Applications of Mathematics

Pavel Šolín; Karel Segeth

Description of the multi-dimensional finite volume solver EULER

Applications of Mathematics, Vol. 47 (2002), No. 2, 169–185

Persistent URL: http://dml.cz/dmlcz/134493

47 (2002) APPLICATIONS OF MATHEMATICS No. 2, 169–185

DESCRIPTION OF THE MULTI-DIMENSIONAL FINITE VOLUME SOLVER EULER*

Pavel Šolín, Linz, Karel Segeth, Prague

Abstract. This paper is aimed at the description of the multi-dimensional finite volume solver EULER, which has been developed for the numerical solution of the compressible Euler equations during several last years. The present overview of numerical schemes and the explanation of numerical techniques and tricks which have been used for EULER could be of certain interest not only for registered users but also for numerical mathematicians who have decided to implement a finite volume solver themselves. This solver has been used also for the computation of numerical examples presented in other papers of the authors.

Keywords: numerical software, computational fluid dynamics, compressible Euler equations, method of lines, approximate Riemann solver, numerical flux, 1D, quasi-1D, 2D, axisymmetric 3D, 3D, finite volume method, unstructured grid, implicit adaptive time integration

MSC 2000: 65M20, 35L65, 35L67, 76N10, 76N15, 76R10

1. Introduction

The numerical software package EULER is written in C++ and is free accessible in the Internet** for non-commercial purposes. The software is currently used by approximately one hundred registered users from various institutes in various countries. The solver is based mainly on theoretical results published in [3], [4], [5], [6], [7] but as the reader knows, there is, in general, a long way from the theory to the implementation.

The presented paper is not intended to provide a basic knowledge about the compressible Euler equations and the finite volume method. Readers who are not familiar

* This work was partly supported by Cooperation Research Project ME 148 (Czech Republic).

** `http://www.numa.uni-linz.ac.at/Staff/solin/euler` (October 1, 2000).

with the application of the finite volume method to the compressible Euler equations are recommended to read at least the article [4] before starting with this one.

Let us devote two short sections to the compressible Euler equations and to the finite volume method before we start dealing with the programming aspects.

2. Compressible Euler equations

Let the domain $\Omega \subset \mathbb{R}^d$ occupied by the fluid be bounded and have a continuous, piecewise linear boundary $\partial\Omega$. The d-dimensional compressible Euler equations consist of the continuity equation, d Euler momentum equations and the energy equation. We will consider them in the space-time cylinder $Q_T = \Omega \times (0,T)$ $(T > 0)$ and write them in the form

$$\frac{\partial \boldsymbol{w}}{\partial t} + \sum_{s=1}^{d} \frac{\partial \boldsymbol{f}_s(\boldsymbol{w})}{\partial x_s} = 0, \tag{1}$$

where

$$\boldsymbol{w} = \begin{pmatrix} \varrho \\ \varrho v_1 \\ \vdots \\ \varrho v_d \\ e \end{pmatrix}, \quad \boldsymbol{f}_s(\boldsymbol{w}) = \begin{pmatrix} \varrho v_s \\ \varrho v_s v_1 + \delta_{s1} p \\ \vdots \\ \varrho v_s v_d + \delta_{sd} p \\ (e+p) v_s \end{pmatrix}, \quad s = 1, \dots, d, \tag{2}$$

and

$$e = \frac{p}{\kappa - 1} + \frac{1}{2} \varrho |\boldsymbol{v}|^2. \tag{3}$$

We use the standard notation: t—time, $x_1, \dots, x_d$—Cartesian coordinates in $\mathbb{R}^d$, $\boldsymbol{x} = (x_1, \dots, x_d)^T$, ϱ—density, $\boldsymbol{v} = (v_1, \dots, v_d)^T$—velocity vector with components $v_1, \dots, v_d$ in the directions $x_1, \dots, x_d$, p—pressure, e—total energy, δ_{sj}—Kronecker delta, $\kappa > 1$—Poisson adiabatic constant. Due to physical reasons it is also suitable to require $\varrho > 0$, $p > 0$. The functions $\boldsymbol{f}_s$, $s = 1, \dots, d$, are called inviscid (Euler) fluxes and are defined in the set

$$\mathcal{Q} = \left\{ (w_1, \dots, w_{d+2})^T \in \mathbb{R}^{d+2};\ w_1 > 0,\ w_{d+2} - \frac{w_2^2 + \dots + w_{d+1}^2}{2w_1} > 0 \right\}. \tag{4}$$

System (1), (3) is equipped with the initial condition

$$\boldsymbol{w}(\boldsymbol{x}, 0) = \boldsymbol{w}^0(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \tag{5}$$

and a set of boundary conditions which will be specified later.

3. Application of the method of lines

Discretization of the above partial differential equations is done in two steps according to the general framework of the method of lines (MOL). In the first step, semi-discretization in space is performed using the finite volume method (FVM). There are several reasons why FVM is popular for the compressible Euler equations (see e.g. [3]). Discretizing only the space variable while keeping the time variable continuous, one obtains an initial value problem for a system of non-linear ordinary differential equations (ODE). In the second step, this ODE system is integrated in time by suitable numerical methods for ODE's.

More precisely, our aim is to find coefficients $\boldsymbol{y} \in C^1(0,T;\mathcal{Q}^N)$ in the FMV setting to the approximate solution

$$\boldsymbol{w}_h(\boldsymbol{x}t) = \sum_{i=1}^{N} \boldsymbol{y}_i(t)\chi_i(\boldsymbol{x}) \tag{6}$$

of the compressible Euler equations satisfying

$$\dot{\boldsymbol{y}}(t) = \boldsymbol{F}(\boldsymbol{y}(t),t) \text{ for all } t \in (0,T), \tag{7}$$

$$\boldsymbol{y}(0) = \boldsymbol{y}^0. \tag{8}$$

The symbol $\chi_i(\boldsymbol{x})$, $1 \leqslant i \leqslant N$, denotes characteristic functions of finite volumes Ω_i. For the description of the finite volume grid, the exact definition of the non-linear right-hand side (RHS) vector function $\boldsymbol{F}(\boldsymbol{y}(t),t)$ and for the construction of the initial condition $\boldsymbol{y}^0$ from the initial data (5) to the continuous problem see [4].

4. Numerical flux

Numerical flux is the basic feature of the FVM since the method is based on the approximation of the flux of physical quantities (density, momentum, energy etc.) through the shared faces of the finite volumes. The solver EULER is intended for the computation of perfect gas flows past solid obstacles (airfoils etc.) and within various channels (e.g. turbines, wind tunnels, diffusors, jets). Therefore, the boundary of domains considered is assumed to consist only of impermeable solid walls and of inlet and outlet parts. Thus, finite volumes can have either *solid wall* faces, *inlet-outlet* faces or *interior* faces (faces shared by two adjacent finite volumes). For infinite domains with periodic geometries we additionally define *periodic* faces which are a subclass of the interior ones. Numerical flux has a special form for each one of these three classes of faces. The treatment of solid wall faces is relatively simple (see [4]).

Numerical flux through inlet-outlet faces (in fact representing the treatment of inlet-outlet boundary conditions to the compressible Euler equations) is not trivial and will be addressed later. Let us first discuss several aspects concerning the interior numerical flux.

4.1. Interior numerical flux.

The numerical flux H_{int} through interior faces has the form

$$H_{\mathrm{int}}(\boldsymbol{w}_i, \boldsymbol{w}_j, \nu_{ij}, |\partial\Omega_{ij}|) = |\partial\Omega_{ij}| T_d^{-1}(\nu_{ij}) \boldsymbol{f}_R\big(T_d(\nu_{ij})\boldsymbol{w}_i, T_d(\nu_{ij})\boldsymbol{w}_j\big), \tag{9}$$

where $\boldsymbol{w}_i$, $\boldsymbol{w}_j$ are $(d+2)$-dimensional state vectors corresponding to the adjacent finite volumes Ω_i, Ω_j and ν_{ij} is a unit outer normal to the shared face $\partial\Omega_{ij}$ of the finite volumes Ω_i, Ω_j pointing from Ω_i to Ω_j. For the exact form of the rotational matrices $T_d(\nu_{ij})$, $T_d^{-1}(\nu_{ij})$, see e.g. [4]. The non-linear vector-valued function $\boldsymbol{f}_R$ is called the *approximate Riemann solver* (ARS). It approximates the constant value $\boldsymbol{f}_1(\boldsymbol{q}_r(0,\tilde{t}))$, $\boldsymbol{q}_r(\tilde{x}_1,\tilde{t})$ being the exact solution to the Riemann problem

$$\frac{\partial \boldsymbol{q}}{\partial \tilde{t}}(\tilde{x}_1,\tilde{t}) + \frac{\partial \boldsymbol{f}_1(\boldsymbol{q})}{\partial \tilde{x}_1}(\tilde{x}_1,\tilde{t}) = 0 \quad \text{in } \mathbb{R}\times(0,+\infty), \tag{10}$$

$$\boldsymbol{q}(\tilde{x}_1,0) = \begin{cases} \boldsymbol{q}_L, & \tilde{x}_1 < 0, \\ \boldsymbol{q}_R, & \tilde{x}_1 > 0, \end{cases} \tag{11}$$

arising in the normal direction on the interface of two adjacent finite volumes. The solution of this 1D Riemann problem is described e.g. in [6].

4.2. ARS's implemented in EULER.

EULER is based on the *Godunov type* schemes which are splitting the flux into its positive and negative parts using the spectral decomposition of the Jacobi matrix

$$\mathbf{A}(\boldsymbol{w}) = \frac{\mathrm{D}\boldsymbol{f}_1}{\mathrm{D}\boldsymbol{w}} \tag{12}$$

of the first Euler flux. The Godunov flux splitting method results in the proposition of an ARS of the general form

$$\boldsymbol{f}_R(\boldsymbol{q}_L,\boldsymbol{q}_R) = \boldsymbol{f}(\boldsymbol{q}_R) - \int_{\boldsymbol{q}_L}^{\boldsymbol{q}_R} \mathbf{A}^+(\boldsymbol{q})\,\mathrm{d}\boldsymbol{q} = \boldsymbol{f}(\boldsymbol{q}_L) + \int_{\boldsymbol{q}_L}^{\boldsymbol{q}_R} \mathbf{A}^-(\boldsymbol{q})\,\mathrm{d}\boldsymbol{q}. \tag{13}$$

Integrals of the matrices $\mathbf{A}^+$ and $\mathbf{A}^-$ (the positive and negative parts of the matrix $\mathbf{A}$: for the exact definition, see e.g. [6]) are not well defined because they depend on the path in the state space. Nevertheless, there are some heuristic approximations of

these integrals leading to successfully working ARS's. The first three ARS's which are implemented in EULER are based on heuristic "numerical quadratures" of the integrals involved in (13):

- ARS by Steger-Warming

$$\boldsymbol{f}_R(\boldsymbol{q}_L, \boldsymbol{q}_R) = \mathbf{A}^+(\boldsymbol{q}_L)\boldsymbol{q}_L + \mathbf{A}^-(\boldsymbol{q}_R)\boldsymbol{q}_R, \tag{14}$$

- ARS by Vijayasundaram

$$\boldsymbol{f}_R(\boldsymbol{q}_L, \boldsymbol{q}_R) = \mathbf{A}^+\Big(\frac{\boldsymbol{q}_L + \boldsymbol{q}_R}{2}\Big)\boldsymbol{q}_L + \mathbf{A}^-\Big(\frac{\boldsymbol{q}_L + \boldsymbol{q}_R}{2}\Big)\boldsymbol{q}_R, \tag{15}$$

- ARS by Van Leer

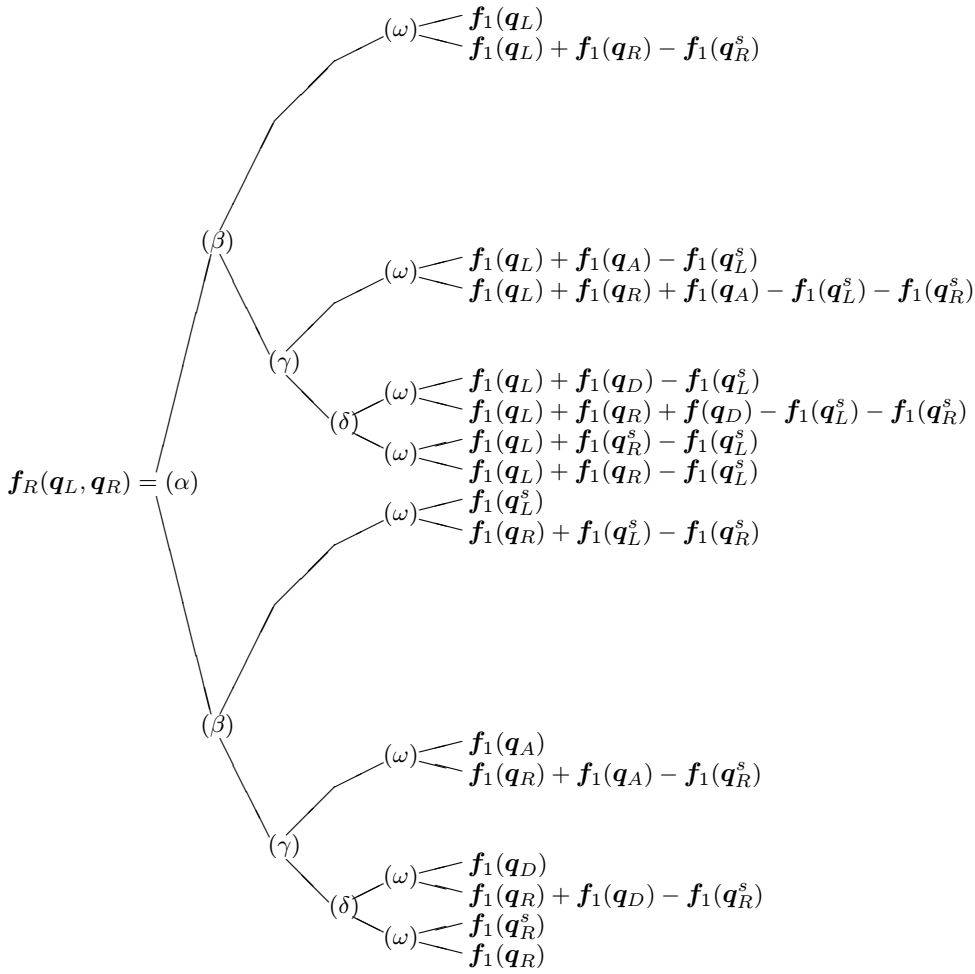
$$\boldsymbol{f}_R(\boldsymbol{q}_L, \boldsymbol{q}_R) = \frac{\boldsymbol{f}_1(\boldsymbol{q}_L) + \boldsymbol{f}_1(\mathbf{q}_R)}{2} - |\mathbf{A}|\Big(\frac{\boldsymbol{q}_L + \boldsymbol{q}_R}{2}\Big)\Big(\frac{\boldsymbol{q}_R - \boldsymbol{q}_L}{2}\Big). \tag{16}$$

The fourth implemented ARS (by Osher-Solomon) is based on the exact computation of the integrals along a sophisticated path in the state space. This scheme is depicted in the diagram on the following page.

The reason why we present the diagram corresponding to the ARS by Osher-Solomon is to show where the remarkable advantage of this scheme in CPU efficiency in comparison with the above ARS's comes from. While the first three schemes need to evaluate the matrices $\mathbf{A}^+, \mathbf{A}^-$ or $|\mathbf{A}|$ and multiply them by the state vectors, the ARS by Osher-Solomon needs only to do a few binary decisions (indicated by α, β, γ, δ, ω in the diagram) and to evaluate a few simple algebraic expressions. The complete scheme can be found either in [5] or directly in the C++ source code. ARS must be computed on each grid face and at each time step—in 3D, one hundred thousand grid faces and fifty thousand time steps still correspond to a relatively small task. It is easy to see that the efficiency of the ARS has crucial influence on the CPU requirements of the computation. In addition, the Osher-Solomon ARS has the best accuracy and robustness among the schemes mentioned.

The reader may ask why we did not remove the first three ARS's when the Osher-Solomon scheme is so much better. The reason is that EULER is not a commercial code and one of its main purposes is to serve as a tool helpful for learning, testing and understanding various aspects of the numerics. One can simply choose an ARS in the configuration file before starting the computation.

The ARS by Steger-Warming is a fully upwind scheme (the positive part of the matrix $\mathbf{A}$ is computed at $\boldsymbol{q}_L$ and its negative part at $\boldsymbol{q}_R$). This brings very good stability but too much diffusion to the scheme. The result is a remarkable loss of



accuracy in comparison with other schemes. The lack of accuracy can be observed mainly along shock waves which in fact are not captured and do not seem to represent a discontinuity in the solution. The scheme by Vijayasundaram involves less diffusion because both the positive and the negative part of the matrix $\mathbf{A}$ are computed exactly at the average of the states $\boldsymbol{q}_L$ and $\boldsymbol{q}_R$. This scheme has less diffusion and is more accurate. It is capable to capture a position and size of shocks very well but sometimes produces oscillations. The ARS by Van Leer is similarly accurate as the Vijayasundaram scheme but has better stability properties. We recommend the reader to compare all the four schemes on some examples included in the EULER package to get a feeling for the relation between the diffusion and robustness of upwind schemes.

4.3. Multi-dimensionality of ARS.

There is an interesting feature of the ARS which enables EULER to compute 1D, 2D and 3D problems using only one code. Let us demonstrate it on an example of ARS based on a heuristic numerical quadrature (where it can be understood easier). The Jacobi matrix $\mathbf{A}$ of the first Euler flux $\boldsymbol{f}_1$ has, in the three-dimensional case, the explicit form

$$\mathbf{A}(\boldsymbol{w}) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ \frac{\kappa-1}{2}|\boldsymbol{v}|^2 - v_1^2 & (3-\kappa)v_1 & (1-\kappa)v_2 & (1-\kappa)v_3 & \kappa-1 \\ -v_1v_2 & v_2 & v_1 & 0 & 0 \\ -v_1v_3 & v_3 & 0 & v_1 & 0 \\ a_{5,1} & a_{5,2} & (1-\kappa)v_1v_2 & (1-\kappa)v_1v_3 & \kappa v_1 \end{pmatrix} \tag{17}$$

where

$$a_{5,1} = v_1\Big((\kappa-1)|\boldsymbol{v}|^2 - \kappa\frac{e}{\varrho}\Big),$$
$$a_{5,2} = \kappa\frac{e}{\varrho} - (\kappa-1)v_1^2 - \frac{\kappa-1}{2}|\boldsymbol{v}|^2.$$

After putting the third velocity component v_3 equal to zero, the fourth row and fourth column of $\mathbf{A}$ become zero with the exception of the fourth diagonal element. Leaving this column and row out, one obtains a 4×4 matrix

$$\mathbf{A}(\boldsymbol{w}) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{\kappa-1}{2}|\boldsymbol{v}|^2 - v_1^2 & (3-\kappa)v_1 & (1-\kappa)v_2 & \kappa-1 \\ -v_1v_2 & v_2 & v_1 & 0 \\ a_{4,1} & a_{4,2} & (1-\kappa)v_1v_2 & \kappa v_1 \end{pmatrix} \tag{18}$$

with

$$a_{4,1} = v_1\Big((\kappa-1)|\boldsymbol{v}|^2 - \kappa\frac{e}{\varrho}\Big),$$
$$a_{4,2} = \kappa\frac{e}{\varrho} - (\kappa-1)v_1^2 - \frac{\kappa-1}{2}|\boldsymbol{v}|^2,$$

which is exactly the two-dimensional form of the Jacobi matrix of the first Euler flux $\boldsymbol{f}_1$. Setting in (17) the last two velocity components v_2, v_3 equal to zero and leaving out the third and fourth row and third and fourth column, one analogously has the one-dimensional form

$$\mathbf{A}(\boldsymbol{w}) = \begin{pmatrix} 0 & 1 & 0 \\ \frac{\kappa-1}{2}|\boldsymbol{v}|^2 - v_1^2 & (3-\kappa)v_1 & \kappa-1 \\ v_1\big((\kappa-1)|\boldsymbol{v}|^2 - \kappa\frac{e}{\varrho}\big) & \kappa\frac{e}{\varrho} - (\kappa-1)v_1^2 - \frac{\kappa-1}{2}|\boldsymbol{v}|^2 & \kappa v_1 \end{pmatrix}. \tag{19}$$

Similar statements hold also for the positive and negative parts $\mathbf{A}^+$ and $\mathbf{A}^-$ of the matrix $\mathbf{A}$ and for the first Euler flux $\boldsymbol{f}_1$.

Thus, when one represents a 2D state vector $\boldsymbol{w} = (\varrho, \varrho v_1, \varrho v_2, e)^T$ in the 3D form $\boldsymbol{w} = (\varrho, \varrho v_1, \varrho v_2, 0, e)^T$, a three-dimensional ARS can be used for 2D computations. Analogously, representing a 1D state vector $\boldsymbol{w} = (\varrho, \varrho v_1, e)^T$ as $\boldsymbol{w} = (\varrho, \varrho v_1, 0, 0, e)^T$, a 3D ARS works as a 1D one.

The reader can see that it is sufficient to implement only the three-dimensional ARS's for 1D, 2D and 3D computations. The price for this benefit of structural simplicity is a low number of irrelevant operations in the 1D and 2D cases which do not affect the efficiency of the solver. With the Osher-Solomon scheme everything works in the same way and, moreover, the cost for the multi-dimensionality in one and two dimensions is even lower.

Below we show that the character of the finite volume method allows to store all data for the 1D and 2D computations in three-dimensional data structures.

5. Evaluation of the RHS

Except for the semi-implicit backward Euler scheme, all explicit as well as implicit time integration schemes which are implemented in the EULER package require explicit evaluation of the non-linear RHS vector function $\boldsymbol{F}(\boldsymbol{y}(t), t)$ from (7). This is done in a very simple way by means of one loop over all grid faces (see [4] for the explicit form of $\boldsymbol{F}$). Let us first shortly describe the data structure which has essential importance for the efficient evaluation of the RHS.

5.1. Data structures.

Grid points are stored in the structure

```
struct Point {
  double x, y, z;
};
```

In the 2D and axisymmetric 3D cases, we put $\mathtt{z} = 0$, for 1D and quasi-1D computations, $\mathtt{y} = \mathtt{z} = 0$. Point operators that facilitate adding points and multiplying them by a real constant are defined.

State vectors and their time derivatives as well as numerical fluxes are represented by the structure

```
struct Vector {
  double q1, q2, q3, q4, q5;
};
```

For 2D and axisymmetric 3D computations, the fourth component `q4` is set to zero. In the 1D and quasi-1D cases, both `q3` = `q4` = 0. Vector operators that facilitate adding vectors and multiplying them by a real constant are defined.

Solid-wall faces are stored in the structure

```
struct SolidFace {
    double sinfi, cosfi, sinpsi, cospsi;
    double size;
    FiniteVolume *e;
};
```

One needs an exact information on the direction of the normal vector to all grid faces for the computation of numerical fluxes. We describe this direction (in 3D) using two angles, φ and ψ. The information on the direction is stored in terms of the values $\sin\varphi$, $\cos\varphi$, $\sin\psi$ and $\cos\psi$ (rather than by means of only the two values φ and ψ). The use of this data during the computation is so extensive that it makes sense to store the pre-computed values in order to improve efficiency of the solver. The usual convention is that the normal vector for solid-wall and inlet-outlet faces points out of the domain Ω. The variables `size` and `e` represent the size of the face and the pointer to the adjacent finite volume, respectively.

Inlet-outlet faces are stored by means of the structure

```
struct IOFace {
    double sinfi, cosfi, sinpsi, cospsi;
    double size;
    int index;
    FiniteVolume *e;
    Vector bdystate;
};
```

In addition to the variables involved in the `SolidFace` data structure, one needs to store the index of the grid face in order to distinguish different boundary conditions at different inlet-outlet parts of the boundary $\partial\Omega$ and a vector `bdystate` representing the boundary state. This vector is used for the computation of the inlet-outlet numerical flux. We will describe the role of the vector `bdystate` in detail in Subsection 5.3 devoted to the treatment of inlet-outlet boundary conditions.

Interior faces are stored in the form

```
struct IntFace {
    double sinfi, cosfi, sinpsi, cospsi;
    double size;
    FiniteVolume *e1, *e2;
};
```

For interior faces, the following convention is used: the normal vector to the face points from the element `e1` to the element `e2`. Computing the numerical flux H_{int}, one uses the state at `e1` as $\boldsymbol{w}_i$ and the state at `e2` as $\boldsymbol{w}_j$ in the relation (9).

Finite volumes are represented by the structure

```
struct FiniteVolume {
    Vector y, ydot;
    double volume;
};
```

Vector variables `y` and `ydot` represent coefficients of the piecewise constant approximate solution $\boldsymbol{w}_h(\boldsymbol{x}, t)$ and its time derivative $\frac{\partial}{\partial t}\boldsymbol{w}_h(\boldsymbol{x}, t)$ on the finite volume, respectively. Let us remark that these constant values approximate the exact solution at its center of gravity.

Solid-wall faces, inlet-outlet faces, interior faces and finite volumes are stored in four separate lists.

5.2. Algorithm.

At the beginning of the computation, the initial condition for the flow is distributed into all finite volumes, filling appropriately their variable `y`.

Evaluation of the right-hand side $\boldsymbol{F}$ is done in one loop over all grid faces. The variable `ydot` in all finite volumes is set to zero at the beginning of each loop. Numerical fluxes for all solid-wall and inlet-outlet faces are computed using the state variable `y` from the adjacent finite volume `e` and the information about rotations of the face and its size. The numerical flux is divided by the volume size of the adjacent finite volume `e` and added with the negative sign to its `ydot` variable. For the interior faces, the numerical flux divided by the volume size of `e1` is added with the negative sign to its `ydot` variable. The same value of the numerical flux, divided by the volume size of the second adjacent finite volume `e2`, is added with a positive sign to its `ydot` variable. It is important that the normal to the face points from `e1` to `e2`.

5.3. Inlet-outlet boundary conditions.

In general, the inlet-outlet boundary conditions for the compressible Euler equations have not been completely understood yet. There are several treatments based more or less on the theory of characteristics. The following three approaches are offered by EULER. They can be applied in various combinations to different parts of the inlet-outlet boundary of the domain Ω, their behaviour is slightly different and it depends on the experience of the user to decide which one of them is optimal for her/his simulation.

At the beginning of the computation, user-defined boundary states are always distributed to all inlet-outlet faces. Often they match the initial condition.

Approach 1: The state represented by the vector `bdystate` in the `IOFace` data structure is assumed to lie exactly on the inlet-outlet face. Before each evaluation of the RHS, some quantities are extrapolated from the adjacent finite volume to the inlet-outlet face or the other way according to the type of flow at the adjacent finite volume. EULER offers several possibilities for the choice of combinations of extrapolated quantities (see the user's guide on the Internet pages). Let us introduce one possible choice as an example:

- The whole vector `y` from the adjacent finite volume `e` is copied to the vector `bdystate` of the inlet-outlet face if the state `y` represents a supersonic outlet (i.e. the corresponding Mach number is greater than one and the normal component of the velocity has the same direction as the normal vector to the inlet-outlet face).
- Density and all velocity components are copied from the vector `y` to the vector `bdystate` and the pressure is copied the other way if the state `y` represents a subsonic outlet (i.e. the corresponding Mach number is lower than one and the normal component of the velocity has the same direction as the normal vector to the inlet-outlet face).
- Density and all velocity components are copied from the vector `bdystate` to the vector `y` and the pressure is copied the other way if the state `y` represents a subsonic inlet (i.e. the corresponding Mach number is lower than one and the normal component of the velocity has the opposite direction to the normal vector to the inlet-outlet face).
- The whole vector `bdystate` is copied to the vector `y` if the state `y` represents a supersonic inlet (i.e. the corresponding Mach number is greater than one and the normal component of the velocity has the opposite direction to the normal vector to the inlet-outlet face).

The numerical flux through the inlet-outlet face is then computed in such a way that the first Euler flux $\boldsymbol{f}_1$ is applied directly to the vector `bdystate`.

Approach 2: The state represented by the vector `bdystate` in the `IOFace` data structure is assumed to lie at the center of gravity of a fictitious finite volume which lies outside of the domain Ω and is adjacent to the inlet-outlet face. Before each evaluation of the RHS, some quantities are extrapolated from the adjacent finite volume `e` to the fictitious one or the other way according to the type of flow at the adjacent finite volume `e` (exactly in the same way as shown in Approach 1). The numerical flux is computed using (9) with both the states in the real and fictitious adjacent finite volumes. The time derivative `ydot` is affected by the numerical flux

only in the real finite volume. This approach leads to a more robust scheme which gives results very similar to the previous one.

Approach 3: The state represented by the vector `bdystate` in the `IOFace` data structure is assumed to represent a fictitious infinite reservoir ("infinite finite volume") which lies outside the domain Ω and is adjacent to the inlet-outlet face. No quantities are extrapolated and the numerical flux is computed using both the vector `y` corresponding to the adjacent finite volume `e` and the constant reservoir state `bdystate`. This approach leads to a very robust scheme which is, for certain models, safer and closer to the physical reality than the previous ones. In fact, this approach extends the geometrically finite computational domain to infinity and removes part of the inlet-outlet boundary. It can be applied very successfully to the simulation of reservoirs and, in fact, very often there is a reservoir in the reality even if it is not involved in the model. The standard inlet-outlet boundary conditions which are the source of huge theoretical and also computational problems are often a consequence of the removal of such a reservoir. These *reservoir boundary conditions* can also be applied very successfully to problems where one can be sure that the flow near the inlet-outlet boundary or some of its parts remains unchanged.

6. Time integration

The advantage of the application of the method of lines is that it strictly separates the time integration from the space discretization, which increases the modularity of the numerical method. One can use any suitable ODE initial value problem solver for the approximate solution of the ODE system (7), (8) arising from the semi-discretization of the compressible Euler equations in space. Several schemes of explicit, semi-implicit and fully implicit types are implemented in EULER.

6.1. Explicit schemes.

Explicit Runge-Kutta schemes of the first, second and fourth orders are the candidates for the explicit time integration in EULER. As they are widely known, we think it is not necessary to repeat their exact definition.

In general these methods are not very suitable for the numerical integration of stiff problems, which our problem definitely is. These schemes are unstable unless the time step is very small, restricted by the diameter of the space discretization as well as by certain properties of the flow (the *CFL condition*, see e.g. [6]). Without adaptivity, lack of precision often leads to approximate solutions containing negative density or pressure values. In this sense, one can easily construct flow problems which do not have any large number of degrees of freedom and which cannot be solved by non-adaptive explicit schemes in any reasonable time, anyway.

Despite these facts explicit schemes are still very popular because of their very simple implementation, clearly understandable function and very low memory requirements. In this way, simpler flow problems are solvable but the question about the precision of the time integration should not be asked. When one just uses two different time steps, unless the computation breaks down, one obtains two different solutions.

In our opinion, it is reasonable to use the explicit schemes when debugging the code as they are too simple to contain hidden errors. In particular, they can be useful when a new treatment of boundary conditions is implemented or when a completely new flow problem is solved. Nevertheless, it is better to switch to higher-quality schemes as soon as possible. The exception are of course huge 3D flow problems where the memory limit is exceeded when using more precise and stable schemes.

6.2. Semi-implicit schemes.

For the purpose of improving the stability and precision of time integration, we implemented a semi-implicit scheme based on the backward Euler method with linearized approximate Riemann solvers. We have not linearized the ARS by Osher-Solomon because it would be extremely complicated and, in the linearized form, this ARS loses its main advantages with respect to the other ARS's.
We linearize the ARS by Steger-Warming (14) as

$$\boldsymbol{f}_R^{(\mathrm{lin})}(\boldsymbol{q}_L^0, \boldsymbol{q}_R^0, \boldsymbol{q}_L, \boldsymbol{q}_R) = \mathbf{A}^+(\boldsymbol{q}_L^0)\boldsymbol{q}_L + \mathbf{A}^-(\boldsymbol{q}_R^0)\boldsymbol{q}_R, \tag{20}$$

the linearized ARS by Vijayasundaram (15) used in our code has the form

$$\boldsymbol{f}_R^{(\mathrm{lin})}(\boldsymbol{q}_L^0, \boldsymbol{q}_R^0, \boldsymbol{q}_L, \boldsymbol{q}_R) = \mathbf{A}^+\left(\frac{\boldsymbol{q}_L^0 + \boldsymbol{q}_R^0}{2}\right)\boldsymbol{q}_L + \mathbf{A}^-\left(\frac{\boldsymbol{q}_L^0 + \boldsymbol{q}_R^0}{2}\right)\boldsymbol{q}_R, \tag{21}$$

Van Leer ARS (16) is linearized using the relation

$$\boldsymbol{f}_1(\boldsymbol{w}) = \mathbf{A}(\boldsymbol{w})\boldsymbol{w} \tag{22}$$

as

$$\begin{aligned}\boldsymbol{f}_R^{(\mathrm{lin})}(\boldsymbol{q}_L^0, \boldsymbol{q}_R^0, \boldsymbol{q}_L, \boldsymbol{q}_R) = {} & \frac{\mathbf{A}(\boldsymbol{q}_L^0)\boldsymbol{q}_L + \mathbf{A}(\boldsymbol{q}_R^0)\boldsymbol{q}_R}{2} \\ & - \frac{1}{2}|\mathbf{A}|\left(\frac{\boldsymbol{q}_L^0 + \boldsymbol{q}_R^0}{2}\right)(\boldsymbol{q}_R - \boldsymbol{q}_L).\end{aligned} \tag{23}$$

In relations (20), (21), (23), the variables $\boldsymbol{q}_L^0, \boldsymbol{q}_R^0$ correspond to the approximate solution at the previous time level. Linearizing also the solid-wall and inlet-outlet

numerical fluxes, we end up with a sparse system of linear algebraic equations. This system is preconditioned using the incomplete LU decomposition and the user can choose several conjugate and bi-conjugate gradient schemes for its iterative solution.

It turns out that the semi-implicit scheme is really remarkably more precise and stable than the explicit ones while it needs less memory than the higher-order adaptive fully implicit schemes which will be mentioned in the next subsection. Therefore, the semi-implicit scheme can be successfully applied to (not extremely large) flow problems which are not solvable with explicit schemes and which are, at the same time, too large for the fully implicit ones.

Obviously, the described linearized backward Euler method is not unconditionally stable and it still needs the CFL condition to keep the time step sufficiently small, but in comparison with the explicit Runge-Kutta schemes, the size of the time step can be several times larger. In our opinion, the quality of the scheme can be essentially improved by applying a suitable multigrid or algebraic multigrid technique to the iterative solution of the sparse system. This topic would be a natural continuation of our work in this direction.

6.3. Implicit higher-order adaptive schemes.

Now we come to the most precise time-integration schemes which are offered by EULER. Profiting from the modularity achieved by the separation of the space and time discretization of partial differential equations, we apply highly efficient ODE schemes offered by the standard ODE packages ODEPACK (see [1]) and DDASPK3.0 (which is a descendant of DDASSL [2]) for the time integration. These packages contain schemes based mainly on the backward difference formula (BDF). In both cases, the detailed description can be found directly in the source codes.

These ODE packages always help to overcome stability problems. One no longer needs to guess a value for the CFL constant and to repeat the computation if his guess has been too large. What is even more important, the implicit solvers always perform a certain precision check which helps to eliminate problems with negative pressures and densities during the computation. These troubles are usual for problems with badly shaped meshes or steep gradients, mainly when being solved by the explicit or semi-implicit schemes. In such cases, due to lack of precision of the approximate time integration, the approximate solution to the semi-discrete problem easily leaves the exact one and runs out of the set of admissible states.

The only restriction for the presented ODE packages is their large memory requirements. Namely, most of the schemes offered are proposed for the solution of stiff systems and therefore use the Jacobi matrix of the right-hand side, which costs a lot of memory. Moreover, due to their multistep algorithms, the solvers need to store the complete solution vector at several times levels. On the other hand, also for large

problems there is a possibility to take advantage of a precise implicit higher-order adaptive scheme of the Adams-Bashforth type offered by ODEPACK for non-stiff problems, where the Jacobi matrix is not used. This scheme is quite efficient also for large problems. Even if one does not have troubles with negative pressures and densities, controlled precision is particularly useful for nonstationary problems when one is interested in the precise solution at all time levels.

7. Multi-dimensional computations

As indicated in Sections 4 and 5, three-dimensional ARS's and three-dimensional data structures allow us to perform 1D, quasi-1D, 2D, axisymmetric 3D and 3D computations. We would like to make several remarks on the details in this section.

7.1. One-dimensional case.

In 1D, the computational domain Ω is a bounded interval and the finite volumes have the form of intervals. EULER allows non-equidistant partitions. There are no solid walls in this case and only two inlet-outlet faces at both ends of the interval are considered. All grid faces (corresponding in 1D to the grid points of the partition of Ω) have a uniform nonzero size (there is no preferred value but one must use the same value also for the computation of the sizes of the finite volumes).

7.2. Quasi-1D problems.

EULER provides the analytical solution (more exactly, the numerical construction of the analytical solution) of the stationary quasi-1D compressible Euler equations as well as their finite volume solution in the non-stationary case. In the latter case, the finite volume scheme is very similar to the 1D one. There are the following two differences: the size of the grid faces is not uniform and one needs to consider the solid walls due to the non-conservativity of the quasi-1D momentum equation. For details on both the analytical and the numerical approach we refer to [7].

7.3. Two-dimensional computations.

In 2D we consider any bounded domain Ω with continuous, piecewise linear boundary. Complete information on the geometry is read as input data from a grid file. The boundary of the domain is splitted into several parts which are assigned different indices. These indices allow the user of EULER to prescribe various boundary conditions for different parts of the boundary. Currently, EULER uses unstructured triangular meshes in 2D but it is only a question of preprocessing to allow for e.g. polygonal finite volumes. Up to now, triangles have been sufficient.

7.4. Axisymmetric 3D case.

As the reader knows, axisymmetric schemes are, in general, enormously advantageous for axisymmetric 3D problems in comparison with the purely 3D schemes. In our case, for the axisymmetric problems the 3D equations are reduced to their axisymmetric form, which is similar to the 2D one. More exactly, the left-hand side of the axisymmetric 3D equations is identical with the left-hand side in the 2D case. There are only some additional source terms on the right-hand side in the axisymmetric case.

In EULER, we consider the x-axis as the axis of symmetry. One uses symmetric half of the axial cut of the 3D domain as the 2D geometry for the axisymmetric 3D computation. As usual, all grid points are assumed to have nonnegative y-coordinates. The 2D geometry is covered by an unstructured triangular grid which is read by EULER and pre-processed (neighbouring relations, rotations, volume and edge sizes, construction of the centers of gravity, mapping of boundary conditions etc.) in the standard 2D way. In the axisymmetric context, finite volumes which are represented by grid triangles are in fact 3D axisymmetric rings (created by the rotation of the grid triangles around the x-axis). Volume sizes of the finite volumes (stored in the variable `volume` of the data structure `FiniteVolume`) are therefore changed to the volume sizes of these rings. Original 2D volume sizes of all grid triangles are stored in a separate list because they are needed for the evaluation of the source terms. Analogously, grid edges represent 3D axisymmetric surfaces created by the rotation of the edges around the x-axis and the variable `size` of all grid faces is changed to the size of this 3D surface. See the source code for the exact formulae describing the 3D axisymmetric volume and surface sizes. After computing the RHS of the system (7), (8), one has to add the source term

$$S = \begin{pmatrix} 0 \\ 0 \\ 2\pi p V_{2D}/V_{3D} \\ 0 \\ 0 \end{pmatrix} \tag{24}$$

to the vector variable `ydot` in all finite volumes. In the definition (24) of the source terms, p has the meaning of the current value of pressure in the finite volume and the values V_{2D} and V_{3D} represent the 2D volume size of the grid triangle and the 3D volume size of the corresponding axisymmetric ring, respectively.

7.5. Three-dimensional computations.

Analogously to the purely 2D case, the domain Ω is bounded and assumed to have a continuous, piecewise linear boundary. Its geometry is defined by means of an

input grid file. Unstructured three-dimensional grids consisting of tetrahedra have been sufficient so far. Similarly as in the 2D case, a generalization of implemented finite volumes to convex polyhedra is only a formal problem of preprocessing.

7.6. Published results computed by EULER.

The best possibility to view numerical results obtained by EULER is to visit the Internet site `http://www.numa.uni-linz.ac.at/Staff/solin`. Most of these results also have been published (e.g. in [3], [4], [5], [6])— a complete list of references is present at the above Internet page.

References

[1] *A. C. Hindmarsh*: ODEPACK, A systematized collection of ODE solvers. In: Scientific Computing (IMACS Transactions on Scientific Computation Volume 1). North-Holland, Amsterdam, 1983, pp. 55-64.

[2] *L. R. Petzold*: A description of DDASSL: A differential/algebraic system solver. Sandia Report No. Sand 82-8637. Sandia National Laboratory, Livermore, CA, 1982.

[3] *K. Segeth, P. Šolín*: Application of the method of lines to compressible flow (Computational aspects). Proceedings of the 10th PANM Seminar, Lázně Libverda, 2000. Mathematical Institute of the Academy of Sciences of the Czech Republic, Praha, 2000. (In Czech.)

[4] *P. Šolín, K. Segeth*: Application of the method of lines to the nonstationary compressible Euler equations. Internat. J. Numer. Methods Fluids. Submitted.

[5] *P. Šolín*: On the method of lines (Application in fluid dynamics and a-posteriori error estimation). Doctoral dissertation. Faculty of Mathematics and Physics, Charles University, Prague, 1999.

[6] *P. Šolín*: Three-dimensonal Euler equations and their numerical solution by the finite volume method. Master thesis (Part 1). Faculty of Mathematics and Physics, Charles University, Prague, 1996.

Authors' addresses: *P. Šolín*, Institute of Computational Mathematics, Johannes Kepler University, Altenbergerstrasse 69, A-4040 Linz, Austria, e-mail: `solin@numa.uni-linz.ac.at`; *K. Segeth*, Mathematical Institute of the Academy of Sciences of the Czech Republic, Žitná 25, CZ-115 67 Praha 1, Czech Republic, e-mail: `segeth@math.cas.cz`.