OpenFOAM® Journal

Volume 2 [Full Papers], Pages 62–80
ISSN: 2753-8168

IMPLEMENTATION OF A MODIFIED LID DRIVEN CAVITY IN OpenFOAM®

G.R.TABOR

CEMPS, University of Exeter, Harrison Building, North Park Road, Exeter EX4 4QF
Email address: g.r.tabor@ex.ac.uk

DOI: 10.51560/ofj.v2.54
Version(s): OpenFOAM® 8, OpenFOAM® 2106
Repo: https://github.com/gavintabor/MLDC

Abstract. The standard lid-driven cavity test case is one of the most used validation cases in CFD. Whilst comparisons with experimental and particularly DNS simulations are possible, there is no analytical solution, and the case is ill-posed when considering the boundary conditions. A modified lid driven cavity (MLDC) case exists in the literature [1] in which the lid velocity is non-uniform and which introduces a spatially varying body force, and for which there is a closed-form analytical solution to the Navier-Stokes equations which is a function of the Reynolds number. In this paper I present an implementation of the MLDC as a modification of the standard OpenFOAM case, using run time coding for the boundary conditions and `fvOptions`, and show how convergence to the solution is affected by numerical parameters of `simpleFoam` such as choice of matrix inversion. The existance of an analytical solution also allows the investigation of the relation between the solver residual and the true solution error.

1. Introduction

Canonical test cases such as the backward facing step [2] or flow around a square prism [3] occupy an important position in CFD. They provide flow conditions which are reasonably simple to set up (particularly in terms of the geometry), which illustrate particular types of flow, such as shear or recirculation, and for which there is known data, typically experimental data, to compare against. One of the most classic canonical cases is the lid driven cavity [4, 5, 6, 7], which generates a large primary vortex in the middle of the domain. In the simplest case, which is of course available in the OpenFOAM [8] tutorials, this is a 2d problem solved on a square domain (typically of unit side), with one moving side, usually the top side, and three stationary walls. Depending on the Reynolds number (based on lid speed and box dimension), this exhibits a range of laminar and turbulent flow, for which there is experimental data available. Given the relative simplicity of the domain, this has also been used extensively for DNS simulation, and this also provides significant validation resources [9].

More complex versions and extensions of the case have also been explored, often to address the conceptual challenges of the simple setup. Extending the case into 3d introduces a variety of additional behaviour [10] including potential periodicity in the third dimension. The behaviour of the flow in the corners of the domain can also be investigated [11]. Under the correct circumstances, the flow exhibits small secondary vortices in the corners which are not necessarily picked up by lower fidelity simulations, and some authors have explored this for example using triangular domains [12, 13].

The challenge of the corner flow is made more complex by a fundamental contradiction in the simple case, namely what is the motion of the corner vertices of the domain? For the corners between the moving and stationary walls, this is ambiguous; if they count as part of the moving wall then they share that velocity in the direction of movement of the wall, but if they are thought of as part of the stationary walls then the velocity is zero. This can be resolved by using a varying lid velocity which drops to zero at the corners. This was introduced by Shih *et al* [1], with a formulation which also provides for a closed form analytical solution. Whilst complex, this has obvious benefits as a test case, as recognised by Marchi *et al* [14]. Implementing this involves creating a new boundary (Dirichlet condition but spatially varying) and introducing a complex body force into the momentum equation. This can be done by extensively modifying the existing solvers [15], but this of course creates a new solver that has to be maintained. OpenFOAM's capacity for run time coding in input files, function objects and `fvOptions` provides an

* Corresponding author
Received: 3 November 2021, Accepted: 28 February 2022, Published: 15 March 2022

alternative approach of implementing the necessary coding in the case directory, and this is the subject of this paper. The existence of an actual analytical solution to the Navier-Stokes equations makes this an interesting validation test case in its own right, but also allows us to investigate the relationship between the solution residual typically used in CFD (which measures the numerical imbalance of the solved equations) and the actual solution error, for example defined as the integrated error over the entire domain.

1.1. **Modified Equations.** Shih *et al* [1] work in dimensionless variables, for which the Navier Stokes equations take the form

$$\nabla \bullet \mathbf{U}^* = 0 \tag{1}$$

$$\mathbf{U}^*.\nabla \mathbf{U}^* = \frac{1}{Re}\nabla^2 \mathbf{U}^* - \nabla p^* - \mathbf{j}B(x_*, y_*, Re) \tag{2}$$

where $x_* = x/L_0$ and L_0 is the box dimension. This includes an additional body force B oriented in the $\mathbf{j}$ direction. On the top surface the wall velocity is

$$U_x^* = 16\left(x_*^4 - 2x_*^3 + x_*^2\right) \tag{3}$$

The body force B has the form

$$\begin{aligned} B(x_*, y_*, Re) = &- \frac{8}{Re}\left[24F(x_*) + 2f'(x_*)g''(x_*) + f'''(x_*)g(y_*)\right] \\ &- 64\left[F_2(x_*)G_1(y_*) - g(y_*)g'(y_*)F_1(x_*)\right] \end{aligned} \tag{4}$$

where

$$f(x_*) = x_*^4 - 2x_*^3 + x_*^2 \tag{5}$$

$$g(y_*) = y_*^4 - y_*^2 \tag{6}$$

$$F(x_*) = \int f(x_*)dx_* = \frac{x_*^5}{5} - \frac{x_*^4}{2} + \frac{x_*^3}{3} \tag{7}$$

$$F_1(x_*) = f(x_*)f''(x_*) - \frac{1}{2}\left[f'(x_*)\right]^2 = -4x_*^6 + 12x_*^5 - 14x_*^4 + 8x_*^3 - 2x_*^2 \tag{8}$$

$$F_2(x_*) = \int f(x_*)f'(x_*)dx_* = \frac{1}{2}\left[f(x_*)\right]^2 \tag{9}$$

$$G(y_*) = g(y_*)g'''(y_*) - g'(y_*)g''(y_*) = -24y_*^5 + 8y_*^3 - 4y_* \tag{10}$$

Here the primes represent derivatives with respect to x_* and y_*. The exact solution to this problem is given by

$$U_x^*(x_*, y_*) = 8f(x_*)g'(y_*) = 8\left(x_*^4 - 2x_*^3 + x_*^2\right)\left(4y_*^3 - 2y_*\right) \tag{11}$$

$$U_y^*(x_*, y_*) = -8f'(x_*)g(y_*) = 8\left(4x_*^3 - 6x_*^2 + 2x_*\right)\left(y_*^4 - y_*^2\right) \tag{12}$$

$$\begin{aligned} p^*(x_*, y_*, Re) = &\frac{8}{Re}\left[F(x_*)g'''(y_*) + f'(x_*)g'(y_*)\right] \\ &+ 64F_2(x_*)\left[g(y_*)g''(y_*) - \left[g'(y_*)\right]^2\right] \end{aligned} \tag{13}$$

Shi et al provide the information that $B(0.5, 0.5, 1) = -3.356250$ as an additional check on the solution.

2. Implementation in OpenFOAM

2.1. **Dimensional equations.** Implementation of these equations in OpenFOAM requires that they first be converted back to dimensional quantities, which can be achieved by multiplying through by U_0^2/L_0 (dimension $[LT^{-2}]$), where U_0 is the maximum lid velocity. In particular this gives

$$U_x = 16U_0\left(x_*^4 - 2x_*^3 + x_*^2\right) \tag{14}$$

$$\begin{aligned} B(x, y, Re) = &- \frac{8}{Re}\left[24F(x_*) + 2f'(x_*)g''(x_*) + f'''(x_*)g(y_*)\right] \\ &- 64\left[F_2(x_*)G_1(y_*) - g(y_*)g'(y_*)F_1(x_*)\right] \end{aligned} \tag{15}$$

together with the solution

$$U_x(x,y) = 8U_0\left(x_*^4 - 2x_*^3 + x_*^2\right)\left(4y_*^3 - 2y_*\right) \tag{16}$$

$$U_y(x,y) = 8U_0\left(4x_*^3 - 6x_*^2 + 2x_*\right)\left(y_*^4 - y_*^2\right) \tag{17}$$

$$p(x,y,Re) = \frac{8U_0^2}{Re}\left[F(x_*)g'''(y_*) + f'(x_*)g'(y_*)\right] + 64F_2(x_*)\left[g(y_*)g''(y_*) - \left[g'(y_*)\right]^2\right] \tag{18}$$

For notational convenience, the non-dimensional coordinates x_*, y_* have been kept here, however the actual coordinates x, y must first be converted to non-dimensional form when evaluating these expressions. After this, the equations (14) – (18) can be implemented as follows :

- Equation (14) can be implemented as a coded boundary condition in the velocity field. The coding for this is reproduced in Table 1.
- Equation (15) can be implemented as a coded `fvOption`. The coding for this is reproduced in Table 3.
- The solution (equations (16) – (18)) can be implemented and compared with the computed solution through a user coded `functionObject`. This is discussed in section 2.5.

Note that the evaluation of the B term and of the overall analytical solution involves evaluation of a number of derivative terms which are common to both sets of equations, so the coding of these functions is placed in a separate file `shiEqn.H` which can be included into the `fvOptions` and `functionObjects` definitions as required. Evaluation of these quantities like this involves repeated creation and destruction of the appropriate `geometricField` objects, which is hardly efficient, but this is not an important consideration here.

2.2. **Force on the lid.** In fact we can take the analysis further and evaluate the force on the moving boundary. Taking the derivative of equation (16)

$$\begin{aligned}\frac{\partial U_x}{\partial y} &= \frac{\partial y_*}{\partial y}\frac{\partial U_x}{\partial y_*} \\ &= \frac{8U_0}{L_0}\left(x_*^4 - 2x_*^3 + x_*^2\right)\left(12y_*^2 - 2\right) \\ &= \frac{80U_0}{L_0}\left(x_*^4 - 2x_*^3 + x_*^2\right)\end{aligned} \tag{19}$$

evaluated at $y_* = 1$ which is the moving boundary. Since

$$\frac{F}{A} = \mu\frac{\partial U_x}{\partial y} \tag{20}$$

for an elemental area on the moving lid $A = dx \times \delta z$, where δz is the cell thickness in the mesh, this gives

$$\begin{aligned}F &= \mu\delta z\int_0^{L_0}\frac{\partial U_x}{\partial y}dx = \mu\delta z\int_0^1\frac{\partial U_x}{\partial y}\frac{\partial x}{\partial x_*}dx_* \\ &= 80U_0\mu\delta z\int_0^1\left(x_*^4 - 2x_*^3 + x_*^2\right)dx_* \\ &= \frac{8}{3}U_0\mu\delta z\end{aligned} \tag{21}$$

which can be compared with the force evaluated from the `functionObject forces`; with the `rho` entry used to set the fluid density. Alternatively we can rearrange this to formulate a dimensionless force coefficient by dividing through by $\frac{1}{2}\rho U_0^2$, giving

$$C_{MLDC} = \frac{16}{3}\frac{1}{Re} \tag{22}$$

For numerical convenience, U_0 was chosen as 3 m/s, giving theoretical forces as decimals of 8, and the viscosity varied to give $Re = 3, 30, 300$ as the test cases.

2.3. **Coded boundary.** The code section in Listing 1 calculates the position-dependent speed of the moving lid. Parameters of the MLDC (`boxSide` $\equiv L_0$ and `boxVel` $\equiv U_0$) are provided in the `transportProperties` dictionary. Since dimension checking is provided at the `geometricField` level not at the level of boundary fields, the values but not the dimensions of these quantities are needed. Note that the `boxSide` parameter is linked to the geometry of the domain, so if this parameter is changed (in the separate file `shiEqn.H`) then `blockMesh` needs to be rerun on the case.

```
movingWall
{
    type            codedFixedValue;
    value           uniform (1 0 0);
    name            variedLid;
    code
    #{
        const fvPatch& boundaryPatch = patch();
        const vectorField& Cf = boundaryPatch.Cf();
        vectorField& field = *this;

        dimensionedScalar boxSide
        (
            "boxSide",
            dimLength,this->db().lookupObject<IOdictionary>
            ("transportProperties").lookup("boxSide")
        );

        dimensionedScalar boxVel
        (
            "boxvel",
            dimVelocity,this->db().lookupObject<IOdictionary>
            ("transportProperties").lookup("boxVel")
        );


        forAll(Cf, faceI)
        {
            const scalar x = Cf[faceI].x()/boxSide.value();
            const scalar bValue = 16.0*boxVel.value()
                *(sqr(x) - 2.0*pow(x,3.0) + pow(x,4.0));
            field[faceI] = vector(bValue,0,0);
        }
    #};
}
```

Listing 1. Coded boundary condition implementing the moving lid.

2.4. **Source Term.** Listing (3) provides the additional source term Equation (15) in the momentum equation. The majority of the code is actually coded in a separate file `shiCalc.H` as it needs to be shared with the `functionObject` `trueSoln` (section 2.5); `codeOptions` being set appropriately so that the file is available at compilation. Listing (5) presents the implementation of equations (5) - (10), with the Reynolds number being calculated and output.

2.5. **Calculation of the solution.** Calculation of the solution (equations (16) - (18) is provided as a user coded `functionObject` called `trueSoln`. This has two purposes :

(1) to calculate the true field solutions for p and $\mathbf{U}$, and
(2) to calculate the error in the calculated fields, defined as the cell weighted average of the error term normalised by the cell weighted average of the true solution, i.e.

$$p_{err} = \frac{\sum_{cell} p_i - p_{Theor,i}}{\sum_{cell} p_{Theor,i}} \tag{23}$$

$$U_{err} = \frac{\sqrt{\sum_{cell} \left(U_i - U_{Theor,i}\right)^2}}{\sum_{cell} U_{Theor,i}} \tag{24}$$

with $\sum_{cell}$ denoting a cell weighted summation – the mesh is currently homogeneous but this could potentially be altered to investigate non-uniform meshes.

```
codedSource
{
    type            vectorCodedSource;
    selectionMode   all;
    active          on;
    redirectType    velocitySource;

    fields          (U);
    name            B;

    codeOptions
    #{
        -I$(FOAM_CASE)/system
    #};

    codeAddSup
    #{
        vectorField& BSource = eqn.source();
        const scalarField& V = mesh().V();

        #include "shiCalc.H"

        BSource = V*(b_y.internalField())*vector(0,1,0);
    #};
}
```

LISTING 2. `fvOptions` code implementing the additional source term.

The local error values are calculated as part of this and are available as `volScalarFields`;

$$p_{err,i} = \frac{p_i - p_{Theor,i}}{\sum_{cell} p_{Theor,i}} \tag{25}$$

$$U_{err,i} = \frac{\sqrt{(U_i - U_{Theor,i})^2}}{\sum_{cell} U_{Theor,i}} \tag{26}$$

As will be seen later, the one problem with this normalisation is that the pressure field ranges from negative to positive, and so the denominator in equation 23 and 25 may be quite small (and sensitive to other aspects of the solution), giving an unreasonably large p_{err} normalisation.

2.6. **Numerical Parameters.** OpenFOAM, being an implicit CFD code, uses matrix inversion techniques to solve the individual equations. Depending on the exact equation being solved, and significantly its mathematical properties (such as symmetry/asymmetry), different inversion/solver algorithms are used, particularly between the pressure equation and the other equations. In OpenFOAM, the pressure equation can be solved either using a variant of Algebraic Multigrid (`GAMG`) or the preconditioned Conjugate Gradient solver (`PCG`), whilst the velocity equation (and other transport equations) use either the direct solver `smoothSolver` or the preconditioned Biconjugate Gradient solver (`PBiCGStab`). In the installation tutorials, the pairing `GAMG`/`smoothSolver` is typically used (particularly for the existing Lid Driven Cavity cases) and is denoted set A in table 1, whilst the combination `PCG`/`PBiCGStab` is denoted set B. Table 1 also sets out the basic numerical parameters used, which are the common defaults (except in section 6 where the tolerances were tightened to ensure that the matrix continued to solve properly). Other numerical parameters and differencing schemes are as standard from the tutorial files; the intention here is to illustrate the relationship between matrix residual and solution error for the commonly used solver settings, not to perform an in-depth analysis of the matrix methods themselves.

3. RESULTS

3.1. **Spatial results.** To demonstrate the nature of the actual solution, the results for the p and U fields are presented in figure 1 for pressure (top) and velocity magnitude (bottom). Plots a,c are the computed solution and b, d the theoretical values from equations (16) – (18), for the $Re = 3$ case. The standard matrix solvers (Case A, `GAMG` and `smoothSolver`) were used. The calculations were carried out on a 160×160 resolution domain, and cell centred results are used for the visualisation in order to avoid further interpolation in `paraview`. As can be seen, the fields are indistinguishable visually at this level.

```
scalar Re = readScalar(mesh().lookupObject<IOdictionary>
            ("transportProperties").lookup("Re"));

dimensionedScalar boxSide
(
    "boxSide",
    dimLength,mesh().lookupObject<IOdictionary>
    ("transportProperties").lookup("boxSide")
);

dimensionedScalar boxVel
(
    "boxvel",
    dimVelocity,mesh().lookupObject<IOdictionary>
    ("transportProperties").lookup("boxVel")
);

Info << "Re = " << Re << endl;
volScalarField x = mesh().C().component(0)/boxSide;
volScalarField y = mesh().C().component(1)/boxSide;

volScalarField f = pow(x,4.0)-2.0*pow(x,3.0)+pow(x,2.0);
volScalarField g = pow(y,4.0) - pow(y,2.0);

volScalarField F=0.2*pow(x,5.0)-0.5*pow(x,4.0)+pow(x,3.0)/3.0;
volScalarField F1=-4.0*pow(x,6.0)+12.0*pow(x,5.0)-14.0*pow(x,4.0)
                        +8.0*pow(x,3.0)-2.0*x*x;

volScalarField F2=0.5*f*f;
volScalarField G1=-24.0*pow(y,5.0)+8.0*pow(y,3.0)-4.0*y;

volScalarField fp=4.0*pow(x,3.0)-6.0*x*x + 2.0*x;
volScalarField fppp = 24.0*x-12.0;
volScalarField gp=4.0*y*y*y-2.0*y;
volScalarField gpp=12.0*y*y-2.0;
volScalarField gppp=24.0*y;

volScalarField b_y
(
    IOobject
    (
        "b_y",
                mesh().time().timeName(),
                mesh(),
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
    ),
    ((-8.0/Re)*(24.0*F+2.0*fp*gpp+fppp*g)-64.0*(F2*G1-g*gp*F1))
        *boxVel*boxVel/boxSide
);
```

LISTING 3. File `shiCalc.H` implementing the derivative functions.

Similar plots can be produced for the other Reynolds number cases, but have not been included to save space.

To identify actual discrepancies between the calculated and theoretical fields, error fields can be evaluated, normalised by the cell weighted values. This has been done and the calculated fields and error fields for pressure and velocity magnitude are displayed in figures 2 and 3 respectively. Note that since the range of p goes from negative to positive, the average value is considerably smaller than the extremum values, so this normalisation in fact exaggerates the level of the error.

```
TrueSoln
{
    functionObjectLibs ("libutilityFunctionObjects.so");
    type coded;
    name TrueSoln;
    writeControl outputTime;
    writeInterval 1;

    codeOptions
    #{
        -I$(FOAM_CASE)/system
    #};

    codeExecute
    #{
        #include "shiCalc.H"

        volScalarField pTheor
        (
            IOobject
            (
                "pTheor",
                mesh().time().timeName(),
                mesh(),
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            ((8.0/Re)*(F*gppp+fp*gp)+64.0*F2*(g*gpp-gp*gp))
            *dimensionedScalar("one",
                    dimensionSet(0,2,-2,0,0,0,0),1.0)
        );

        volVectorField UTheor
        (
            IOobject
            (
                "UTheor",
                mesh().time().timeName(),
                mesh(),
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            mesh().lookupObject<volVectorField>("U")
        );

        volScalarField ux = 8.0*f*gp*boxVel.value();
        volScalarField uy = -8.0*fp*g*boxVel.value();

        UTheor.replace(0,ux);
        UTheor.replace(1,uy);
```

LISTING 4. First part of `functionObject` evaluating the true solution.

3.2. **Forces on lid.** Evolution of the lid force and correlations with the matrix residuals are presented in figures 4 and 5 for the standard Multigrid solvers (A) and for the Conjugate Gradient solvers (B) respectively.

Figure 4.a. demonstrates the (partial) convergence of the lid force against iteration number for the standard solver configuration for the three different Reynolds numbers under investigation. To allow for

```
        scalar pWeight = pTheor.weightedAverage(mesh().V())
                        .value();
        scalar UWeight = magSqr(UTheor)()
                        .weightedAverage(mesh().V()).value();

        const volVectorField& U = mesh().lookupObject
                                    <volVectorField>("U");
        const volScalarField& p = mesh().lookupObject
                                    <volScalarField>("p");

        volScalarField pErr
        (
            IOobject
            (
                "pErr",
                mesh().time().timeName(),
                mesh(),
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            mag(p-pTheor)/pWeight
        );

        volScalarField UErr
        (
            IOobject
            (
                "UErr",
                mesh().time().timeName(),
                mesh(),
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            magSqr(U-UTheor)/UWeight
        );

        Info << "Errors "
             << mesh().time().timeName()
             << "   "
             << pErr.weightedAverage(mesh().V()).value()
             << "   "
             << UErr.weightedAverage(mesh().V()).value()
             << endl;

        if (mesh().time().writeTime())
        {
            pTheor.write();
            UTheor.write();
            pErr.write();
            UErr.write();
        }
    #};
}
```

Listing 5. Second part of `functionObject` evaluating the solution error.

TABLE 1. Matrix solvers used.

Case		p	U
A	Solver	GAMG	smoothSolver
	Smoother	GaussSeidel	symGaussSeidel
	Tolerance	10^{-6}	10^{-5}
B	Solver	PCG	PBiCGStab
	Preconditioner	DIC	DILU
	Tolerance	10^{-8}	10^{-7}

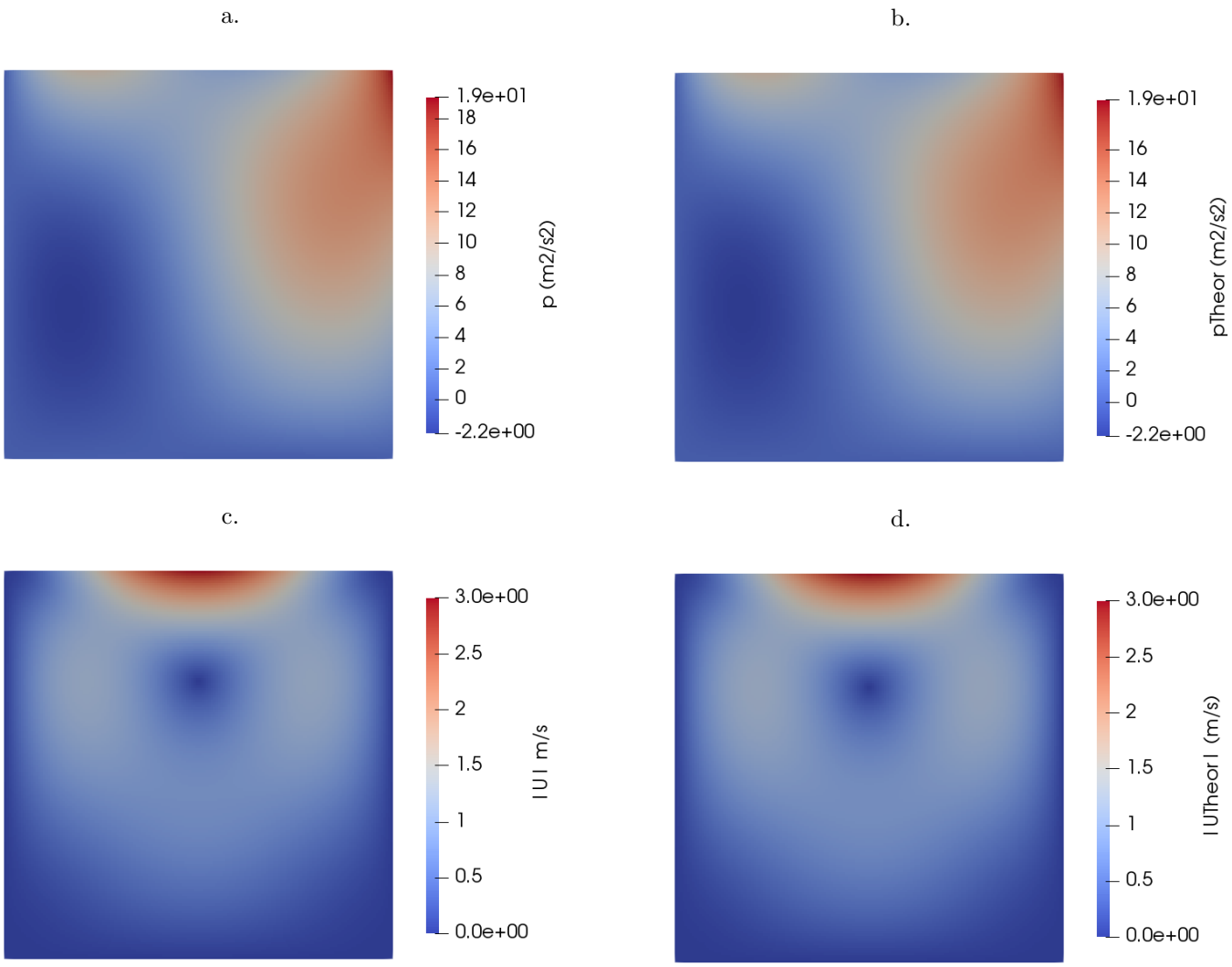


FIGURE 1. Comparison between computed fields (a,c) and theoretical values from equations (16) – (18) for pressure (top) and velocity (bottom), for $Re = 3$ case.

easy comparison, the forces are normalised by the appropriate value of F_{theor} calculated from equation (2.2). Convergence for the cases $Re = 3, 30$ seems to follow very much the same pattern, however the calculated force for the $Re = 300$ case seems to overshoot before returning. The simulation terminates when the matrix residuals drop below 10^{-4}; it is fairly obvious from this graph that the forces are still evolving at this stage and a final solution has not been achieved.

Figure 4.b. shows the correlation between the solution errors for the dependent variables and the matrix residuals for the same variables. Here the solid lines give the correlation between the pressure error p_{err} (as defined by equation (23) and the initial pressure residual (`p_0` extracted from running `foamLog`). The dashed and dash-dot lines demonstrate the correlation between U_{err} (defined by equation (24) and the `Ux_0` and `Uy_0` residuals. As can be seen there are substantial differences between the

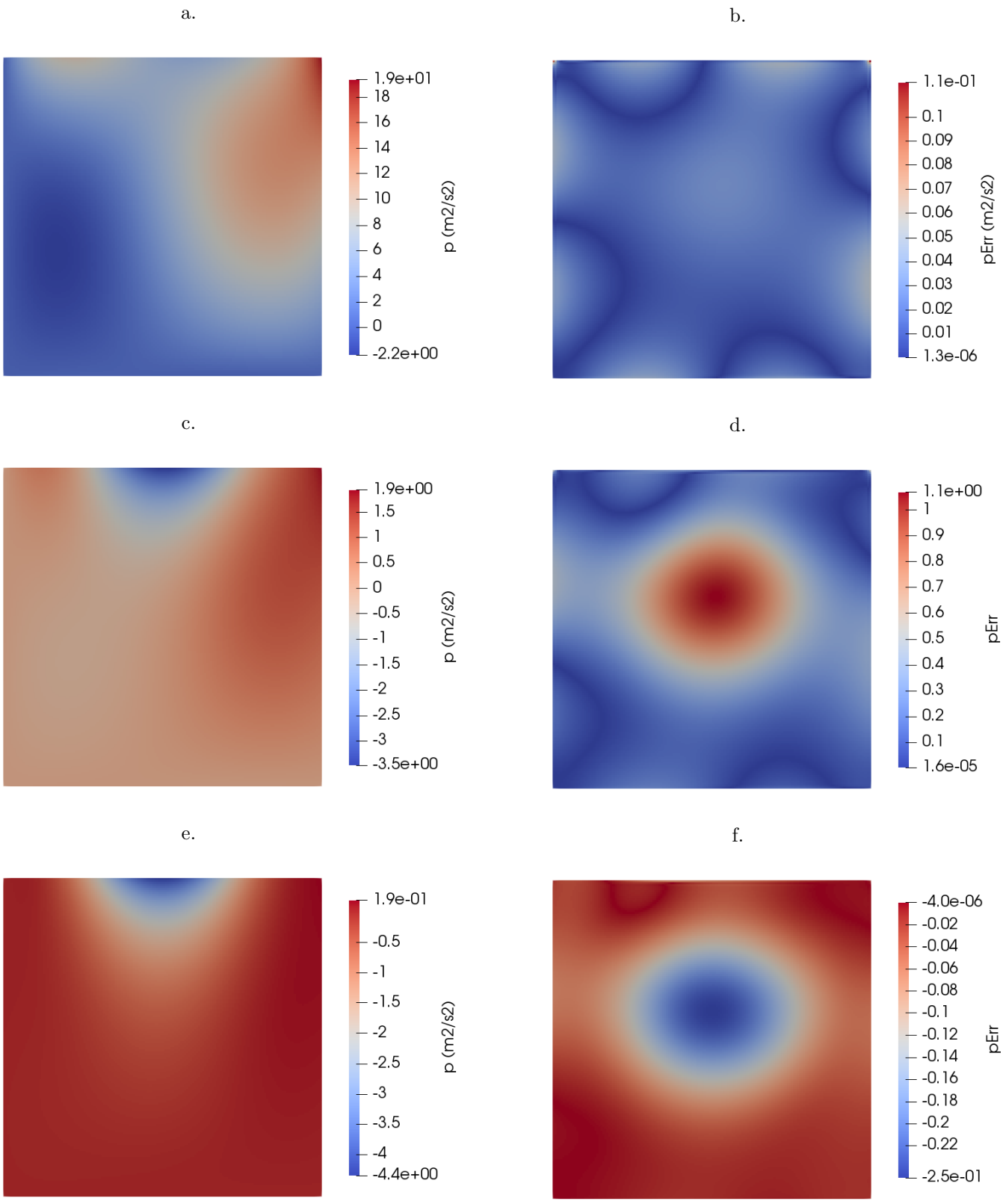


FIGURE 2. Pressure plots (left column) and pressure error plots (right column). 1st row : $Re = 3$, second row $Re = 30$, third row $Re = 300$.

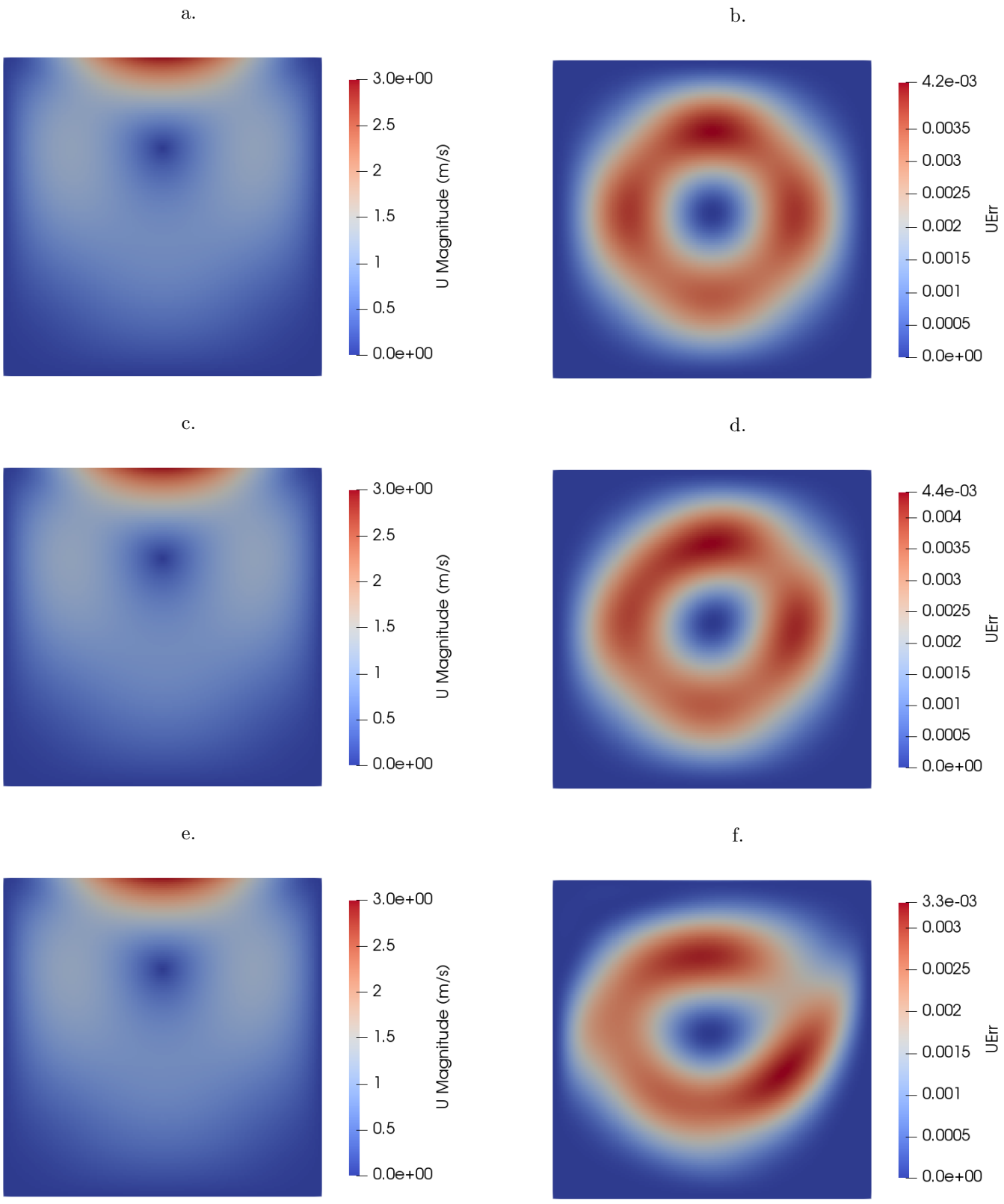


FIGURE 3. Velocity magnitude plots (left column) and velocity error plots (right column). 1st row : $Re = 3$, second row $Re = 30$, third row $Re = 300$.

pressure curves although some of this may be down to overall normalisation – the pressure field of course ranges from negative to positive so the denominator in equation 23 is possibly small. The other curves relating U_{err} to the U residuals all more or less follow the same path. What is interesting here is how the solution errors change as the calculation proceeds, i.e. as the matrix residuals reduce in magnitude. This corresponds to tracing the curves from right to left as the SIMPLE loop executes. As can be seen, initially the curves are quite flat; improving the matrix residual $1 \rightarrow 0.1 \rightarrow 0.01$ has comparatively little effect on the solution accuracy. The solid black line on the figure shows a 1:1 correlation between the Error and Residual, which would be ideal. The actual solution only really starts to substantially improve when the matrix residuals drop to $< 10^{-3}$, and as observed above, the errors are still reducing when the matrix residuals reach 10^{-4} and the algorithm terminates.

Figure 5 shows the same graphs for the Conjugate Gradient solution. Largely this shows the same behaviour. Convergence in this case is slightly quicker (in terms of number of iterations of the SIMPLE loop) but not significantly so, and roughly the same comments may be made about the relationship between matrix residuals and the solution errors.

3.3. **Richardson Extrapolation.** Figure 6 (top) presents results for the different Reynolds numbers based on convergence of the force results. Instead of stopping the simulation when the matrix residuals drop below 10^{-4}, the simulation was run until the calculated lid force reached a stable value. This was achieved for the coarse meshes to 6 significant figures but required significantly more effort for the finer meshes ($N = 320$ meshes required 20,000 iterations to converge to achieve this). Tolerances on the matrix solvers were tightened to 1×10^{-9} to ensure that the matrices continued to solve throughout; otherwise the matrix combination A was used. Richardson extrapolation was then applied by fitting a quadratic curve to the data for each Reynolds number and then extrapolating to a zero reduced mesh spacing (the mesh spacing for $N = 320$ being 1.0) to find the infinite mesh resolution value for the force. Table 2 shows the zero value for the different Reynolds numbers, indicating an error in this parameter $< 0.1\%$.

The results can also be analysed to calculate the actual numerical error in the algorithm. As is well known, the error term for any numerical scheme can be written as

$$F - F_{theor} = Ah^c + \text{higher order terms} \ldots \tag{27}$$

where h is the cell dimension and c the order of the scheme. Since the standard settings (used here) for `simpleFoam` comprise a blend of 1st and 2nd order numerics we expect the effective value of c to lie $1 < c < 2$. In order to compare the values for all three Reynolds numbers we can divide this expression through by F_{theor} and evaluate

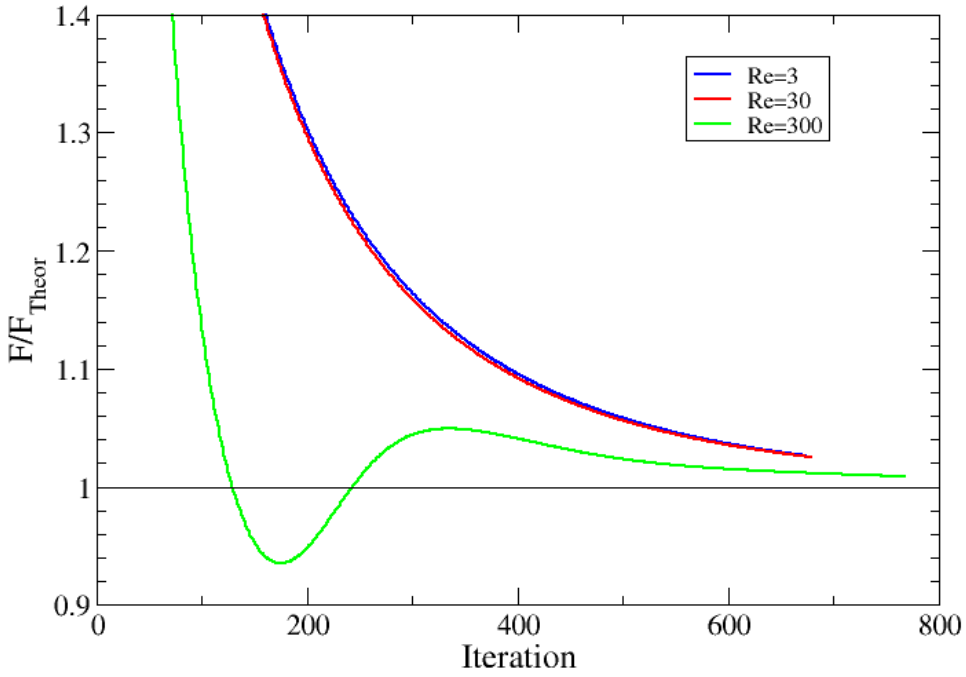
$$\text{Err}(F) = 1 - \frac{F}{F_{theor}} = A'h^c \ldots \tag{28}$$

Figure 6 (lower figure) presents the results for the three different Reynolds numbers together with curves demonstrating pure 1st and 2nd order schemes. Using non-linear curve fitting gives the values presented in table 2. For $Re = 300$ the order of the algorithm is 1.53. Values for the other two Reynolds numbers are lower but the fit is compromised by the downturn curve of the graphs on coarser meshes; neglecting the final point ($h = 0.05$ for a mesh spacing N=20) gave values of 1.47424 and 1.47925 for $Re = 3$ and $Re = 30$, respectively.

Figure 7 shows the evolution of both matrix residuals and solution errors against iteration for the three different Reynolds number cases. Judging by the solution errors the simulations have converged at 2500 iterations and no improvement in solution accuracy is possible, although the matrix residuals continue to decrease until they reach the set level of 10^{-9}. The error in U reaches a level of $10^{-7} - 10^{-8}$; however the error in force F and the pressure error remain orders of magnitude higher. Since it is the viscous force being evaluated (related to the velocity gradients) these two errors are not obviously related. Figure 8 shows the correlation between error and residual for different quantities; errors in force and pressure plotted against pressure residual (top graph) and error in U plotted against U_x, U_y residuals (bottom graph, A curves). Only the first 2500 iterations (up to convergence) are plotted. The straight lines give a 1:1 relationship for comparison. As before (section 3.2, figure 4), initial improvements in residual have little effect on the solution; residual convergence beyond 10^{-3} is more effective, whilst the final residual convergence beyond 10^{-6} is ineffectual again. The somewhat odd behaviour of the force error for the $Re = 300$ case is because the error goes negative for a time (the absolute error is being plotted here).

Since the force on the lid depends on the gradient of the velocity, figure 8 also shows the correlation between the error in F and the U_y and U_x residuals (solid and dashed lines, respectively, lower figure, B curves). As expected the correlations are somewhat smoother here; the lines for $Re = 3$ and $Re = 30$ are basically coincident, whilst those for $Re = 300$ diverge somewhat as the force error goes negative – again, the absolute error is being plotted (in order to use logarithmic axes).

a.



b.

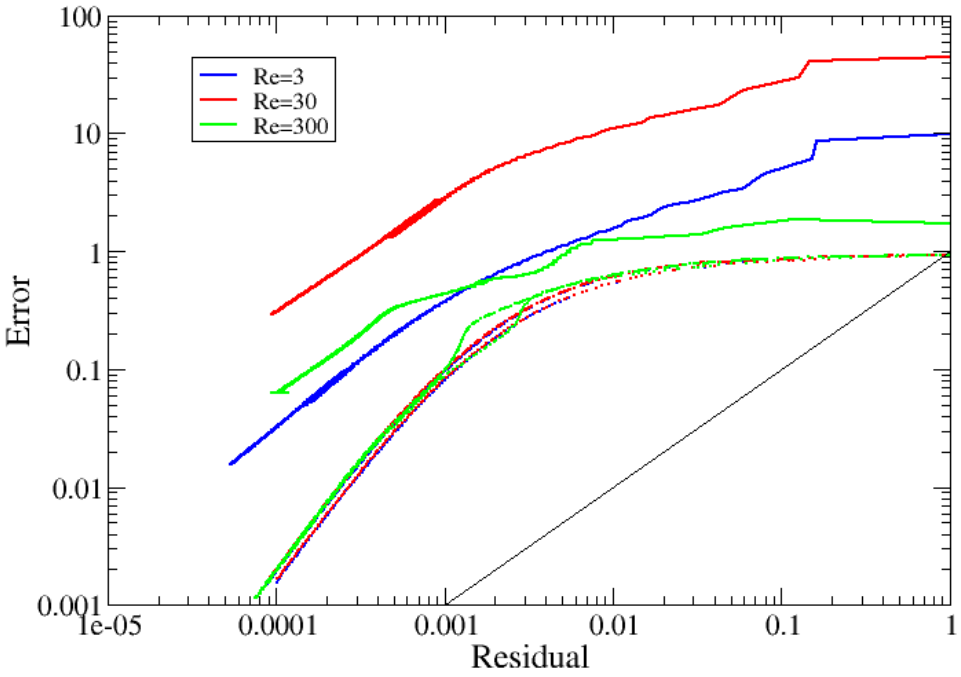
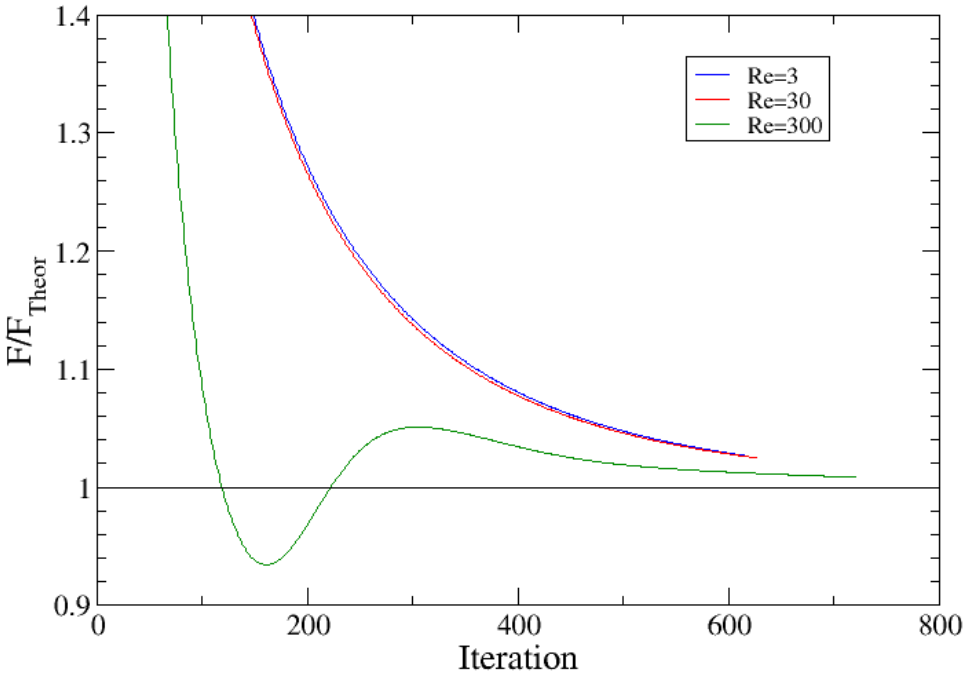


FIGURE 4. Errors in the force evaluation (a, top figure) and comparison between the solution error (defined by the weighted average of the error fields p_{err} and U_{err}, equations...) and the residuals for the pressure, U_x and U_y equations (b, bottom figure). In both figures, blue curves represent $Re = 3$, red $Re = 30$ and green $Re = 300$. In the lower graph, the solid lines represent the comparison between the pressure error p_{err} and the pressure equation residual, and the dashed and dotted lines represent the equivalent comparison between U_{err} and the U_x and U_y residuals respectively. The solid black line shows what a simple linear relation between error and residual would look like.

a.



b.

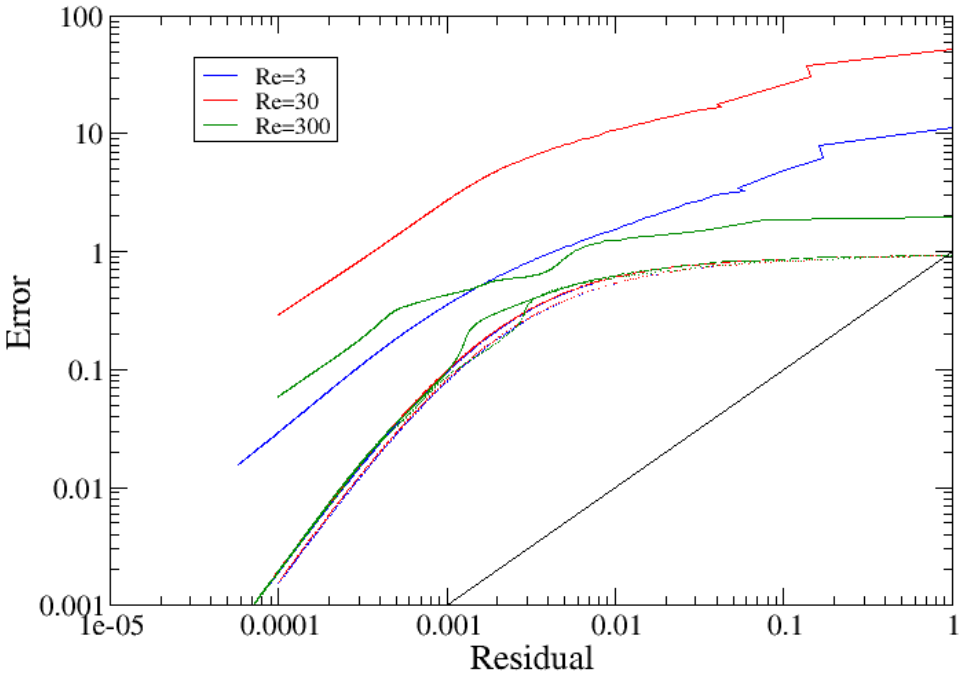


FIGURE 5. Errors in the force evaluation (a, top figure) and comparison between the solution error and the residuals for the pressure, U_x and U_y equations using Conjugate Gradient solvers (b. bottom figure). Details of the line styles are as given in figure 4.

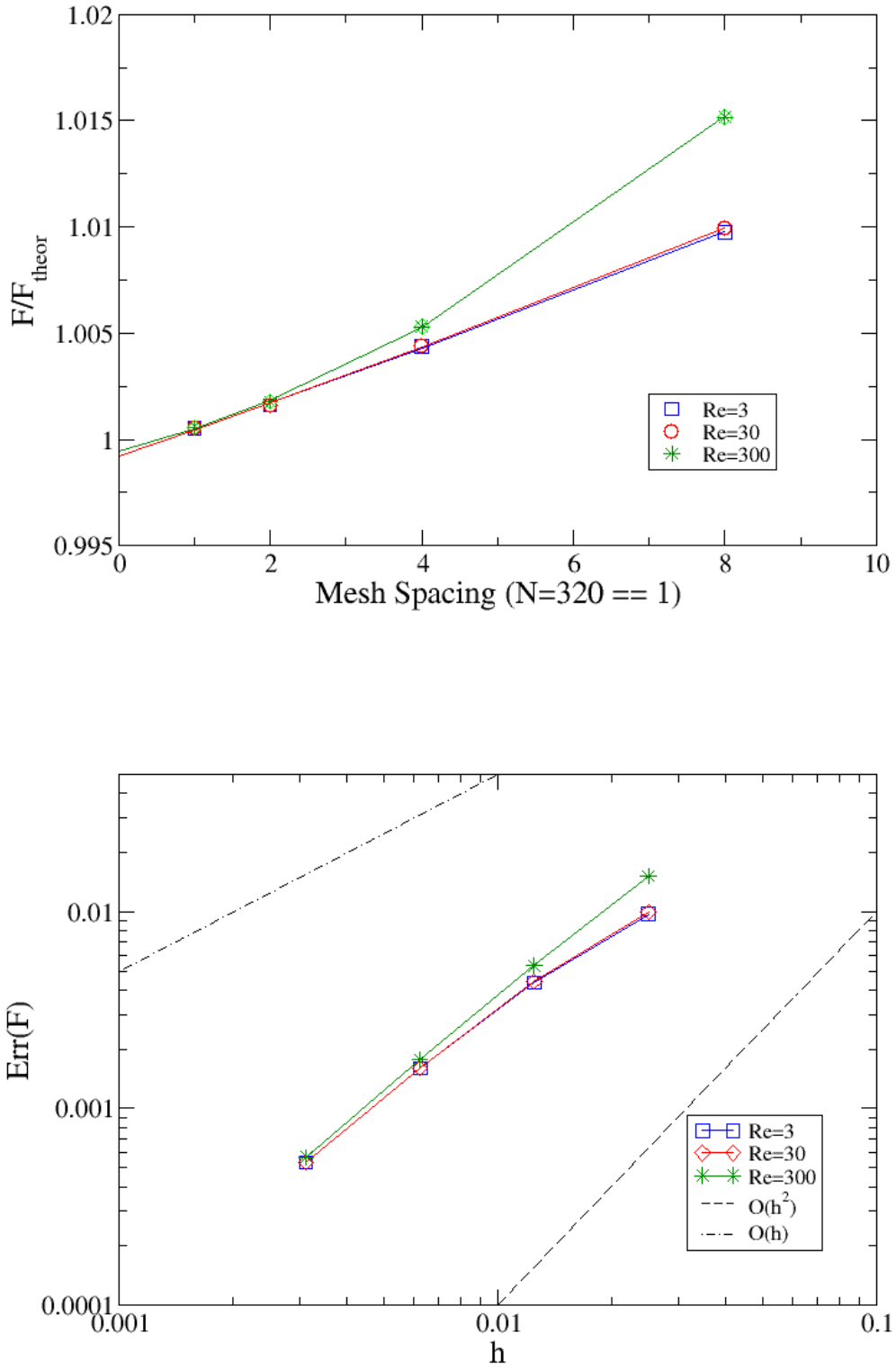


FIGURE 6. (Top) Richardson extrapolation on the meshes; plotting F/F_{theor} against mesh spacing index (taking the spacing for $N = 320$ as 1.0). The calculated points are shown as coloured symbols. The lines are quadratic fits to each dataset, enabling extrapolation to zero mesh spacing (infinite mesh resolution). (Bottom) Plot of error in the numerical solution against mesh spacing h. Also shown are lines denoting $O(h)$ and $O(h^2)$ convergence.

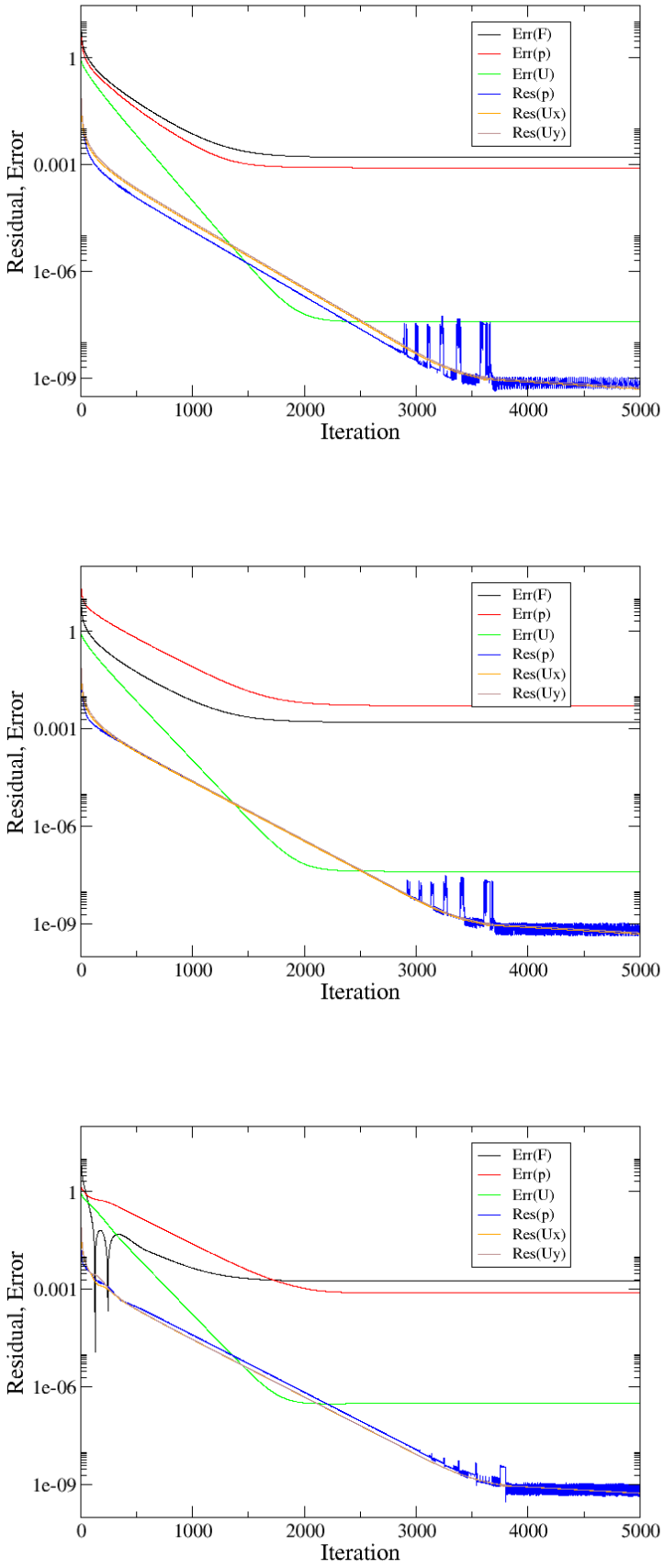
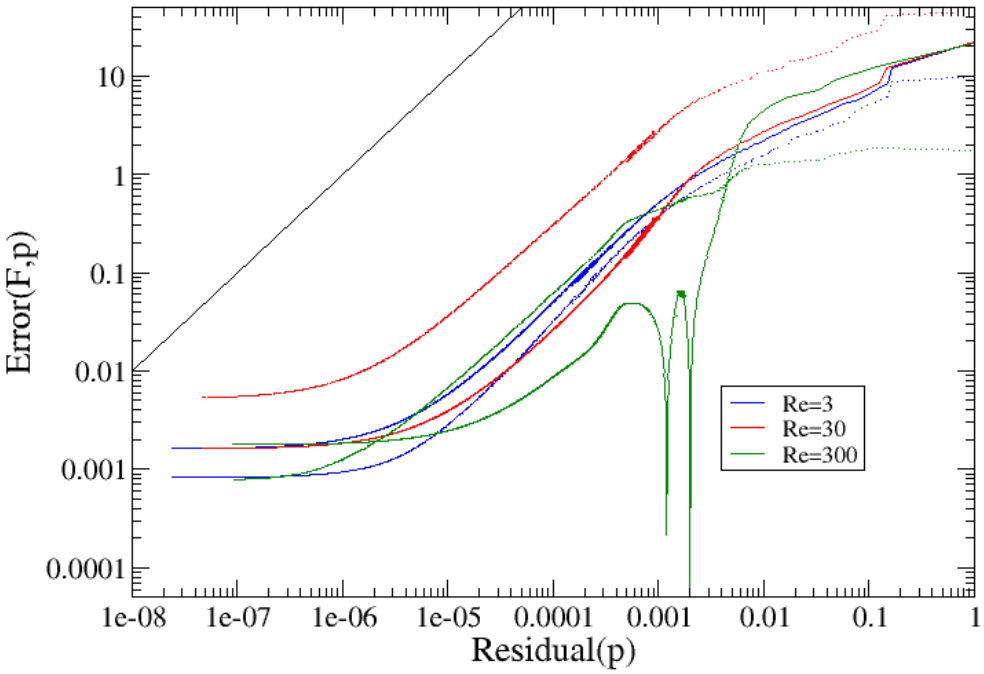


FIGURE 7. Convergence of errors and residuals over $5,000$ iterations. Top: $Re = 3$, Middle $Re = 30$, Bottom: $Re = 300$.



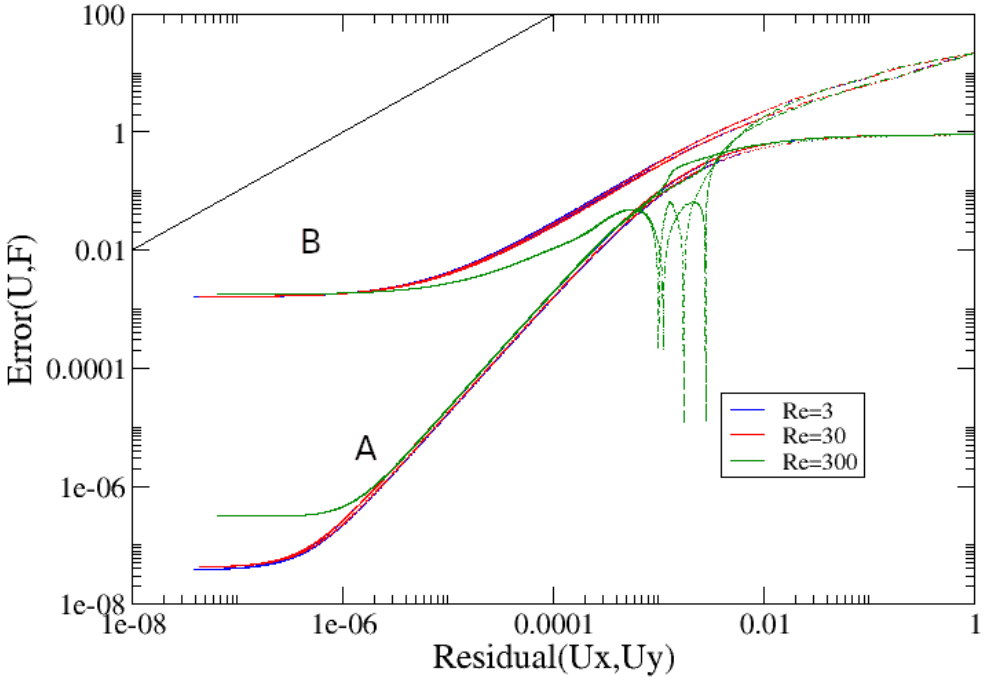


FIGURE 8. Solution errors vs. matrix residuals, complete simulation. Top: errors in force F (solid lines) and pressure p (dotted lines) against pressure residual. Bottom: Error in U (curves A) plotted against U_x, U_y residual (solid, dotted lines respectively). Error in F (curves B) is also plotted against U_x, U_y residual (dash-dot, dash-dash-dot).

TABLE 2. Values of F/F_{theor} from Richardson extrapolation for the different Reynolds numbers. Values are also given for the error coefficient c.

Re	F/F_{theor}	c
3	0.999219	1.24309
30	0.99923	1.25798
300	0.999451	1.53126

4. Conclusions

Validation and verification are important aspects of CFD. The distinction between these activities can be summed up as : verification is the act of checking that the mathematical equations have been validly implemented in the code, whilst validation is the act of checking that the solution matches mathematical or physical reality – that the mathematical model is itself correct. In both activities, canonical cases, which illustrate specific types of flow in simple geometries, hold a significant position. Because of the complex nature of the Navier Stokes equations, mathematical solutions are rarely available, so experimental or DNS numerical data has often been used. However where analytical results are possible they can provide a very high quality of comparison. Juretic [16] for example has demonstrated validation of OpenFOAM against algebraic solutions for a planar jet, as well as analytical solutions for simpler physics such as creeping flow and heat conduction.

This paper presents an implementation in OpenFOAM of a modified Lid Driven Cavity test case. The modifications, due to Shi *et al* [1, 14] include a non-uniform lid velocity (thus avoiding the ambiguity in the velocity at the corners of the domain) and a body force, and allow a closed form analytical solution. This has been extended to calculate the force on the lid, which provides a convenient physical and quantitative comparison between the solution and the numerical results. In addition to validating the classic `simpleFoam` solver for both multigrid and Conjugate Gradient solvers, this provides an excellent test case to examine the relation between the solution residual from the matrix solver and the actual mathematical error in the results. This shows the following :

- Monitoring residuals alone is not necessarily sufficient to identify convergence of a simulation. In particular even a relatively tight residual tolerance (of 10^{-4}) does not guarantee convergence of physical properties such as boundary forces.
- In general the relationship between the matrix residuals and the physical solution error is monotonic but not linear. Initial reductions in residual may not be accompanied by equivalent reductions in error. Where there is a mathematical link between the quantity being solved and the physical parameter being monitored (so between the velocity field and the force which is a function of the gradient of that field) there may be a closer relationship between the residual and the error.
- Fully converging the forces and applying Richardson extrapolation across a range of mesh resolutions, the error in the solution can be reduced to less than 0.1%, with a numerical error order of around 1.5, indicating a combination of 1st and 2nd order schemes. Further tuning of the numerical parameters might improve this, and this could be a valuable test case for comparing the actual effects of various numerical schemes (both differencing schemes and matrix solvers).

Author Contributions: Conceptualisation, G.T.; methodology, G.T.; software, G.T.; validation, G.T.; formal analysis, G.T.; investigation, G.T.; resources, G.T.; data curation, G.T.; writing—original draft preparation, G.T.; writing—review and editing, G.T.; visualisation, G.T.; supervision, G.T.; project administration, G.T.; funding acquisition, G.T.; All authors have read and agreed to the published version of the manuscript.

References

[1] T. M. Shih, C. H. Tan, and B. C. Hwang, "Effects of grid staggering on numerical schemes," *Int.J.Num.Methods in Fluids*, vol. 9, pp. 193 – 212, 1989.

[2] F. Wang, S. Wu, and S. Zhu, "Numerical simulation of flow separation over a backward-facing step with high Reynolds number," *Water Sci. Engng.*, vol. 12(2), pp. 145 – 154, 2019.

[3] C. Fureby, G. Tabor, H. G. Weller, and A. D. Gosman, "Large eddy simulation of the flow around a square prism," *AIAA.J.*, vol. 38, no. 3, pp. 442 – 452, 2000.

[4] U. Ghia, K. Ghia, and C. Shin, "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method," *Journal of Computational Physics*, vol. 48, no. 3, pp. 387–411, 1982. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0021999182900584

[5] E. Erturk, T. C. Corke, and C. Gökçöl, "Numerical solutions of 2-d steady incompressible driven cavity flow at high Reynolds numbers," *International Journal for Numerical Methods in Fluids*, vol. 48, no. 7, pp. 747–774, 2005. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.953

[6] P. Shankar and M. D. Deshpande, "Fluid mechanics in the driven cavity," *Ann.Rev.Fluid Mech.*, vol. 32, pp. 93 – 136, 2000.

[7] H. Kuhlmann and F. Romano, *The Lid-Driven Cavity*, 04 2018.

[8] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby, "A tensorial approach to computational continuum mechanics using object orientated techniques," *Computers in Physics*, vol. 12, pp. 620–631, 1998.

[9] O. Botella and R. Peyret, "Benchmark spectral results on the lid-driven cavity flow," *Comp. Fluids*, pp. 421–433, 1998.

[10] A. Y. Gelfgat, "Linear instability of the lid-driven flow in a cubic cavity," *Theor. Comput. Fluid Dyn.*, vol. 33, pp. 59 – 82, 2019.

[11] T. P. Chiang, H. R. R, and W. H. Sheu, "On end-wall corner vortices in a lid-driven cavity," *J. Fluids Engng.*, vol. 119(1), pp. 201–204, 1997.

[12] I. Mat Sahat, N. A. Che Sidik, and N. Mohd Izual, "Vortex structure in a two dimensional triangular lid-driven cavity," vol. 1440, 06 2012, pp. 1078–1084.

[13] B. An and J. Bergada, "Numerical study of the 2d lid-driven triangular cavities based on the Lattice Boltzmann method," 12 2016.

[14] C. H. Marchi, R. Suero, and L. K. Araki, "The lid-driven square cavity flow: Numerical solution with a 1024×1024 grid," *J. Braz.Soc. of Mech.Sci & Eng.*, vol. 31#3, pp. 186 – 198, 2009.

[15] G. R. Tabor, "OpenFOAM(tm): an Exeter Perspective," in *ECCOMAS CFD 2010, Lisbon, Portugal, 14 – 17th June 2010*, J. C. F. Pereira and A. Sequeira, Eds. ECCOMAS, 2010.

[16] F. Juretic, "Error analysis in finite volume CFD," Ph.D. dissertation, Imperial College, University of London, Department of Mechanical Engineering, December 2004.