Comparison of Explicit Method of Solution for CFD Euler Problems using MATLAB® and FORTRAN 77

A. Nordli*, H. Khawaja
UiT-The Arctic University of Norway

ABSTRACT

This work presents a comparison of an explicit method of solution for an inviscid compressible fluid mechanics problem using Euler equations for two-dimensional internal flows. The same algorithm was implemented in both FORTRAN 77 and MATLAB®. The algorithm includes Runge–Kutta time marching scheme with smoothing. Both solvers were initialized in the same manner. In addition, it was ensured that both solvers have the exact same values for time step, convergence criteria, boundary conditions, and the grid. The only difference between the two solvers was the precision of variables.

The problem solved was a two-dimensional dual bump with an accelerating flow through a duct. The same algorithm solving the Euler equations of fluid flow is implemented in both FORTRAN 77 and MATLAB®, and applied to identical input. While the solutions look qualitativly the same, a 20% difference in the stationary solution is observed. No claim is made of the relevance of the computations to actual fluid flow, rather the key takeaway being that two finite and deterministic computations of the same algorithm on the same input in FORTRAN 77 and MATLAB® produce different output.

1. INTRODUCTION

FORTRAN 77 computing language has been externsively used by scientists and mathematicians for numeric computation and scientific computing [1]. Similarly MATLAB® is a also common software being widely used among the research commuity [2]. Though both computing software are capable of managing complex mathematical calculations, there are subtle difference in their implementation [3]. Complex CFD problems has been implemented in FORTRAN and validated against experiments [4-6]. Similar studies are reported using MATLAB® as well, e.g. [7].

In this paper the same CFD Euler algorithm with identical input is implemented in both FORTRAN 77 and MATLAB®, and the results are compared. It is found that there is a 20% difference in the outputs. The algorithm solves fluid mechanics Euler equations [8] in a planar specified bounded domain for a steady state solution. The Euler equations is a system of partial differential equations given in Equation (1).

In this work the model and solutions in themselves are of secondary interest. The point is to use this model to highlight that different implementations of the same algorithm in different programming languages can result in different outputs for identical inputs. Therefore, for the

*Corresponding Author: anders.s.nordli@uit.no

$$\begin{cases} \partial_t \rho + \nabla \cdot (\rho \boldsymbol{u}) = 0, \\ \partial_t (\rho \boldsymbol{u}) + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u}) + \nabla p = 0, \\ \partial_t E + \nabla \cdot \big(\boldsymbol{u}(E + p)\big) = 0, \\ E = \frac{p}{\gamma - 1} + \frac{1}{2} \rho |\boldsymbol{u}|^2. \end{cases} \tag{1}$$

main results of this work to be valid it is enough that the algorithm produces finite and deterministic output from the admissible input. The selection of this model for this purpose was that we had a FORTRAN 77 code available and wished to rewrite it into MATLAB®. Since FORTRAN 77 does not support object-oriented programming, we wanted to rewrite the code into MATLAB® to make it easier to maintain and adapt the code. After the rewriting was done and we started to compare the results we discovered that the different versions produced qualitatively similar, but quantitatively different output from the same input. In Section 2 the algorithm details are provided, while in Section 3, the results are presented and discussed.

2. METHOD

The main difference in the implementation of the algorithm in FORTRAN 77 and MATLAB®, is that the MATLAB® code is object-oriented, while the FORTRAN 77 code is not [9]. However, object-orientation is only a way of organizing the data and methods into objects and will not in itself affect the variables and functions used. The algorithm is a numerical solver for the time-dependent system of partial differential Equation (1), which is used for as long as the changes from one-time step to the next is above a threshold value given as input to the algorithm.

The time stepping algorithm is shown in Figure 1. First boundary conditions are updated, then one-time step is performed with a Runge—Kutta method [10]. Solutions are then smoothed, and secondary variables are computed. The primary variables are density, momentum density, and energy density, while the secondary variables are pressure, temperature, and velocity field. Finally, the solutions are checked for convergence, and if convergence is obtained according to an input convergence criterion the solution is kept. For the first time step a guess at a solution is computed, and for the algorithm to converge to the correct solution in a feasible number of time steps it is essential that the first guess of the solution is not too far off the true solution. If the guessed solution is far from the steady state, the solver could potentially need a too high number of time steps. It could also tend toward a different steady state or violate some of the assumptions for the algorithm to work such as flow from left to right.

The computational domain and grid is shown in Figure 2 and is identical for the MATLAB® and FORTRAN 77 as the bottom grid coordinates, top grid coordinates, and number of grid points in each direction are part of the input to the algorithm. The minimum step size in space is $\Delta x_{min} = 0.12$.

Central differences are used to approximate fluxes at gridpoints. In Figure 3 the grid close to grid point (i, j) is shown, with the area of the grid cells indicated by $A_{i\pm\frac{1}{2}, j\pm\frac{1}{2}}$.

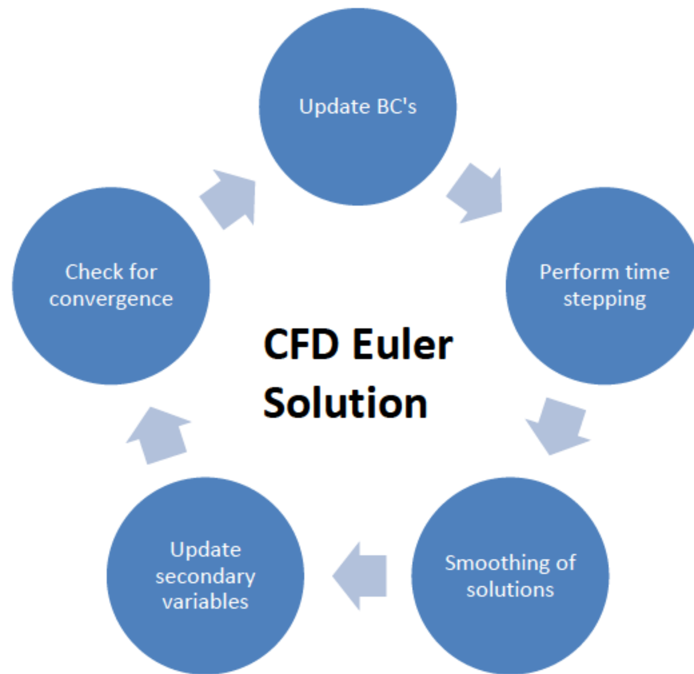

Figure 1: Flow chart of the algorithm.

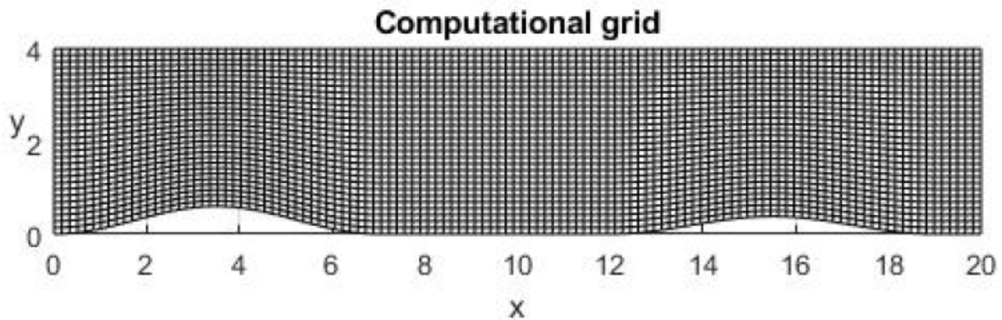

Figure 2: The computational grid employed in the computation. It is identical for the MATLAB® and FORTRAN 77 versions.

The lengths of grid edges are calculated as shown in Equation (2).

$$\Delta l_{ix}^{j} = x_{i+1,j} - x_{i,j},\ \Delta l_{jx}^{i} = x_{i,j+1} - x_{i,j},\ \Delta l_{iy}^{j} = y_{i+1,j} - y_{i,j},\ \Delta l_{jy}^{i} = y_{i,j+1} - y_{i,j} \quad (2)$$

and in vector form $\boldsymbol{\Delta l_i^j} = \left(\Delta l_{ix}^{j}, \Delta l_{iy}^{j}\right), \boldsymbol{\Delta l_j^i} = \left(\Delta l_{jx}^{i}, \Delta l_{jy}^{i}\right)$. A conservation law $\partial_t w + \nabla \cdot \boldsymbol{F} = 0$ with flux function $\boldsymbol{F} = (F^1, F^2)$ can then be approximated at grid point (i, j) as follows. First define fluxes across edges as shown in Equation (3).

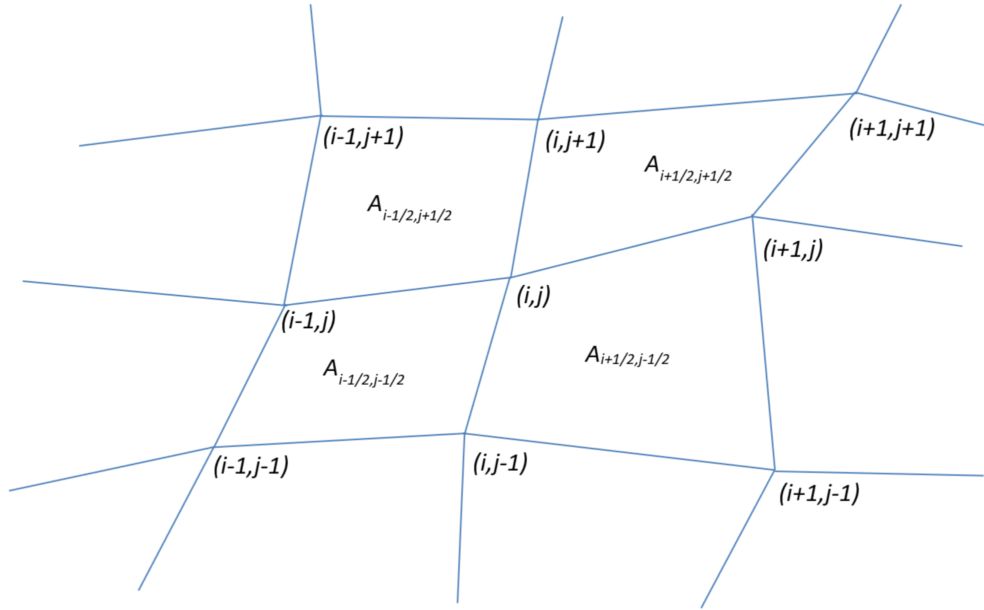

Figure 3: Detail of grid close to grid point (i, j). The areas of the neighboring grid cells are labeled $A_{i\pm\frac{1}{2},j\pm\frac{1}{2}}$.

$$\begin{aligned}\Phi^1_{i,j+\frac{1}{2}} &= \frac{1}{2}\left(\boldsymbol{F}_{i,j} + \boldsymbol{F}_{i,j+1}\right) \cdot \boldsymbol{\Delta l}^{j}_{i},\\ \Phi^2_{i+\frac{1}{2},j} &= \frac{1}{2}\left(\boldsymbol{F}_{i,j} + \boldsymbol{F}_{i+1,j}\right) \cdot \boldsymbol{\Delta l}^{i}_{j}.\end{aligned} \tag{3}$$

The flux out of grid cell $\left(i + \frac{1}{2}, j + \frac{1}{2}\right)$ is then given by Equation (4).

$$F_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{1}{A_{i+\frac{1}{2},j+\frac{1}{2}}}\left(\Phi^1_{i+1,j+\frac{1}{2}} - \Phi^1_{i,j+\frac{1}{2}} + \Phi^2_{i+\frac{1}{2},j+1} - \Phi^2_{i+\frac{1}{2},j}\right) \tag{4}$$

and the conservation law approximated as shown in Equation (5).

$$\partial_t w_{i,j} \approx -\frac{1}{4}\left(F_{i-\frac{1}{2},j-\frac{1}{2}} + F_{i-\frac{1}{2},j+\frac{1}{2}} + F_{i+\frac{1}{2},j-\frac{1}{2}} + F_{i+\frac{1}{2},j+\frac{1}{2}}\right) \tag{5}$$

For the temporal discretization a four step explicit Runge—Kutta method with Butcher tableau [11] given in Figure 3 is employed.

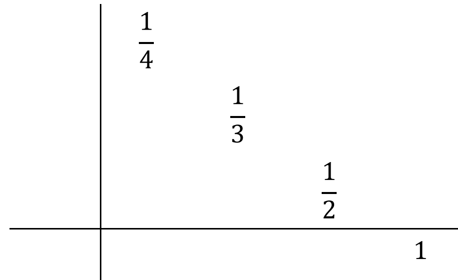

Figure 4: Butcher tableau for the employed Runge—Kutta method. All off-diagonal elements are zero.

The time step is set to $\Delta t = 3.0 \cdot 10^{-5}$, since the maximal characteristic speed at final time $|\boldsymbol{u}| + c = 1.3 \cdot 10^5$. Strictly speaking the CFL-condition [12] is not fulfilled with this time step, however the computation stays finite at all steps for both the FORTRAN 77 and MATLAB® implementation. A reduction in Δt so that the CFL-condition was satisfied was attempted, but the numerical fluxes where then on the scale of round off errors. Thus, instead of increasing accuracy it made the computations less stable and more computationally costly. No claim is therefore made of the relevance of the computations to actual fluid flow, rather the key takeaway being that two finite and deterministic computations of the same algorithm on the same input in FORTRAN 77 and MATLAB® produce different output. Note that if there are no sound waves propagating in the solution, then the time step should be small enough since $|\boldsymbol{u}| \ll c$ in the converged solutions.

For stability, the solution has to be smoothed at each time step. A numerical solution ψ is smoothed at (i, j) and time step n as shown in Equation (6).

$$\begin{aligned} \psi_{i,j}^{n,avg} &= \frac{1}{4}\left(\psi_{i-1,j}^{n} + \psi_{i+1,j}^{n} + \psi_{i,j-1}^{n} + \psi_{i,j+1}^{n}\right), \\ \psi_{i,j}^{n,smoothed} &= (1-\kappa)\psi_{i,j}^{n} + \kappa\left(\psi_{i,j}^{n,avg} + \lambda\psi_{i,j}^{n-1} + (1-\lambda)\nu\left(\psi_{i,j}^{n} - \psi_{i,j}^{n,avg}\right)\right) \end{aligned} \tag{6}$$

where $\kappa = 0.2500, \lambda = 0.999999, \nu = 0.9000.$

The solution is checked for convergence and determined to be converged if the changes in the variables from one-time step to the next falls below a threshold given in the input. If either the maximal difference in any of the variables exceeds the threshold, or the average difference exceeds half of the threshold, the convergence test fails, and the next time step is performed.

The boundary conditions used at top and bottom of the domain are solid walls, i.e. $\boldsymbol{u} \cdot \boldsymbol{v} = 0$. Boundary conditions have to be applied at the right and left boundaries as well. On the right boundary a new pressure is set to p_{down}. On the left the inlet density $\rho_{in,j}^{n}$ at gridpoint $1, j$ at time step n is set as shown in Equation (7).

$$\rho_{in,j}^{n} = \min\left\{0.75 \cdot \rho_{in,j}^{n-1} + 0.25 \cdot \rho_{1,j}^{n-1}, 0.9999 \cdot \rho_{stagnation}\right\} \tag{7}$$

Note that this will not change the density at the boundary. The idea is to ρ_{in} affect the inlet fluxes and energy density and in that manner relax the left-hand boundary to the appropriate final state in order to make the method more stable. Then the pressure at the inlet is set as shown in Equation (8).

$$p_{1,j}^{n} = p_{stagnation} \cdot \left(\frac{\rho_{in,j}^{n}}{\rho_{stagnation}}\right)^{\gamma} \tag{8}$$

while $(\rho u)_{1,j}^{n}$ and $(\rho v)_{1,j}^{n}$ are computed using Equation (9).

$$\begin{aligned}(\rho u)_{1,j}^{n}&=\rho_{in,j}^{n}\sqrt{2c_{p}T_{stagnation}\left(1-\left(\frac{\rho_{in,j}^{n}}{\rho_{stagnation}}\right)^{\gamma-1}\right)}\cos\alpha,\\(\rho v)_{1,j}^{n}&=\rho_{in,j}^{n}\sqrt{2c_{p}T_{stagnation}\left(1-\left(\frac{\rho_{in,j}^{n}}{\rho_{stagnation}}\right)^{\gamma-1}\right)}\sin\alpha,\end{aligned} \tag{9}$$

and respectively velocities are computed using Equation (10).

$$\begin{aligned}u_{1,j}^{n}&=\frac{(\rho u)_{1,j}^{n}}{\rho_{1,j}^{n}},\\v_{1,j}^{n}&=\frac{(\rho v)_{1,j}^{n}}{\rho_{1,j}^{n}}.\end{aligned} \tag{10}$$

The energy density is calculated using Equation (11).

$$(\rho e)_{1,j}^{n} = \rho_{in,j}^{n}\left(c_{v}T_{stagnation}\left(\frac{\rho_{in,j}^{n}}{\rho_{stagnation}}\right)^{\gamma-1} + c_{p}T_{stagnation}\left(1-\left(\frac{\rho_{in,j}^{n}}{\rho_{stagnation}}\right)^{\gamma-1}\right)\right) \tag{11}$$

To avoid the problem being ill posed one must restrict the solution to the states where net mass flow is always positive across the boundary.

3. RESULTS AND DISCUSSIONS

In Figure 5 the pressure field of the converged solution is shown, and in Figure 6 is the corresponding Mach field. Note that the flow is supersonic downstream of each of the bumps. Still the Mach number is less than 1.7. The pressure and Mach fields look reasonable, so it seems that the algorithm is producing sensible results.

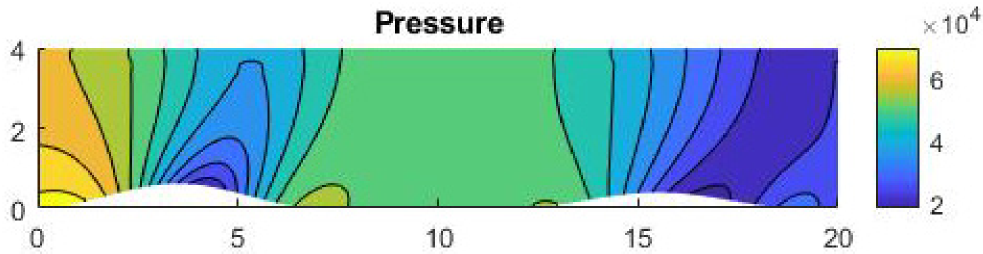

Figure 5: The final pressure field after convergence is obtained.

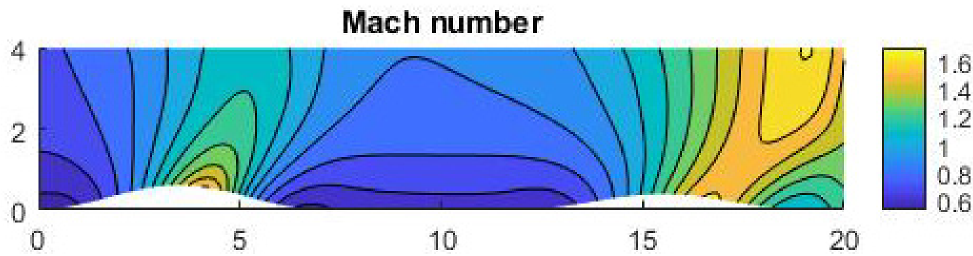

Figure 6: The final Mach numbers after convergence is obtained. Note that the Mach number is lower than 1.5, but that the flow is supersonic downstream of the bumps.

In Figure 7 the final density computed by FORTRAN 77, MATLAB®, and the relative differences are plotted. Qualitatively the solutions look similar, but from the plot of relative differences one can see that there is a 20% difference in the densities.

In Figure 8 the convergence plots of the estimated errors for 20000 and 100000 are shown. One can see that the errors are quite similar for the first 5000 time steps, and then they diverge. It looks like the FORTRAN 77 code reaches steady state after 30000 time steps, while MATLAB® needs 100000 time steps.

On the other hand, if one looks at the inlet to outlet flow ratios in Figure 10, it is clear that from a mass conservation perspective the MATLAB® version is superior.

Even though the errors seem similar for the first-time steps, the discrepancies are at the same level all through the computation. This is revealed in Figure 11 where the algorithm of the absolute differences in errors is plotted. The curious dips in the curve means that the sign of the difference switches at that point.

It is clear that both the FORTRAN 77 and MATLAB® implementations of the algorithm produces finite and sensible results, but that they differ in a number of ways. First of all, they

converge to different solutions, with the difference in density of about 20%. This is a serious discrepancy for what is supposed to be identical algorithm. Second, the path to convergence is different. It seems that both solvers resolve dynamical effects, and then the FORTRAN 77 solver quickly converges to a steady state, while the MATLAB® solver uses over three times as many time steps to get to a similar level of error estimate. Third the inlet to outlet flow ratios are different throughout the computation. For a true steady state conservation of mass implies that it should be equal to one. In this regard the MATLAB® solver outperforms the FORTRAN 77 solver.

The reasons for the different results are not clear. One possibility is that FORTRAN 77 uses 32-bit numbers [13], while MATLAB® uses 64-bit numbers [14]. MATLAB® is thus more accurately represents real numbers than FORTRAN 77. This could result in more numerical diffusion in the FORTRAN 77 solver. Another possibility is more on the level of compilers and how binary operations are performed on a fundamental level. The complexity of the algorithm also makes it harder to identify exactly where computations start to differ.

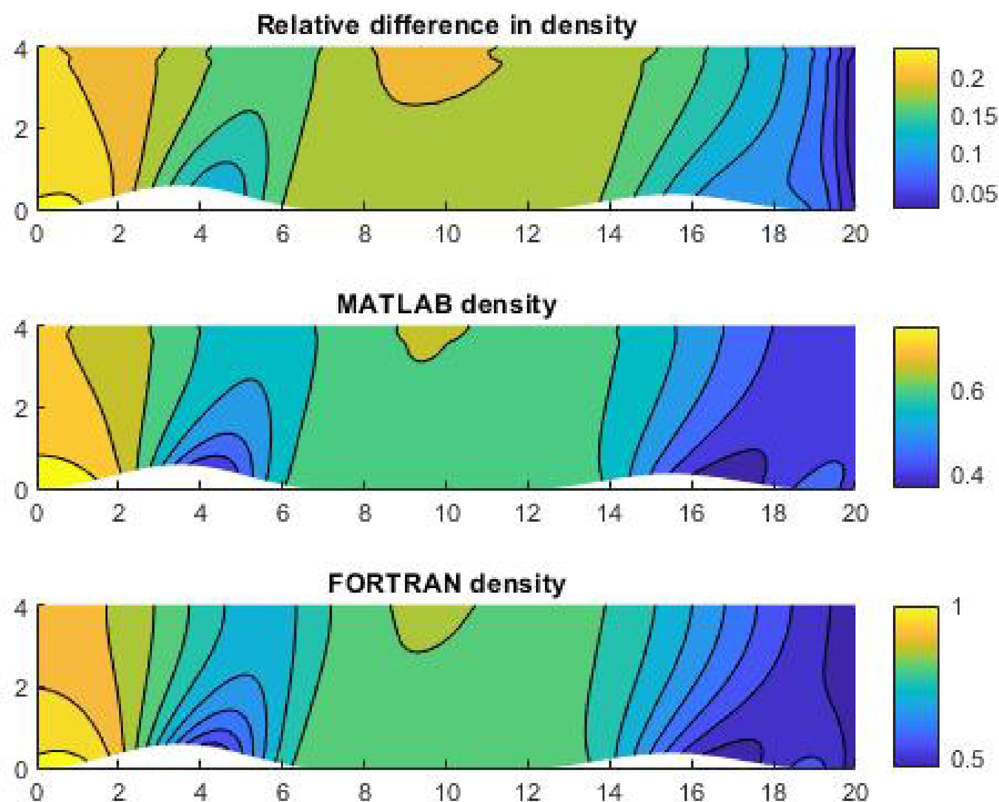

Figure 7: The final density after convergence is obtained. The plot shows that there is a discrepancy between the density computed by FORTRAN 77 and MATLAB® even though both solutions have satisfied the convergence criterion.

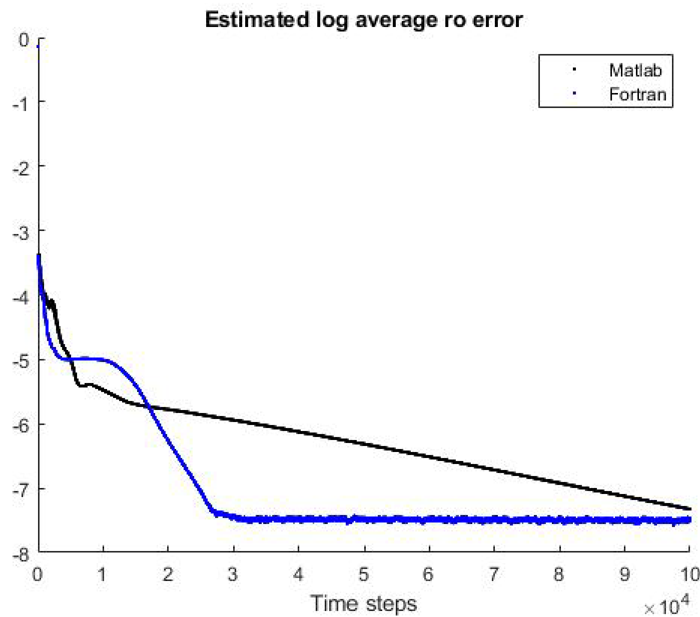

Figure 8: The logarithm of the estimate of the error in density averaged over the spatial grid. The graph is up to 100000 time steps. The blue line represents FORTRAN 77, while the black line represents MATLAB®. The horizontal line is the convergence criterion. The same pattern applies to the other variables.

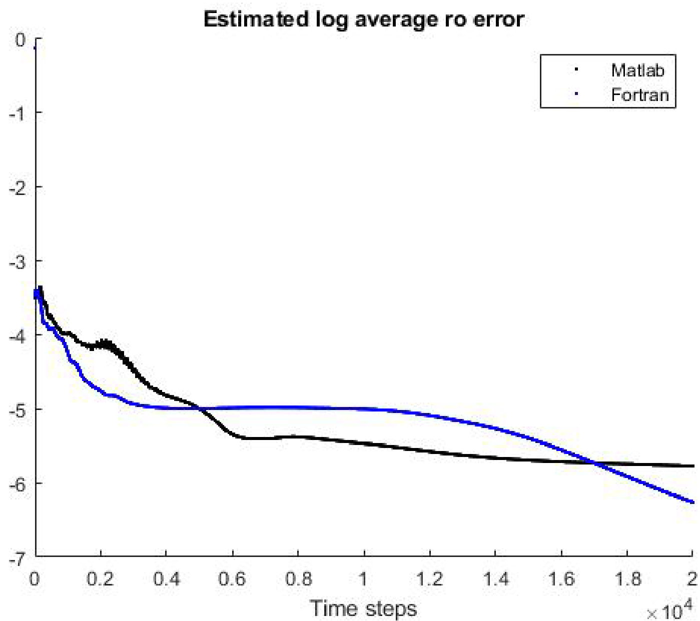

Figure 9: Detail of the graph in Figure 8. The logarithm of the estimate of the error in density averaged over the spatial grid. The graph is up to 20000 time steps. The blue line represents FORTRAN 77, while the black line represents MATLAB®. The horizontal line is the convergence criterion. The same pattern applies to the other variables.

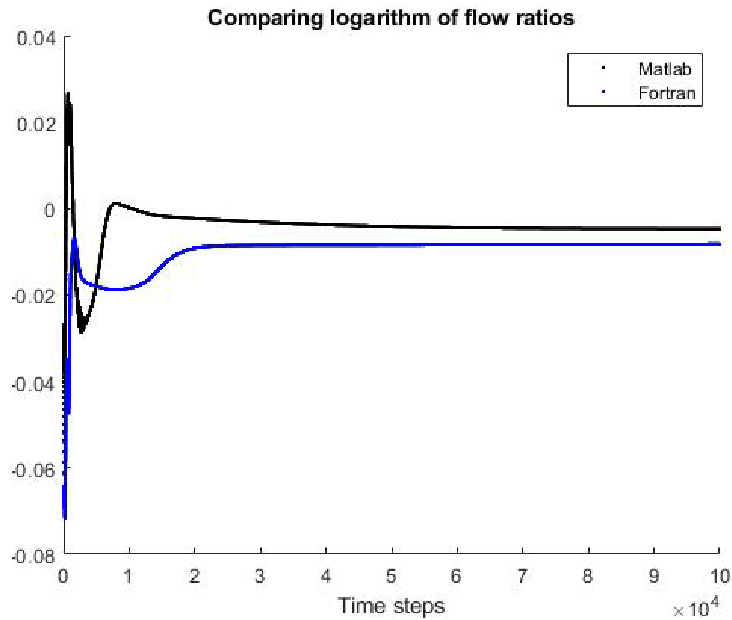

Figure 101: The logarithm of the ratio of inflow and outflow. The blue curve is computed by FORTRAN 77, and the black curve is computed by MATLAB®. In a steady state the logarithm of the flow ratio equals zero. Note that the MATLAB® solution is closer to achieving the desired flow ratio of one.

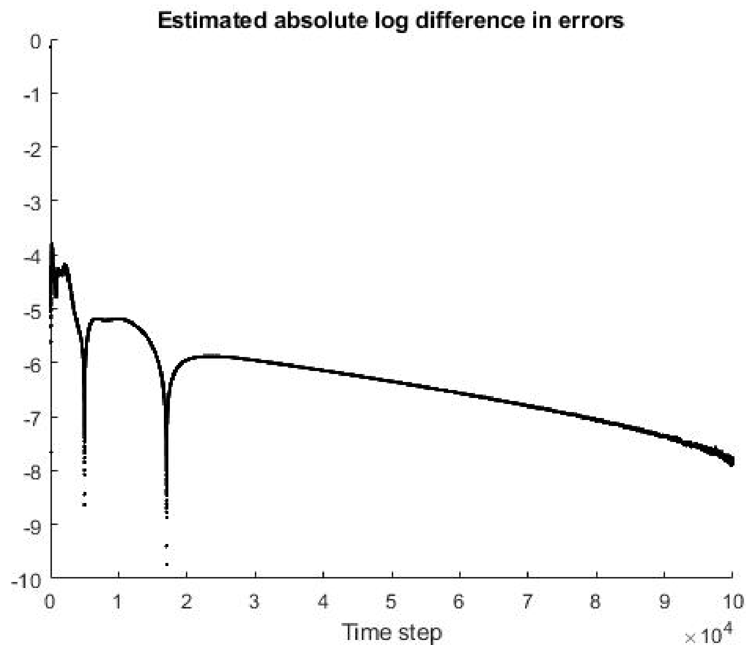

Figure 11: Logarithm of the absolute value of the difference in estimated error in density.

4. CONCLUSION

The compressible Euler equations has been solved by the same method implemented in FORTRAN 77 and MATLAB®, and even though the solutions look qualitatively similar some differences discovered. First of all, there was a 20% difference in final densities between the solvers. Second the path to convergence was different with the FORTRAN 77 version converging in one third of the time steps the MATLAB® version needed to reach the same level of confidence. Third the inlet to outlet flow ratio differed. The MATLAB® version was closer to one. The reasons for the discrepancies between the FORTRAN 77 and MATLAB® solvers are unclear. One possibility is that FORTRAN 77 use 32-bit precision to represent numbers and MATLAB® use 64-bit precision. No claim is made of the relevance of the computations to actual fluid flow, rather the key takeaway being that two finite and deterministic computations of the same algorithm on the same input in FORTRAN 77 and MATLAB® produce different output.

ACKNOWLEDGEMENTS

The authors have used part of the FORTRAN 77 code provided by Prof. William Dawes and Prof. Paul Tucker during the course CFD-4A2 – Computational Fluid Dynamics at Cambridge University Engineering Department.

REFERENCES

[1] Press, W.H. and W.T. Vetterling, Numerical Recipes in FORTRAN: The Art of Scientific Computing. 1992: Cambridge University Press.

[2] MATLAB®. 2015, The MathWorks Inc.: Natick, Massachusetts.

[3] Rose, L.D. and D. Padua, Techniques for the translation of MATLAB programs into Fortran 90. ACM Trans. Program. Lang. Syst., 1999. 21(2): p. 286-323.

[4] Khawaja, H. and S. Scott, CFD-DEM Simulation of Propagation of Sound Waves in Fluid Particles Fluidised Medium. The International Journal of Multiphysics, 2011. 5(1): p. 47-60.

[5] Khawaja, H.A., CFD-DEM and Experimental Study of Bubbling in a Fluidized Bed. The Journal of Computational Multiphase Flows, 2015. 7(4): p. 227-240.

[6] Khawaja, H.A., Sound waves in fluidized bed using CFD–DEM simulations. Particuology, 2018. 38: p. 126-133.

[7] Khawaja, H. and M. Moatamedi, Semi-Implicit Method for Pressure-Linked Equations (SIMPLE) – solution in MATLAB®. 2018, 2018. 12(4).

[8] Batchelor, G.K., An Introduction to Fluid Dynamics. Cambridge Mathematical Library. 2000, Cambridge: Cambridge University Press.

[9] Warburton, R., Object-oriented Vs. Functional Programming: Bridging the Divide Between Opposing Paradigms. 2015: O'Reilly Media.

[10] Runge, C., Ueber die numerische Auflösung von Differentialgleichungen. Mathematische Annalen, 1895. 46(2): p. 167-178.

[11] Butcher, J.C., A stability property of implicit Runge-Kutta methods. BIT Numerical Mathematics, 1975. 15(4): p. 358-361.

[12] Courant, R., K. Friedrichs, and H. Lewy, Über die partiellen Differenzengleichungen der mathematischen Physik. Mathematische Annalen, 1928. 100(1): p. 32-74.

[13] Page, C.G., The Professional Programmers Guide to Fortran 77. 1988: Pitman.

[14] The MathWorks, Inc. [cited 2019; Available from: https://se.mathworks.com/help/symbolic/digits html.