)));
     rho_minus_1 = p_minus_1/(R_univ*T_minus_1);
     a_minus_1 = sqrt(Gamma*R_univ*T_minus_1);
     u_minus_1 = M_minus_1 * a_minus_1;

     total_enthalpy_minus_1 = (((Gamma*R_univ)/(Gamma-1.0))*T_minus_1) +
(u_minus_1*u_minus_1/2.0);
     total_energy_minus_1 = total_enthalpy_minus_1 - (p_minus_1/rho_minus_1);

// i_max+2 cell :
  M_plus_2 =  M_n[i_max+1] +  (abs(M_n[i_max] - M_n[i_max+1]));

  if (M_plus_2 < (0.18999999999999995/100))
     {

      M_plus_2= 0.18999999999999995/100;

     }

// if (M_plus_2 < M_n[i_max+1])
// {
//     cout << " M_plus_2 is less thta M_n[i_max+1]"<<endl;
// }


psi_bc_plus_2 = 1.0+ ((Gamma-1.0)/2.0)*M_plus_2*M_plus_2;

     T_plus_2 = T_stag / psi_bc_plus_2;
     p_plus_2 = (p_stag) / pow(psi_bc_plus_2,(Gamma/(Gamma-1)));
     rho_plus_2 = p_plus_2/(R_univ*T_plus_2);
     a_plus_2 = sqrt(Gamma*R_univ*T_plus_2);
     u_plus_2 = M_plus_2 * a_plus_2;

    total_enthalpy_plus_2 = (((Gamma*R_univ)/(Gamma-1.0))*T_plus_2) +
(u_plus_2*u_plus_2/2.0);
    total_energy_plus_2 = total_enthalpy_plus_2 - (p_plus_2/rho_plus_2);

    defect_test_extra_ghost_cells(n);


return 0 ;
}

double compute_lambada_max(int i)
{
 double lambda_max = 0;
 double a = 0;
 a = sqrt(Gamma*R_univ*T_n[i]);
 lambda_max = abs(u_n[i]) + a;
 return lambda_max;


}



double stability(double d_x,int i)
{
   double delta_t = 0;
   delta_t= CFL * (d_x/(compute_lambada_max(i)));
   return delta_t;
}

double find_max_nu(int i)
{

   double max_nu = 0;
   double nu_i = 0;
   double nu_i_minus_1 = 0 , nu_i_plus_1 = 0, nu_i_plus_2 = 0;

 nu_i_minus_1 = abs( (p_n[i+1-1] - (2*p_n[i-1]) + p_n[i-1-1] )/ (p_n[i+1-1] +
(2*p_n[i-1]) + p_n[i-1-1]) );
 nu_i =         abs( (p_n[i+1] - (2*p_n[i]) + p_n[i-1] )/ (p_n[i+1] +
(2*p_n[i]) + p_n[i-1]) ) ;
 nu_i_plus_1 =  abs( (p_n[i+1+1] - (2*p_n[i+1]) + p_n[i-1+1] )/ (p_n[i+1+1] +
(2*p_n[i+1]) + p_n[i-1+1]) );
 nu_i_plus_2 =  abs( (p_n[i+1+2] - (2*p_n[i+2]) + p_n[i-1+2] )/ (p_n[i+1+2] +
(2*p_n[i+2]) + p_n[i-1+2]) ) ;


 max_nu  = max({nu_i_minus_1,nu_i,nu_i_plus_1,nu_i_plus_2});

if (i == 0)
 {
 nu_i =         abs( (p_n[i+1] - (2*p_n[i]) + p_minus_1 )/ (p_n[i+1] +
(2*p_n[i]) + p_minus_1) ) ;
 nu_i_plus_1 =  abs( (p_n[i+1+1] - (2*p_n[i+1]) + p_n[i-1+1] )/ (p_n[i+1+1] +
(2*p_n[i+1]) + p_n[i-1+1]) );
 nu_i_plus_2 =  abs( (p_n[i+1+2] - (2*p_n[i+2]) + p_n[i-1+2] )/ (p_n[i+1+2] +
(2*p_n[i+2]) + p_n[i-1+2]) ) ;

 max_nu  = max({nu_i,nu_i_plus_1,nu_i_plus_2});

//  if ((max_nu - 0.010505298195820)  >= pow(10,-10))
//  {
//    cout << " The max_nu for 0th cell has changed and needs checking!" <<
endl;
//  }
 }
 if (i==1)
 {
 nu_i_minus_1 = abs( (p_n[i+1-1] - (2*p_n[i-1]) + p_minus_1 )/ (p_n[i+1-1] +
(2*p_n[i-1]) + p_minus_1) );
 nu_i =         abs( (p_n[i+1] - (2*p_n[i]) + p_n[i-1] )/ (p_n[i+1] +
(2*p_n[i]) + p_n[i-1]) ) ;
 nu_i_plus_1 =  abs( (p_n[i+1+1] - (2*p_n[i+1]) + p_n[i-1+1] )/ (p_n[i+1+1] +
(2*p_n[i+1]) + p_n[i-1+1]) );
 nu_i_plus_2 =  abs( (p_n[i+1+2] - (2*p_n[i+2]) + p_n[i-1+2] )/ (p_n[i+1+2] +
(2*p_n[i+2]) + p_n[i-1+2]) ) ;


 max_nu  = max({nu_i_minus_1,nu_i,nu_i_plus_1,nu_i_plus_2});


 }
 if (i == (i_max-1))
 {

 nu_i_minus_1 = abs( (p_n[i+1-1] - (2*p_n[i-1]) + p_n[i-1-1] )/ (p_n[i+1-1] +
(2*p_n[i-1]) + p_n[i-1-1]) );
 nu_i =         abs( (p_n[i+1] - (2*p_n[i]) + p_n[i-1] )/ (p_n[i+1] +
(2*p_n[i]) + p_n[i-1]) ) ;
 nu_i_plus_1 =  abs( (p_n[i+1+1] - (2*p_n[i+1]) + p_n[i-1+1] )/ (p_n[i+1+1] +
(2*p_n[i+1]) + p_n[i-1+1]) );
 nu_i_plus_2 =  abs( (p_plus_2 - (2*p_n[i+2]) + p_n[i-1+2] )/ (p_plus_2 +
(2*p_n[i+2]) + p_n[i-1+2]) ) ;

 max_nu  = max({nu_i_minus_1,nu_i,nu_i_plus_1,nu_i_plus_2});

 }
 if (i == i_max)
 {

 nu_i_minus_1 = abs( (p_n[i+1-1] - (2*p_n[i-1]) + p_n[i-1-1] )/ (p_n[i+1-1] +
(2*p_n[i-1]) + p_n[i-1-1]) );
 nu_i =         abs( (p_n[i+1] - (2*p_n[i]) + p_n[i-1] )/ (p_n[i+1] +
(2*p_n[i]) + p_n[i-1]) ) ;
 nu_i_plus_1 =  abs( (p_plus_2 - (2*p_n[i+1]) + p_n[i-1+1] )/ (p_plus_2 +
(2*p_n[i+1]) + p_n[i-1+1]) );

 max_nu  = max({nu_i_minus_1,nu_i,nu_i_plus_1});
 }
 return max_nu;
}



double compute_Epsilon_2_i_plus_half(int k)
{

double Kappa_2 = 0.5; // range is from 0.25 to 0.5
double Epsilon_2_i_plus_half = Kappa_2 * find_max_nu(k) ;
return Epsilon_2_i_plus_half;
//cout << " Epsilon_2_i_plus_half = " << Epsilon_2_i_plus_half<<endl ; //
should decrease as i increases.
}

double compute_Epsilon_4_i_plus_half(int k)
{
double Kappa_2 = 0.5; // range is from 0.25 to 0.5
double Kappa_4 = 1.0/32.0; // ramge is from 0.015625 to 0.03125
double zero = 0;
double Epsilon_4_i_plus_half = 0;
Epsilon_4_i_plus_half = max(zero,(Kappa_4 -
compute_Epsilon_2_i_plus_half(k)));
return Epsilon_4_i_plus_half;
//cout << " Epsilon_4_i_plus_half = " << Epsilon_4_i_plus_half<<endl;
}


double compute_lambda_i_plus_half(int k)
{
   double lambda_i_plus_half = 0.5*(compute_lambada_max(k)+
compute_lambada_max(k+1));
   return lambda_i_plus_half;
   //cout << " lambda_i_plus_half = " << lambda_i_plus_half<<endl;

}

double print_flow_variables(std::ofstream& outfile, int n)
{
   if ((std::fmod(n,500) == 0 ) || (n == n_max))
   {
      outfile.seekp(0, std::ios::end);
      outfile << "zone T="<<"'"<<n<<"'" <<endl;
      outfile << "I="<< i_max+2+2 <<endl<<"DATAPACKING =
POINT"<<endl<<"DT=(DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE
DOUBLE)"<<endl;
      outfile << scientific << setprecision(15);
      outfile << x_location(0-1) <<" "<< rho_minus_1<<" "<< u_minus_1 <<"
"<<p_minus_1<<" "<<total_energy_minus_1<<" "<< M_minus_1<<" " << T_minus_1<<
endl;
      for ( int j = 0; j<= i_max+1 ; j++)
      {
        outfile.seekp(0, std::ios::end);
        outfile << x_location(j) <<" "<< rho_n[j]<<" "<< u_n[j] <<"
"<<p_n[j]<<" "<<total_energy_n[j]<<" "<< M_n[j]<<" " << T_n[j] << endl;
      }

       outfile.seekp(0, std::ios::end);
       outfile << x_location(i_max+2) <<" "<< rho_plus_2<<" "<< u_plus_2 <<"
"<<p_plus_2<<" "<<total_energy_plus_2<<" "<< M_plus_2<<" " << T_plus_2 <<
endl;

   }


return 0;
}



double print_norm_file(int n, double print_eq1_L1, double print_eq1_L2, double
print_eq1_Linf, double print_eq2_L1, double print_eq2_L2,double
print_eq2_Linf,double print_eq3_L1,double print_eq3_L2,double print_eq3_Linf)
 {
    if (std::fmod(n,1000) == 0)
    {
     static bool file_initialized = false;
    std::ofstream outfile("Norms.dat",file_initialized ? std::ios_base::app :
std::ios_base:: trunc);  // create a file output stream to write data to
"output.dat"

 if (!file_initialized)
 {
outfile << "'TITLE = Norms of the solution'" << endl;
outfile <<"DATAPACKING = POINT"<<endl<<"DT=(DOUBLE DOUBLE DOUBLE DOUBLE DOUBLE
DOUBLE DOUBLE DOUBLE)"<<endl;
string variables = "variables = 'conv_eq_1_L1' 'conv_eq_2_L1'
'conv_eq_3_L1'  'conv_eq_1_L2' 'conv_eq_2_L2'  'conv_eq_3_L2' 'conv_eq_1_Linf'
'conv_eq_2_Linf'  'conv_eq_3_Linf' ";
outfile << variables  << endl;
file_initialized = true;
 }

        outfile.seekp(0, std::ios::end);
        outfile << "zone T="<<"'"<<n<<"'" <<endl;
        outfile.seekp(0, std::ios::end);
        outfile << scientific << setprecision(15);
        outfile << print_eq1_L1 <<" "<< print_eq2_L1<<" "<<print_eq3_L1<<" "<<
print_eq1_L2 <<" "<< print_eq2_L2<<" "<<print_eq3_L2<<" "<< print_eq1_Linf<<"
"<< print_eq2_Linf<<" "<<print_eq3_Linf <<endl;

    }
return 0;

    }


double compute_norms(int n)
{

//    double sum_1 = 0;
//    double sum_2 = 0;
//    double sum_3 = 0;
//    double sum_of_squares_1 = 0;
//    double sum_of_squares_2 = 0;
//    double sum_of_squares_3 = 0;
//    double residual_eq1_positive[i_max+1] = {0};
//    double residual_eq2_positive[i_max+1] = {0};
//    double residual_eq3_positive[i_max+1] = {0};


   double sum_1 = 0;
   double sum_2 = 0;
   double sum_3  = 0;
   double sum_of_squares_1 = 0;
   double sum_of_squares_2 = 0;
   double sum_of_squares_3 = 0 ;
   double residual_eq1_positive[i_max+1] = {0};
   double residual_eq2_positive[i_max+1] = {0};
   double residual_eq3_positive[i_max+1] = {0};

   double temp_var_1 = 0;
   double temp_var_2 = 0;
   double temp_var_3 = 0;

    // calcularting norms at every time step


for(int v = 1; v <=i_max; v++)
{
residual_eq1_positive[v] = abs(residual_eq1[v]);
residual_eq2_positive[v] = abs(residual_eq2[v]);
residual_eq3_positive[v] = abs(residual_eq3[v]);
```

```cpp
}

 for(int c = 1; c <=i_max; c++)
  {
    sum_1 = sum_1 + abs(residual_eq1_positive[c]);
    sum_2 = sum_2 + abs(residual_eq2_positive[c]);
    sum_3=  sum_3 + abs(residual_eq3_positive[c]);

   sum_of_squares_1 = sum_of_squares_1 + pow(residual_eq1_positive[c],2);
   sum_of_squares_2 = sum_of_squares_2 + pow(residual_eq2_positive[c],2);
   sum_of_squares_3 = sum_of_squares_3 + pow(residual_eq3_positive[c],2);

  }

L_1_norm_eq1[n] = sum_1*(1/(double(i_max)));
L_2_norm_eq1[n] = sqrt(sum_of_squares_1/double(i_max));
double *max_it_1 = max_element((residual_eq1_positive),residual_eq1_positive +
(i_max+1));
temp_var_1 = *max_it_1;
L_inf_norm_eq1[n] = temp_var_1;

L_1_norm_eq2[n] = sum_2*(1/double(i_max));
L_2_norm_eq2[n] = sqrt(sum_of_squares_2/double(i_max));
double *max_it_2 = max_element(residual_eq2_positive,residual_eq2_positive +
(i_max+1));
temp_var_2 = *max_it_2;
L_inf_norm_eq2[n] = temp_var_2;

L_1_norm_eq3[n] = sum_3*(1/double(i_max));
L_2_norm_eq3[n] = sqrt(sum_of_squares_3/double(i_max));
double *max_it_3 = max_element(residual_eq3_positive, residual_eq3_positive +
(i_max+1));
temp_var_3 = *max_it_3;
L_inf_norm_eq3[n] = temp_var_3;

// cout << " L_1_norm_eq1[n] = "<< L_1_norm_eq1[n] <<endl;
// cout << " L_1_norm_eq2[n] = " << L_1_norm_eq2[n] <<endl;
// cout << " L_1_norm_eq3[n] = "<< L_1_norm_eq3[n] << endl;

// cout << " L_2_norm_eq1[n] = "<< L_2_norm_eq1[n] <<endl;
// cout << " L_2_norm_eq2[n] = " <<L_2_norm_eq2[n] <<endl;
// cout << " L_2_norm_eq3[n] = "<< L_2_norm_eq3[n] << endl;

// cout << " L_inf_norm_eq1[n] = " << L_inf_norm_eq1[n] << endl;
// cout << " L_inf_norm_eq2[n] = " << L_inf_norm_eq2[n] << endl;
// cout << " L_inf_norm_eq3[n] = " << L_inf_norm_eq3[n] << endl;

if (n % 100 == 0 )
```

```
    {
    conv_eq1_L1[n] = L_1_norm_eq1[n]/L_1_norm_eq1[2];
    conv_eq2_L1[n] = L_1_norm_eq2[n]/L_1_norm_eq2[2];
    conv_eq3_L1[n] = L_1_norm_eq3[n]/L_1_norm_eq3[2];

    conv_eq1_L2[n] = L_2_norm_eq1[n]/L_2_norm_eq1[2];
    conv_eq2_L2[n] = L_2_norm_eq2[n]/L_2_norm_eq2[2];
    conv_eq3_L2[n] = L_2_norm_eq3[n]/L_2_norm_eq3[2];

   conv_eq1_Linf[n] = L_inf_norm_eq1[n]/L_inf_norm_eq1[2];
   conv_eq2_Linf[n] = L_inf_norm_eq2[n]/L_inf_norm_eq2[2];
   conv_eq3_Linf[n] = L_inf_norm_eq3[n]/L_inf_norm_eq3[2];

   cout<< " Time step = " << n << endl;

   //  cout << "conv_eq1_L1[n] " <<conv_eq1_L1[n]<<endl;
   //  cout << "conv_eq2_L1[n] " <<conv_eq2_L1[n]<<endl;
   //  cout << "conv_eq3_L1[n] " <<conv_eq3_L1[n]<<endl;


   //  cout << "conv_eq1_L2[n] " <<conv_eq1_L2[n]<<endl;
   //  cout << "conv_eq2_L2[n] " <<conv_eq2_L2[n]<<endl;
   //  cout << "conv_eq3_L2[n] " <<conv_eq3_L2[n]<<endl;


   //  cout << "conv_eq1_Linf[n] " <<conv_eq1_Linf[n]<<endl;
   //  cout << "conv_eq2_Linf[n] " <<conv_eq2_Linf[n]<<endl;
   //  cout << "conv_eq3_Linf[n] " <<conv_eq3_Linf[n]<<endl;

    print_norm_file(n,conv_eq1_L1[n],conv_eq1_L2[n],conv_eq1_Linf[n],conv_eq2_
L1[n],conv_eq2_L2[n],conv_eq2_Linf[n],conv_eq3_L1[n],conv_eq3_L2[n],conv_eq3_L
inf[n]);

}
return 0;
}

void print_artificial_dissipation(int k, int n, double d1_eq1,double d2_eq2,
double d3_eq3, double d1_eq1_0, double d2_eq2_0, double d3_eq3_0)
{
    static bool file_initialized = false;
     std::ofstream outfile("artificial_dissipation.dat", file_initialized ?
std::ios_base::app : std::ios_base::trunc);
 if (!file_initialized)
 {
outfile << "'TITLE = artificial_dissipation'" << endl;
string variables = " variables = 'i+half' 'x_location' 'd1_plus_half_eq1'
'd2_plus_half_eq2' 'd3_plus_half_eq3' ";
outfile << variables  << std::endl;
```

```
file_initialized = true;
 }


    if (k==1) {
    outfile.seekp(0, std::ios::end);
    outfile << "zone T="<<"'"<<n<<"'" <<endl;
    outfile << "I="<< i_max+1 <<endl<<"DATAPACKING = POINT"<<endl<<"DT=(DOUBLE
DOUBLE DOUBLE DOUBLE)"<<endl;
    outfile << scientific << setprecision(15);
    outfile << "0" <<" "<< x_location(0 + 0.5) <<" "<< d1_eq1_0<<" "<<
d2_eq2_0 <<" "<<d3_eq3_0<<" "<< endl;
    }
        outfile.seekp(0, std::ios::end);
        outfile << scientific << setprecision(15);
        outfile << k <<" "<< x_location(k+0.5) <<" "<< d1_eq1<<" "<< d2_eq2
<<" "<<d3_eq3<<" "<< endl;
}

void defect_test_artificial_dissipation_0th_interface(int n)
 {

   if (n == 0)
   {
   if ((d1_plus_half[0] - (-0.14360693849040107) >= pow(10,-5))\
   || (d2_plus_half[0]) - (-1503.7597232288381) >= pow(10,-5)\
   || (d3_plus_half[0]) - (-54476.439692112268) >= pow(10,-5))

   {
      cout << "If this is a 10 cell mesh then there is a problem with the 0th
interface artificial dissipation "<<endl;
   }
   }
 }


 void defect_test_artificial_dissipation_i_max_interface(int n )
 {

   if (n == 0)
   {
   if ((d1_plus_half[i_max] - (0.50882177071265888) >= pow(10,-5))\
   || (d2_plus_half[i_max]) - (308.31681642011949) >= pow(10,-5)\
   || (d3_plus_half[i_max]) - (242018.65017890101) >= pow(10,-5))

   {
      cout << "If this is a 10 cell mesh then there is a problem with the
i_max interface artificial dissipation "<<endl;
   }
   }
 }


vector<double>artificial_dissipation(int k, int n)
{
// k = i from main loop of Euler equation;
// n= n from main loop of Euler equation;

double D_1_U_1[i_max+1] = {0};
double D_1_U_2[i_max+1] = {0};
double D_1_U_3[i_max+1] = {0};

double D_3_U_1[i_max+1] = {0};
double D_3_U_2[i_max+1] = {0};
double D_3_U_3[i_max+1] = {0};

//U_1[k] = rho_n[k];
//U_2[k] = rho_n[k]*u_n[k];
//U_3[k] = rho_n[k]*total_energy_n[k];

 for (int l = 1; l < i_max; l++) // for every i, it should give a value of
damping between i and i+1. // stencil goes ouutside the boundary. // simple
extrapolation
 {

   // if ( (l = i_max) && (k = i_max))
// {
//     cout << " check now";
//     }
// (((Gamma*R_univ)/(Gamma-1.0))*T_n[0]) + (u_n[0]*u_n[0]/2.0);
double total_enthalpy_art_dissi_loop_l = ((Gamma)/(Gamma-1.0)) *
primitive_variable_n[l][2]/(primitive_variable_n[l][0]) +
(primitive_variable_n[l][1]*primitive_variable_n[l][1])/2;
double total_energy_art_dissi_loop_l =  total_enthalpy_art_dissi_loop_l -
(primitive_variable_n[l][2]/primitive_variable_n[l][0]);

double total_enthalpy_art_dissi_loop_l_plus_1 = ((Gamma/(Gamma-1)) *
primitive_variable_n[l+1][2]/primitive_variable_n[l+1][0]) +
(pow(primitive_variable_n[l+1][1],2)/2);
double total_energy_art_dissi_loop_l_plus_1
=  total_enthalpy_art_dissi_loop_l_plus_1 -
(primitive_variable_n[l+1][2]/primitive_variable_n[l+1][0]);

double total_enthalpy_art_dissi_loop_l_minus_1 = ((Gamma/(Gamma-1)) *
primitive_variable_n[l-1][2]/primitive_variable_n[l-1][0]) +
(pow(primitive_variable_n[l-1][1],2)/2);
```

```
double total_energy_art_dissi_loop_l_minus_1
=  total_enthalpy_art_dissi_loop_l_minus_1 - (primitive_variable_n[l-
1][2]/primitive_variable_n[l-1][0]);

double total_enthalpy_art_dissi_loop_l_plus_2 = ((Gamma/(Gamma-1)) *
primitive_variable_n[l+2][2]/primitive_variable_n[l+2][0]) +
(pow(primitive_variable_n[l+2][1],2)/2);
double total_energy_art_dissi_loop_l_plus_2
=  total_enthalpy_art_dissi_loop_l_plus_2 -
(primitive_variable_n[l+2][2]/primitive_variable_n[l+2][0]);

 D_1_U_1[l] = (compute_lambda_i_plus_half(l)) *
(compute_Epsilon_2_i_plus_half(l)) * (conserved_variable_n[l+1][0]-
conserved_variable_n[l][0]);
 D_1_U_2[l] = (compute_lambda_i_plus_half(l)) *
(compute_Epsilon_2_i_plus_half(l)) * ((conserved_variable_n[l+1][1]) -
(conserved_variable_n[l][1]));
 D_1_U_3[l] = (compute_lambda_i_plus_half(l)) *
(compute_Epsilon_2_i_plus_half(l)) * ((conserved_variable_n[l+1][2]) -
(conserved_variable_n[l][2]));

 D_3_U_1[l] = (compute_lambda_i_plus_half(l)) *
(compute_Epsilon_4_i_plus_half(l)) * (conserved_variable_n[l+2][0] -
(3.0*conserved_variable_n[l+1][0]) + (3.0*conserved_variable_n[l][0]) -
conserved_variable_n[l-1][0]);
 D_3_U_2[l] = (compute_lambda_i_plus_half(l)) *
(compute_Epsilon_4_i_plus_half(l)) * ((conserved_variable_n[l+2][1])-
(3*conserved_variable_n[l+1][1])+(3*conserved_variable_n[l][1])-
(conserved_variable_n[l-1][1]));
 D_3_U_3[l] = (compute_lambda_i_plus_half(l)) *
(compute_Epsilon_4_i_plus_half(l)) * ((conserved_variable_n[l+2][2])-
(3*conserved_variable_n[l+1][2])+(3*conserved_variable_n[l][2])-
(conserved_variable_n[l-1][2]));


//  D_1_U_1[l] = (compute_lambda_i_plus_half(l)) *
(compute_Epsilon_2_i_plus_half(l)) * (primitive_variable_n[l+1][0]-
primitive_variable_n[l][0]);
//  D_1_U_2[l] = (compute_lambda_i_plus_half(l)) *
(compute_Epsilon_2_i_plus_half(l)) *
((primitive_variable_n[l+1][1]*primitive_variable_n[l+1][0]) -
(primitive_variable_n[l][1]*primitive_variable_n[l][0] ));
//  D_1_U_3[l] = (compute_lambda_i_plus_half(l)) *
(compute_Epsilon_2_i_plus_half(l)) *
((primitive_variable_n[l+1][0]*total_energy_art_dissi_loop_l_plus_1) -
(primitive_variable_n[l][0]*total_energy_art_dissi_loop_l ));
```

```
//  D_3_U_1[l] = (compute_lambda_i_plus_half(l)) *
(compute_Epsilon_4_i_plus_half(l)) * (primitive_variable_n[l+2][0] -
(3.0*primitive_variable_n[l+1][0]) + (3.0*primitive_variable_n[l][0]) -
primitive_variable_n[l-1][0]);
// cout<< compute_lambda_i_plus_half(l)<<endl;
// cout << compute_Epsilon_4_i_plus_half(l)<<endl;
// cout << (conserved_variable_n[l+2][0] - (3.0*conserved_variable_n[l+1][0])+
(3.0*conserved_variable_n[l][0]) - conserved_variable_n[l-1][0]) <<endl;
// cout << (conserved_variable_n[l-1][0]) << endl; // is 0 as the conserved
variable does not have ay value for the 0 cell. use primitive variable here.

//  D_3_U_2[l] = (compute_lambda_i_plus_half(l)) *
(compute_Epsilon_4_i_plus_half(l)) *
((primitive_variable_n[l+2][0]*primitive_variable_n[l+2][1])-
(3*primitive_variable_n[l+1][0]*primitive_variable_n[l+1][1])+(3*primitive_var
iable_n[l][0]*primitive_variable_n[l][1])-(primitive_variable_n[l-
1][0]*primitive_variable_n[l-1][1]));
//  D_3_U_3[l] = (compute_lambda_i_plus_half(l)) *
(compute_Epsilon_4_i_plus_half(l)) *
((primitive_variable_n[l+2][0]*total_energy_art_dissi_loop_l_plus_2)-
(3*primitive_variable_n[l+1][0]*total_energy_art_dissi_loop_l_plus_1)+(3*primi
tive_variable_n[l][0]* total_energy_art_dissi_loop_l)-(primitive_variable_n[l-
1][0]*total_energy_art_dissi_loop_l_minus_1));


 d1_plus_half[l] = -(D_1_U_1[l] - D_3_U_1[l]);
 d2_plus_half[l] = -(D_1_U_2[l] - D_3_U_2[l]);
 d3_plus_half[l] = -(D_1_U_3[l] - D_3_U_3[l]);


 // cout << "D_1_U_1 ="<<D_1_U_1[l]<<endl;
//   cout << "D_1_U_2 = "<<D_1_U_2[l]<<endl;
//   cout << "D_1_U_3 = "<<D_1_U_3[l]<<endl;
  //cout << "D_3_U_1 = "<<D_3_U_1[l]<<endl;
//   cout << "D_3_U_2 = "<<D_3_U_2[l]<<endl;
//   cout << "D_3_U_3 = "<<D_3_U_3[l]<<endl;
 }

// 0 the cell :

   //  d1_plus_half[0] = (2.0 * d1_plus_half[1]) - d1_plus_half[2];
   //  d2_plus_half[0] = (2.0 * d2_plus_half[1]) - d2_plus_half[2];
   //  d3_plus_half[0] = (2.0 * d3_plus_half[1]) - d3_plus_half[2];

 // test for artificial dissipation extrapolation
// d1_plus_half[0] = d1_plus_half[1];
// d2_plus_half[0] = d2_plus_half[1];


// d3_plus_half[0] = d3_plus_half[1];

double total_enthalpy_art_dissi_0 = ((Gamma/(Gamma-1)) *
primitive_variable_n[0][2]/primitive_variable_n[0][0]) +
(pow(primitive_variable_n[0][1],2)/2);
double total_energy_art_dissi_0 =  total_enthalpy_art_dissi_0 -
(primitive_variable_n[0][2]/primitive_variable_n[0][0]);

double total_enthalpy_art_dissi_0_plus_1 = ((Gamma/(Gamma-1)) *
primitive_variable_n[0+1][2]/primitive_variable_n[0+1][0]) +
(pow(primitive_variable_n[0+1][1],2) / 2);
double total_energy_art_dissi_0_plus_1 =  total_enthalpy_art_dissi_0_plus_1 -
(primitive_variable_n[0+1][2]/primitive_variable_n[0+1][0]);

double total_enthalpy_art_dissi_0_plus_2 = ((Gamma/(Gamma-1)) *
primitive_variable_n[0+2][2]/primitive_variable_n[0+2][0]) +
(pow(primitive_variable_n[0+2][1],2)/2);
double total_energy_art_dissi_0_plus_2 =  total_enthalpy_art_dissi_0_plus_2 -
(primitive_variable_n[0+2][2]/primitive_variable_n[0+2][0]);

//  D_1_U_1[0] = (compute_lambda_i_plus_half(0)) *
(compute_Epsilon_2_i_plus_half(0)) * (primitive_variable_n[0+1][0]-
primitive_variable_n[0][0]);
//  D_1_U_2[0] = (compute_lambda_i_plus_half(0)) *
(compute_Epsilon_2_i_plus_half(0)) *
((primitive_variable_n[0+1][0]*primitive_variable_n[0+1][1])  -
(primitive_variable_n[0][0]*primitive_variable_n[0][1]));
//  D_1_U_3[0] = (compute_lambda_i_plus_half(0)) *
(compute_Epsilon_2_i_plus_half(0)) *
((primitive_variable_n[0+1][0]*total_energy_art_dissi_0_plus_1)-
(primitive_variable_n[0][0]*total_energy_art_dissi_0));

//  D_3_U_1[0] = (compute_lambda_i_plus_half(0)) *
(compute_Epsilon_4_i_plus_half(0)) * ((primitive_variable_n[0+2][0])-
(3*primitive_variable_n[0+1][0])+(3*primitive_variable_n[0][0])-
(rho_minus_1));
//  D_3_U_2[0] = (compute_lambda_i_plus_half(0)) *
(compute_Epsilon_4_i_plus_half(0)) *
((primitive_variable_n[0+2][0]*primitive_variable_n[0+2][1])-
(3*primitive_variable_n[0+1][0]*primitive_variable_n[0+1][1])+(3*primitive_var
iable_n[0][0]*primitive_variable_n[0][1])-(rho_minus_1 * u_minus_1));
//  D_3_U_3[0] = (compute_lambda_i_plus_half(0)) *
(compute_Epsilon_4_i_plus_half(0)) *
((primitive_variable_n[0+2][0]*total_energy_art_dissi_0_plus_2)-
(3*primitive_variable_n[0+1][0]*total_energy_art_dissi_0_plus_1)+(3*primitive_
variable_n[0][0]*total_energy_art_dissi_0)-
(rho_minus_1*total_energy_minus_1));

 D_1_U_1[0] = (compute_lambda_i_plus_half(0)) *
(compute_Epsilon_2_i_plus_half(0)) * (conserved_variable_n[0+1][0]-
conserved_variable_n[0][0]);
 D_1_U_2[0] = (compute_lambda_i_plus_half(0)) *
(compute_Epsilon_2_i_plus_half(0)) * ((conserved_variable_n[0+1][1]) -
(conserved_variable_n[0][1]));
 D_1_U_3[0] = (compute_lambda_i_plus_half(0)) *
(compute_Epsilon_2_i_plus_half(0)) * ((conserved_variable_n[0+1][2])-
(conserved_variable_n[0][2]));

 D_3_U_1[0] = (compute_lambda_i_plus_half(0)) *
(compute_Epsilon_4_i_plus_half(0)) * ((conserved_variable_n[0+2][0])-
(3*conserved_variable_n[0+1][0])+(3*conserved_variable_n[0][0])-
(rho_minus_1));
 D_3_U_2[0] = (compute_lambda_i_plus_half(0)) *
(compute_Epsilon_4_i_plus_half(0)) * ((conserved_variable_n[0+2][1])-
(3*conserved_variable_n[0+1][1])+(3*conserved_variable_n[0][1])-(rho_minus_1 *
u_minus_1));
 D_3_U_3[0] = (compute_lambda_i_plus_half(0)) *
(compute_Epsilon_4_i_plus_half(0)) * ((conserved_variable_n[0+2][2])-
(3*conserved_variable_n[0+1][2])+(3*conserved_variable_n[0][2])-
(rho_minus_1*total_energy_minus_1));

 d1_plus_half[0] = -(D_1_U_1[0] - D_3_U_1[0]);
 d2_plus_half[0] = -(D_1_U_2[0] - D_3_U_2[0]);
 d3_plus_half[0] = -(D_1_U_3[0] - D_3_U_3[0]);

 defect_test_artificial_dissipation_0th_interface(n);



// i_max and i_max + 1 cell:

// artificial dissipation for i_max and i_max-1
   // d1_plus_half[i_max] = (2.0*d1_plus_half[i_max-1]) - d1_plus_half[i_max-
2];
   // d2_plus_half[i_max] = (2.0*d2_plus_half[i_max-1]) - d2_plus_half[i_max-
2];
   // d3_plus_half[i_max] = (2.0*d3_plus_half[i_max-1]) - d3_plus_half[i_max-
2];

   // test for artificial dissipation extrapolation
   // d1_plus_half[i_max] = d1_plus_half[i_max-1];
   // d2_plus_half[i_max] = d2_plus_half[i_max-1];


   // d3_plus_half[i_max] = d3_plus_half[i_max-1];


 D_1_U_1[i_max] = (compute_lambda_i_plus_half(i_max)) *
(compute_Epsilon_2_i_plus_half(i_max)) * (conserved_variable_n[i_max+1][0]-
conserved_variable_n[i_max][0]);
 D_1_U_2[i_max] = (compute_lambda_i_plus_half(i_max)) *
(compute_Epsilon_2_i_plus_half(i_max)) * ((conserved_variable_n[i_max+1][1])-
(conserved_variable_n[i_max][1]));
 D_1_U_3[i_max] = (compute_lambda_i_plus_half(i_max)) *
(compute_Epsilon_2_i_plus_half(i_max)) * ((conserved_variable_n[i_max+1][2]) -
(conserved_variable_n[i_max][2]));

 D_3_U_1[i_max] = (compute_lambda_i_plus_half(i_max)) *
(compute_Epsilon_4_i_plus_half(i_max)) * ((rho_plus_2)-
(3*conserved_variable_n[i_max+1][0])+(3*conserved_variable_n[i_max][0])-
(conserved_variable_n[i_max-1][0]));
 D_3_U_2[i_max] = (compute_lambda_i_plus_half(i_max)) *
(compute_Epsilon_4_i_plus_half(i_max)) * ((rho_plus_2*u_plus_2)-
(3*conserved_variable_n[i_max+1][1])+(3*conserved_variable_n[i_max][1])-
(conserved_variable_n[i_max-1][1]));
 D_3_U_3[i_max] = (compute_lambda_i_plus_half(i_max)) *
(compute_Epsilon_4_i_plus_half(i_max)) * ((rho_plus_2*total_energy_plus_2)-
(3*conserved_variable_n[i_max+1][2])+(3*conserved_variable_n[i_max][2])-
(conserved_variable_n[i_max-1][2]));


double total_enthalpy_art_dissi_i_max_minus_1 = ((Gamma/(Gamma-1)) *
primitive_variable_n[i_max-1][2]/primitive_variable_n[i_max-1][0]) +
(pow(primitive_variable_n[i_max-1][1],2)/2);
double total_energy_art_dissi_i_max_minus_1
=  total_enthalpy_art_dissi_i_max_minus_1 - (primitive_variable_n[i_max-
1][2]/primitive_variable_n[i_max-1][0]);

double total_enthalpy_art_dissi_i_max = ((Gamma/(Gamma-1)) *
primitive_variable_n[i_max][2]/primitive_variable_n[i_max][0]) +
(pow(primitive_variable_n[i_max][1],2)/2);
double total_energy_art_dissi_i_max =  total_enthalpy_art_dissi_i_max -
(primitive_variable_n[i_max][2]/primitive_variable_n[i_max][0]);

double total_enthalpy_art_dissi_i_max_plus_1 = ((Gamma/(Gamma-1)) *
primitive_variable_n[i_max+1][2]/primitive_variable_n[i_max+1][0]) +
(pow(primitive_variable_n[i_max+1][1],2) / 2);
```

```
double total_energy_art_dissi_i_max_plus_1
=  total_enthalpy_art_dissi_i_max_plus_1 -
(primitive_variable_n[i_max+1][2]/primitive_variable_n[i_max+1][0]);


//  D_1_U_1[i_max] = (compute_lambda_i_plus_half(i_max)) *
(compute_Epsilon_2_i_plus_half(i_max)) * (primitive_variable_n[i_max+1][0]-
primitive_variable_n[i_max][0]);
//  D_1_U_2[i_max] = (compute_lambda_i_plus_half(i_max)) *
(compute_Epsilon_2_i_plus_half(i_max)) *
((primitive_variable_n[i_max+1][0]*primitive_variable_n[i_max+1][1]) -
(primitive_variable_n[i_max][0]*primitive_variable_n[i_max][1]));
//  D_1_U_3[i_max] = (compute_lambda_i_plus_half(i_max)) *
(compute_Epsilon_2_i_plus_half(i_max)) *
((primitive_variable_n[i_max+1][0]*total_energy_art_dissi_i_max_plus_1)-
(primitive_variable_n[i_max][0]*total_energy_art_dissi_i_max));

//  D_3_U_1[i_max] = (compute_lambda_i_plus_half(i_max)) *
(compute_Epsilon_4_i_plus_half(i_max)) * ((rho_plus_2)-
(3*primitive_variable_n[i_max+1][0])+(3*primitive_variable_n[i_max][0])-
(primitive_variable_n[i_max-1][0]));
//  D_3_U_2[i_max] = (compute_lambda_i_plus_half(i_max)) *
(compute_Epsilon_4_i_plus_half(i_max)) * ((rho_plus_2*u_plus_2)-
(3*primitive_variable_n[i_max+1][0]*primitive_variable_n[i_max+1][1])+(3*primi
tive_variable_n[i_max][0]*primitive_variable_n[i_max][1])-
(primitive_variable_n[i_max-1][0]*primitive_variable_n[i_max-1][1]));
//  D_3_U_3[i_max] = (compute_lambda_i_plus_half(i_max)) *
(compute_Epsilon_4_i_plus_half(i_max)) * ((rho_plus_2*total_energy_plus_2)-
(3*primitive_variable_n[i_max+1][0]*total_energy_art_dissi_i_max_plus_1)+(3*pr
imitive_variable_n[i_max][0]*total_energy_art_dissi_i_max)-
(primitive_variable_n[i_max-1][0]*total_energy_art_dissi_i_max_minus_1));

 d1_plus_half[i_max] = -(D_1_U_1[i_max] - D_3_U_1[i_max]);
 d2_plus_half[i_max] = -(D_1_U_2[i_max] - D_3_U_2[i_max]);
 d3_plus_half[i_max] = -(D_1_U_3[i_max] - D_3_U_3[i_max]);

 defect_test_artificial_dissipation_i_max_interface(n);

artificial_dissipation_output =
{d1_plus_half[k],d2_plus_half[k],d3_plus_half[k],d1_plus_half[k-
1],d2_plus_half[k-1],d3_plus_half[k-1]};

//print_artificial_dissipation(k,n,d1_plus_half[k],d2_plus_half[k],d3_plus_hal
f[k],d1_plus_half[0],d2_plus_half[0],d3_plus_half[0]);


// cout << "Time step = " << k << endl ;
// cout << "artificial dissipation "<< artificial_dissipation_output[0]
<<endl;
```

```
// cout << "artificial dissipation "<< artificial_dissipation_output[1]
<<endl;
// cout << "artificial dissipation "<< artificial_dissipation_output[2]
<<endl;
// cout << "artificial dissipation "<< artificial_dissipation_output[3]
<<endl;
// cout << "artificial dissipation "<< artificial_dissipation_output[4]
<<endl;
// cout << "artificial dissipation "<< artificial_dissipation_output[5]
<<endl;

return artificial_dissipation_output;
}


 void update_values(int n)
 {

 for (int i = 0; i <= i_max+1  ; i++)
 {
  // here the arrays of the primitive variables have no 0 and i_max + 1 values
as the updated arrays dont have these values
  // it is crucial that these arrays get their 0th and i_max cell populated by
the boundary conditions function.
 conserved_variable_n[i][0] = conserved_variable_n_plus_1[i][0];
 conserved_variable_n[i][1] = conserved_variable_n_plus_1[i][1];
 conserved_variable_n[i][2] = conserved_variable_n_plus_1[i][2];

 }

 }



// reminder : make sure to have a face at the throat
double Euler_equation (std::ofstream&outfile)
{
// solve for u, u+a,u-a


//main loop for solving euler eqns :
for (int n=0; n < n_max; n++)
{
   if (n == 0)
   {
   set_initial_condition_primitive_variable(n);

   }
   if (n > 0)
   {
update_domain_print_variables(n);
   }
set_initial_boundary_conditions_insetropic(n); // add to primitive
set_extra_ghost_cells(n);
// Now we should have the values for domain + boundaries + extra ghost cells.
print_flow_variables(outfile,n);


 for (int i = 0; i <= i_max; i++) // if i = 0 flux is calculated between the
face of 0 and 1.
  {

   F1_plus_half[i]  = ((primitive_variable_n[i][0]*primitive_variable_n[i][1])
+ (primitive_variable_n[i+1][0]*primitive_variable_n[i+1][1])) / 2;

   F2_plus_half[i]  = (((primitive_variable_n[i][0]
*pow(primitive_variable_n[i][1],2)) + primitive_variable_n[i][2]) +
((primitive_variable_n[i+1][0] *pow(primitive_variable_n[i+1][1],2)) +
primitive_variable_n[i+1][2]))/2 ;

   F3_plus_half[i]  = ((((Gamma/(Gamma-
1))*primitive_variable_n[i][2]*primitive_variable_n[i][1]) +
(primitive_variable_n[i][0]*pow(primitive_variable_n[i][1],3)/2)) +
(((Gamma/(Gamma-1))*primitive_variable_n[i+1][2]*primitive_variable_n[i+1][1])
+ (primitive_variable_n[i+1][0]*pow(primitive_variable_n[i+1][1],3)/2)))/2 ;
   }

primitive_to_conserved_variable(n);
for (int i = 1; i <= i_max; i++)
{



artificial_dissipation(i,n);

Source[i] = primitive_variable_n[i][2] * ( 0.4 * M_PI *
cos(M_PI*(x_location(i)-0.5)));
Vol[i] = find_area(i) * d_x;

//checking values :
// status of values =  Values are increasig as we march in time but the values
for the governing equations remain the same.
// Verfication step:  Hand calculate the values of each variable at a few
points and check these values at the start.
//cout<< " value for i = " << i<<endl;
// cout<< " F1_plus_half= " << F1_plus_half[i]<<"  F2_plus_half= " <<
F2_plus_half[i]<< "  F3_plus_half= " << F3_plus_half[i]<<   endl;
// cout << " total_energy_n[i]= "<< total_energy_n[i]<<endl;
// cout << " Source[i]= " << Source[i]<<endl;
// cout << " Vol[i]= " << Vol[i]<<endl;

// U_vector = [rho , rho*u , rho*total_energy]T

if (n == 0)

{
   //cout << n << endl;
   defect_test_initial_values(n);
}


// calculating residual :

residual_eq1[i] =
((F1_plus_half[i]+artificial_dissipation_output[0])*find_area(i+0.5)) -
((F1_plus_half[i-1]+artificial_dissipation_output[3])*find_area(i-0.5)) ;
residual_eq2[i] =
((F2_plus_half[i]+artificial_dissipation_output[1])*find_area(i+0.5)) -
((F2_plus_half[i-1]+artificial_dissipation_output[4])*find_area(i-0.5)) -
(Source[i]*d_x);
residual_eq3[i] =
((F3_plus_half[i]+artificial_dissipation_output[2])*find_area(i+0.5)) -
((F3_plus_half[i-1]+artificial_dissipation_output[5])*find_area(i-0.5)) ;

// defect_test_first_iteration(i,n);
// defect_test_first_iteration(int i,int n)

// {
//    if (n==0)
//    {
//    if((residual_eq1[i] -  >= pow(10,-10))\
//    (residual_eq2[i] -  >= pow(10,-10))\
//    (residual_eq3[i] -  >= pow(10,-10)))
// {
//    cout<< " The residuals for first iteration are wrong "<< endl;
// }

//    }
// }


// cout << " residual eq1 =" << residual_eq1[i] <<endl;
// cout << " residual eq2 =" << residual_eq2[i] << endl;
// cout << " residual eq3 =" << residual_eq3[i] << endl;
// eq 1 : solves U1 :
conserved_variable_n_plus_1[i][0]  =  conserved_variable_n[i][0] -
(residual_eq1[i] * stability(d_x,i)/Vol[i]) ;
```

```
//cout<< stability(d_x,i)<< endl;
// eq 1 : solves U2 :
conserved_variable_n_plus_1[i][1]  =  conserved_variable_n[i][1] -
(residual_eq2[i] * stability(d_x,i)/Vol[i]) ;
// eq 1 : solves U3 :
conserved_variable_n_plus_1[i][2]  =  conserved_variable_n[i][2] -
(residual_eq3[i] * stability(d_x,i)/Vol[i]) ;
// apply limited when converting to the primitive variable!


// //eq 1 : solves rho_n_plus_1

// rho_n_plus_1[i] = rho_n[i] + (stability(d_x,i)/Vol[i])*( -
((F1_plus_half[i]+artificial_dissipation_output[0])*find_area(i+0.5)) +
((F1_plus_half[i-1]+artificial_dissipation_output[3])*find_area(i-0.5)));

// // eq 2 : solves u_n_plus_1

// u_n_plus_1[i] = ((rho_n[i]*u_n[i])+ ((stability(d_x,i)/Vol[i]) *
((Source[i]*d_x) -
((F2_plus_half[i]+artificial_dissipation_output[1])*find_area(i+0.5)) +
((F2_plus_half[i-1]+artificial_dissipation_output[4])*find_area(i-
0.5)))))/rho_n_plus_1[i];

// // eq 3 : solves p_n_plus_1

// total_energy_n_plus_1[i] = ((rho_n[i]*total_energy_n[i]) +
((stability(d_x,i)/Vol[i])* ( -
((F3_plus_half[i]+artificial_dissipation_output[2])*find_area(i+0.5)) +
((F3_plus_half[i-1]+artificial_dissipation_output[5])*find_area(i-
0.5)))))/rho_n_plus_1[i];




//calculating othher physical properties using the updated values from the
main 3 equtions :

// (u_n_plus_1^2 - Gamma*u_n_plus_1^2 +
2*Gamma*R_univ*T_stag)/(2*Gamma*R_univ)

// T_n_plus_1[i] = (pow(u_n_plus_1[i],2) - (Gamma*pow(u_n_plus_1[i],2))  + (2*
Gamma*R_univ*T_stag)) /(2*Gamma*R_univ);
// M_n_plus_1[i] = u_n_plus_1[i]/sqrt(Gamma*R_univ*T_n_plus_1[i]);
// psi_n_plus_1[i] = 1 + (((Gamma-1)/2)*M_n_plus_1[i]);
// p_n_plus_1[i] = p_stag/(pow(psi_n_plus_1[i],(Gamma/(Gamma-1))));


   //cout << " Values at i = " << i <<endl;
   //   cout << " Values for M_n_plus_1 = " << M_n_plus_1[i]<<endl;
   //  cout << " Values for u_n_plus_1 = " << u_n_plus_1[i]<<endl;
   //   cout << " Values for p_n_plus_1= " << p_n_plus_1[i]<<endl;
   //   cout << " Values for rho_n_plus_1 = "<< rho_n_plus_1[i]<<endl;
    //cout << " Values of total_energy_n_plus_1 = " <<
total_energy_n_plus_1[i]<<endl;

}
 if (n % 1 == 0)
 {

 compute_norms(n);
 }

 // updating values :
 update_values(n);

conserved_to_primitive_variable_to_print_variable(n);

}


return 0;
}

int main()
{
double example [5+2] = {0,1,2,3,4,5,6};
cout << "example[4]" << example[4]<<endl;
double* max_element_example = max_element(example, example+(6));
cout << "example[i_max] = "<< example[5]<<endl;

std::ofstream outfile("flow_variables.dat");
outfile << "'TITLE = updated flow variables'" << endl;
string variables = " variables = 'x_location' 'rho_n' 'u_n' 'p_n'
'total_energy' 'M_n' 'T_n' ";
outfile << variables  << std::endl;

Euler_equation(outfile);

outfile.close();




return 0;
}
```