Euler 1D

1.0

Generated by Doxygen 1.8.6

Wed Jan 27 2016 12:02:39

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Namespace Documentation

3.1 Eu1D Namespace Reference

Namespace to hold all necesary constants.

Variables

- const double **gam** = 5.0/3.0

3.1.1 Detailed Description

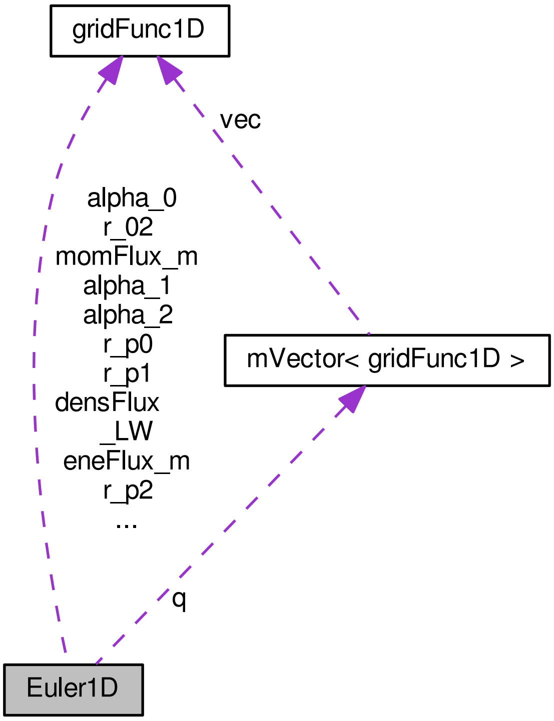
Namespace to hold all necesary constants.

Chapter 4

Class Documentation

4.1 Euler1D Class Reference

Collaboration diagram for Euler1D:



Public Member Functions

- Euler1D (const parameterReader &p)

  q is a mVector that hold the conserver variables q[0] is the density q[1] is momentum q[2] is energy
- const gridFunc1D & **density** () const

- void initialData (gridFunc1D &x, const parameterReader &p)

  Four types of shock tubes as initial data.
- void advanceStep (double dt, double dx)

  Evolve in time from t to t+dt.
- void **calcFlux** (double dt, double dx)
- void **calcLWFlux** (double dt, double dx)
- void **output** (gridFunc1D &x, double t)
- void **calc_vel** ()
- void **calc_pres** ()

Private Attributes

- mVector< gridFunc1D > **q**
- gridFunc1D **vel**
- gridFunc1D **pres**
- gridFunc1D **densFlux_p**
- gridFunc1D **densFlux_m**
- gridFunc1D **momFlux_p**
- gridFunc1D **momFlux_m**
- gridFunc1D **eneFlux_p**
- gridFunc1D **eneFlux_m**
- gridFunc1D **densFlux_LW**
- gridFunc1D **momFlux_LW**
- gridFunc1D **eneFlux_LW**
- gridFunc1D **lambda_0**
- gridFunc1D **lambda_1**
- gridFunc1D **lambda_2**
- gridFunc1D **r_m0**
- gridFunc1D **r_m1**
- gridFunc1D **r_m2**
- gridFunc1D **r_00**
- gridFunc1D **r_01**
- gridFunc1D **r_02**
- gridFunc1D **r_p0**
- gridFunc1D **r_p1**
- gridFunc1D **r_p2**
- gridFunc1D **alpha_0**
- gridFunc1D **alpha_1**
- gridFunc1D **alpha_2**
- int **convFactor**

4.1.1 Detailed Description

Definition at line 7 of file Euler1D.hpp.

4.1.2 Member Function Documentation

4.1.2.1 void Euler1D::advanceStep (double *dt,* double *dx*)

Evolve in time from t to t+dt.

This is the most simple thing we can do. It is a first order method

Definition at line 139 of file Euler1D.cpp.

```
{
  int N = q[0].Npoints();
  //gridFunc1D fluxDens(N), fluxMom(N), fluxEner(N);
  mVector<gridFunc1D> flux(3);
  flux[0].create(N);
  flux[1].create(N);
  flux[2].create(N);

  // compute fluxes
  calcFlux( dt, dx );

  // compute Lax-Wendroff flux term
  //calcLWFlux( dt, dx );

  // compute net fluxes
  for( int i=2; i<=N-1; i++ ){
    flux[0][i] = densFlux_p[i] + densFlux_m[i+1] + densFlux_LW[i+1] - densFlux_LW[i-1];
    flux[1][i] = momFlux_p[i] + momFlux_m[i+1] + momFlux_LW[i+1] - momFlux_LW[i-1];
    flux[2][i] = eneFlux_p[i] + eneFlux_m[i+1] + eneFlux_LW[i+1] - eneFlux_LW[i-1];
  }

  // advance in time
  //density = density - dt/dx * fluxDens;
  //momentum = momentum - dt/dx * fluxMom;
  //energy = energy - dt/dx * fluxEner;
  q = q - dt/dx * flux;

  // constant extrapolation at boundaries
  //density[1] = density[2];
  //momentum[1] = momentum[2];
  //energy[1] = energy[2];
  q[0][1] = q[0][2];
  q[1][1] = q[1][2];
  q[2][1] = q[2][2];

  //density[N] = density[N-1];
  //momentum[N] = momentum[N-1];
  //energy[N] = energy[N-1];
  q[0][N] = q[0][N-1];
  q[1][N] = q[1][N-1];
  q[2][N] = q[2][N-1];

}
```

The documentation for this class was generated from the following files:

- Euler1D.hpp
- Euler1D.cpp

4.2 gridFunc1D Class Reference

Public Member Functions

- gridFunc1D ()

  Constructor with no arguments.
- gridFunc1D (int)

  Initializes and creates space to hold n elements.
- gridFunc1D (const gridFunc1D &)

  Copy constructor.
- void create (int)

  Creates and resize the objects.
- void **erase** ()
- double & **operator[]** (float) const
- double & **operator[]** (float)
- int Npoints ()

  Get the number of points.
- void **setBoundaryCondition** (int)
- void **setIsFlux** (int)

- void outputGnuplotFake (ofstream &, gridFunc1D &, const double)

  Gnuplot style output.
- void outputByLine (ofstream &, const double t)

  Output all the values of the variable in a single line.
- void outputByColumn (ofstream &, gridFunc1D &, const double t) const

  ygraph output style
- gridFunc1D **operator+** (const gridFunc1D &B) const
- gridFunc1D **operator-** (const gridFunc1D &B) const
- gridFunc1D **operator*** (const gridFunc1D &B) const
- gridFunc1D **operator*** (const double &b) const
- gridFunc1D **operator/** (const double &b) const
- gridFunc1D **operator/** (const gridFunc1D &B) const
- const gridFunc1D & **operator=** (const gridFunc1D &B)

Private Attributes

- int **n_points**
- double * **data**
- double * **datamid**
- int **boundaryType**
- int **isFlux**

Friends

- gridFunc1D **operator*** (const double &a, const gridFunc1D &B)

4.2.1 Detailed Description

Definition at line 13 of file gridFunc1D.hpp.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 gridFunc1D::gridFunc1D ( )

Constructor with no arguments.

It initializes everything to zero

Definition at line 7 of file gridFunc1D.cpp.

```
{
  n_points = 0;
  data = NULL;
  datamid = NULL;
  boundaryType = -1;
  isFlux = 0;
}
```

4.2.3 Member Function Documentation

4.2.3.1 void gridFunc1D::create (int *n*)

Creates and resize the objects.

This function assigns space and also can resize an already existing object

Definition at line 54 of file gridFunc1D.cpp.

```
55 {
56   erase();
57
58   if ( n > 0 ){
59     n_points = n;
60     data = new double[n];
61     datamid = new double[n+1];
62
63     for( int i=0; i<n; i++ )   data[i] = 0.0;
64     for( int i=0; i<n+1; i++ ) datamid[i] = 0.0;
65   }
66   else {
67     cout << "ERROR: the number of points must be positive."<<endl;
68     exit(1);
69   }
70 }
```

4.2.3.2 void gridFunc1D::outputByColumn (ofstream & *out,* gridFunc1D & *x,* const double *t*) const

ygraph output style

ygraph output style consists in starting a block with the current time followed with the position and the value of the variable in one line

Definition at line 149 of file gridFunc1D.cpp.

```
150 {
151   out << "#time = " << t << endl;
152
153   for( int i=1; i<=n_points; i++ )
154     out << x[i] << "\t" << (*this)[i] << endl;
155
156   out << endl;
157 }
```

4.2.3.3 void gridFunc1D::outputByLine (ofstream & *out,* const double *t*)

Output all the values of the variable in a single line.

Time is in the first column followed by all the pointwise values of the variable. The position is not output. This is useful the integrate the velocity to get the position of hypothetical particles.

Definition at line 133 of file gridFunc1D.cpp.

```
134 {
135   out << t;
136
137   for( int i=1; i<=n_points; i++ )
138     out << "\t" << (*this)[i];
139
140   out << endl;
141 }
```

4.2.3.4 void gridFunc1D::outputGnuplotFake (ofstream & *out,* gridFunc1D & *x,* const double *t*)

Gnuplot style output.

This function is trick to make a 1D variable look like a 2D one. Its purpose is the make a density plot with Gnuplot

Definition at line 111 of file gridFunc1D.cpp.

```
112 {
113   out << "#time = " << t << endl;
114
115   for( int i=1; i<=n_points; i++ )
116     out << 0.0 << "\t" << x[i] << "\t" << (*this)[i] << endl;
117
118   out << endl;
119
```

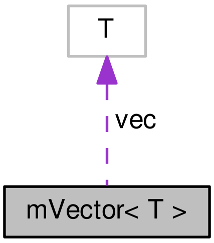
```
  for( int i=1; i<=n_points; i++ )
    out << 1.0 << "\t" << x[i] << "\t" << (*this)[i] << endl;

  out << endl << endl;
}
```

The documentation for this class was generated from the following files:

- gridFunc1D.hpp
- gridFunc1D.cpp

4.3 mVector< T > Class Template Reference

Collaboration diagram for mVector< T >:



Public Member Functions

- mVector ()

  Constructor.
- **mVector** (int)
- void **resize** (int)
- mVector (const mVector< T > &)

  Copy constructor.
- void **erase** ()
- void set (int, T)

  Set the value of each components.
- int **getDim** () const
- T & operator[] (int i) const

  Overload operator[] to get the values of each component.
- T & operator[] (int i)

  Overload operator[] to set the values of each component.
- const mVector< T > & **operator=** (const mVector< T > &)
- mVector< T > **operator+** (const mVector< T > &) const
- mVector< T > **operator-** (const mVector< T > &) const
- mVector< T > **operator*** (const double a) const

Private Attributes

- int **dim**
- T ∗ **vec**

4.3.1 Detailed Description

template<class T>class mVector< T >

Definition at line 13 of file mvector.hpp.

The documentation for this class was generated from the following files:

- mvector.hpp
- mvector.cpp

4.4 parameterReader Class Reference

Public Member Functions

- parameterReader (std::string)

  Constructor.
- const char ∗ getParam (std::string s) const

  Returns the value of the parameter name given.

Private Attributes

- int len

  the length of the parameter table
- std::string paramTable [1000][2]

  a len by 2 matrix holding the parameter name and the parameter value

4.4.1 Detailed Description

Definition at line 13 of file parameterReader.hpp.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 parameterReader::parameterReader (std::string *fileName*)

Constructor.

In the constructor the Parameters file is read. The pairs "parameter name" and "parameter value" are stored in a matrix.

Definition at line 9 of file parameterReader.cpp.

```
{
  std::ifstream infile( fileName.c_str() );
  std::string line, paramName, paramValue;
  std::size_t pos;
  int c = 0;

  while( std::getline( infile, line ) ){
    // get position of the = sign
    pos = line.find( "=" );
    // get the string before and after the = sign
    paramName  = line.substr( 0, pos-1 );
    paramValue = line.substr( pos+1 );
    // remove space infront of parameter value
    paramValue.erase(remove_if(paramValue.begin(), paramValue.end(), isspace),paramValue.end());
    // assing to paramTable
    paramTable[c][0] = paramName;
    paramTable[c][1] = paramValue;

    //cout << paramTable[c][0] << "qqq" << paramTable[c][1] << "q"<<endl;
    c++;
  }

  len = c;
  infile.close();
}
```

4.4.3 Member Function Documentation

4.4.3.1 const char ∗ parameterReader::getParam (std::string *s*) const

Returns the value of the parameter name given.

The function takes the parameter name s and returns a the value as a const char∗ If the value is numeric, it has to be converted to int or double.

Definition at line 41 of file parameterReader.cpp.

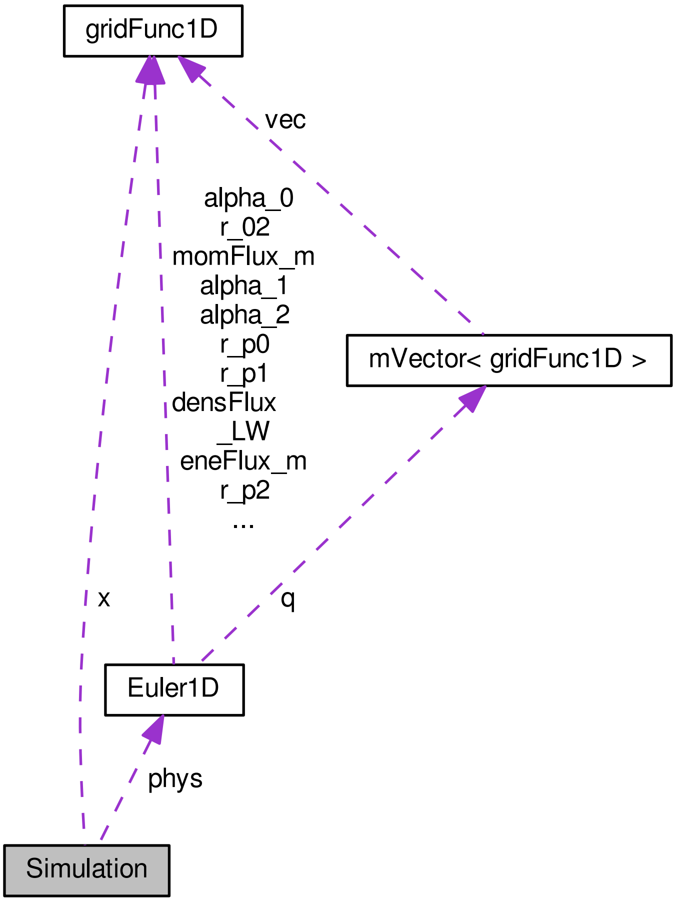
```
{
  int i=0;

  while( paramTable[i][0] != s ){
    i++;
    if ( i>len ){
      std::cerr << "ERROR: Parameter "<<s<<" was not found in the parameter file"<<endl;
      exit(1);
    }
  }

  return paramTable[i][1].c_str();
}
```

The documentation for this class was generated from the following files:

- parameterReader.hpp
- parameterReader.cpp

4.5 Simulation Class Reference

Collaboration diagram for Simulation:



Public Member Functions

- **Simulation** (const parameterReader &p)
- void **initialData** (const parameterReader &p)
- void **evolve** ()

Protected Attributes

- const double fudge

  A small quantity.
- int n_points_x

  Number of points in the x direction.
- double xMin

  Minimum value of x coordinate.
- double xMax

  Maximum value of x coordinate.
- double CFL

  Courant-Friederich-Levy factor.
- double finalTime

  Final time, where simulation stops.
- double outEveryTime

  How often in time we do output.
- double time

  The time coordinate (variable)
- double dx

  Separation between points in x.
- double dt

  Separation between points in time.
- int outEvery

  How many iterations we do output.
- int ITMAX

  Maximum nunmber of iterations, when simulation stops.
- gridFunc1D x

  x coordinate
- Euler1D ∗ phys

  The physical system of equations.

4.5.1 Detailed Description

Definition at line 10 of file simulation.cpp.

The documentation for this class was generated from the following file:

- simulation.cpp

Index