A 2D Spectral Element Method for Unsteady Stokes Equations

Alex Vukadinovic (MMATH)

1 Introduction

This paper presents a numerical solution to the 2D unsteady Stokes equation modeled after the CAM-SE dynamical core. The MATLAB code accompanying this overview implements the following CAM-SE methods: Gauss-Lobatto-Legendre (GLL) quadrature for numerical integration, degree 3 Lagrange polynomials interpolated at the tensor product of GLL points (4x4 point square grid) and the spectral element method (using the Lagrange polynomial basis) for spatial discretization. Each of these methods is outlined in what follows. The file *stokesU.m* calls on several functions which are placed in the same folder as the above file. All files included with this documentation were written by the author of this paper. It should be noted that many derivations and details are omitted in order to provide a general overview of the procedure.

1.1 The Governing Equations

The Stokes equations are derived by considering the nondimensionalised ($\tilde{}$) Navier-Stokes equations:

$$Re\left(\frac{\partial}{\partial t}\tilde{\mathbf{u}} + \tilde{\mathbf{u}}\cdot\nabla\tilde{\mathbf{u}}\right) = -\nabla\tilde{\mathbf{p}} + \nabla^2\tilde{\mathbf{u}} \tag{1}$$

where $Re = UL \setminus \nu$ is the Reynolds number, and the velocity and pressure terms are in dimensionless form. If the kinematic viscosity ν is much larger than the typical speeds U and lengths L, then $Re << 1$ and the left hand side can be omitted to obtain the Stokes equations. It is possible however that there is more than one Reynolds number, i.e. if there is more than one choice for U and L. For example, if the fluid sits atop an oscillating lower boundary which impacts the velocity field of the fluid, the typical scales of this oscillation can result in a Reynolds number which is not $<< 1$. In this case we may not necessarily drop the time derivative form the LHS. This case is the unsteady Stokes equations which is handled by the MATLAB code. In component form, our system is given by the unsteady Stokes equations and the incompressibility condition:

$$\rho\frac{\partial u_x}{\partial t} = -\frac{\partial p}{\partial x} + \mu\left(\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2}\right) \tag{2}$$

$$\rho\frac{\partial u_y}{\partial t} = -\frac{\partial p}{\partial y} + \mu\left(\frac{\partial^2 u_y}{\partial x^2} + \frac{\partial^2 u_y}{\partial y^2}\right) \tag{3}$$

$$\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} = 0 \tag{4}$$

These equations are again in their dimensional form, specifically the code will use SI units for all variables. The numerical method solves this system for the x and y velocity components u_x and u_y respectively, as well as the pressure p.

2 Solution Method

2.1 Lagrange Polynomials and 2D Interpolation

The spectral element method employed in the code uses Lagrange polynomials to approximate the functions u_x, u_y and p satisfying (2), (3) and (4) for a given density ρ, viscosity μ and set of initial and boundary conditions at a given time t. To approximate a know function $f(x, y)$ over a given domain using third degree Lagrange polynomials we have the following:

$$f(x, y) \approx L_f(x, y) \equiv \sum_{i,j=1}^{4} f(x_i, y_j)\psi_i(x)\psi_j(y) \tag{5}$$

$$\psi_1(\xi) = \frac{(\xi - \xi_2)(\xi - \xi_3)(\xi - \xi_4)}{(\xi_1 - \xi_2)(\xi_1 - \xi_3)(\xi_1 - \xi_4)}, ..., \psi_4(\xi) = \frac{(\xi - \xi_1)(\xi - \xi_2)(\xi - \xi_3)}{(\xi_4 - \xi_1)(\xi_4 - \xi_2)(\xi_4 - \xi_3)} \tag{6}$$

Hence for example $\psi_1(\xi_k)$ is zero if $k \neq 1$ and one if $k = 1$ for $k = 1, 2, 3, 4$. In the code the function *gll_lag.m* returns the 16 basis polynomials as well as their derivatives, which will be needed for the spectral element method. The 16 unique interpolation points (x_i, y_j) can be chosen anywhere in the domain. We use the tensor product of the 4 GLL points for reasons discussed in section 2.3. The arbitrary function $f(x, y) = ysin(x)cos(y) + 2xy$ is plotted in figure 1 along with its interpolation $L_f(x, y)$ on $[-1, 1] \times [-1, 1]$ and the error $f(x, y) - L_f(x, y)$.

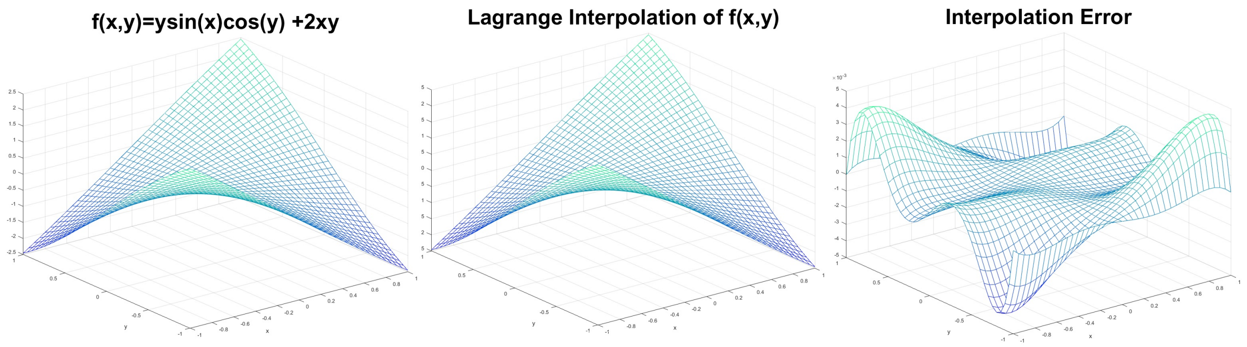

Figure 1: Plot of $f(x, y) = ysin(x)cos(y) + 2xy$, $L_f(x, y)$ and $f(x, y) - L_f(x, y)$.

The function $f(x, y)$ is sampled at the 16 GLL points and the only error is the interpolation error since $f(x_i, y_j)$ is known at all 16 points. In the case where $f(x, y)$ is unknown, the coefficients $f(x_i, y_j)$ must be estimated, introducing a second source of error. Total error reduction is facilitaed by *mesh refinement* discussed in he next section.

2.2 Domain Discretisatioin

The MATLAB code handles a rectangular domain which can be subdivided into further rectangular elements, each of which contains 16 interpolation nodes. The MATLAB code uses the numbering convention of the nodes as depicted for a sample domain $[0, 2] \times [0, 1]$ subdivided into two elements.The global labels for each node are given in blue, while the local labeling of the nodes is given in red. No local labels are given for the first element since they coincide with their global labels. Each element is mapped to the canonical ξ-η domain $\Lambda = [-1,1] \times [-1,1]$ using the linear maps:

$$x(\xi) = \frac{x_1^{E_k} + x_2^{E_k}}{2} + \frac{x_2^{E_k} - x_1^{E_k}}{2}\xi \tag{7}$$

$$y(\eta) = \frac{y_1^{E_k} + y_2^{E_k}}{2} + \frac{y_2^{E_k} - y_1^{E_k}}{2}\eta \tag{8}$$

The values $x_1^{E_k}, x_2^{E_k}, y_1^{E_k}$ and $y_2^{E_k}$ are the minimum and maximum x and y values of an element E_k respectively.

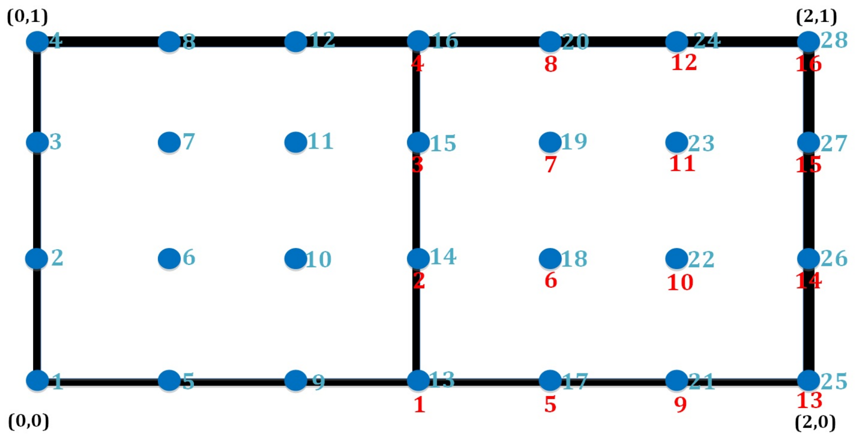

Figure 2: Discretized domain $[0,2] \times [0,1]$ into 2 elements.
Left: Element 1, Right: Element 2

The spectral element code solves for the polynomial coefficients locally over the Λ domain, then maps these coefficients to the proper *global* node. Hence we require a mapping from the local labels to the global labels for each element. For example, for the second element we require a function such that $F(7) \to 19$. This correspondence is accomplished using the function *connectivity.m* which returns the *connectivity matrix*. The connectivity matrix **C** has the form:

C_{ij} = the global label of the *jth* node in the *ith* element

For the discretization setup in Figure 2, entering in C in the MATLAB console after running the code gives:

```
>> C

C =

     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16
    13    14    15    16    17    18    19    20    21    22    23    24    25    26    27    28
```

Figure 3: The connectivity matrix for Figure 2.

The function *connectivity.m* also returns a vector called *boundary* which lists all the nodes which lye on the global boundary. For the same example we have:

```
>> boundary

boundary =

     1     2     3     4     5     8     9    12    13    16    17    20    21    24    25    26    27    28
```

Figure 4: The the boundary points for Figure 2.

The connectivity matrix is largely used during the *assembly stage* (section 2.5) while the boundary matrix identifies boundary nodes for the implementation of boundary conditions.

2.3 Galerkin Spectral Element Method for Spatial Discretization

In the MATLAB code the variable N_g contains the number of global nodes and N_e is the number of elements the domain is partitioned by. The function estimates of u_x, u_y and p are given by:

$$\mathbf{u} \approx \sum_{j=1}^{N_g} \mathbf{u}_j(t)\phi_j \qquad p \approx \sum_{j=1}^{N_e} p_j(t)\phi_j^p \tag{9}$$

The functions ϕ_j are the short form notation for the Lagrange polynomials in (5), and the coefficients are now functions of the time t. The pressure approximation is simplified in the code to be a constant over each element by way of defining $\phi_j^p = 1$ on element k and $\phi_j^p = 0$ otherwise. We multiply (2) and (3) by a *test function*, which in the case of the Galerkin spectral element method is just ϕ_i, the same polynomials used for (9). The resulting equation is integrated over the element E_k and after some manipulations, the order of the derivatives on the RHS of (2) and (3) are reduced and the following system is produced:

$$\rho \sum_{j=1}^{16} M_{ij} \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{u}_j(t) = \sum_{j=1}^{16} \mathbf{Q}_{ij} p_j(t) + \sum_{j=1}^{16} \mathbf{N}_{ji} p_j(t) - \mu \sum_{j=1}^{16} R_{ij}\mathbf{u}_j(t) - \mu \sum_{j=1}^{16} D_{ij}\mathbf{u}_j(t) \tag{10}$$

where

$$\mathbf{Q}_{ij} = \begin{bmatrix} Q_{xij} \\ Q_{yij} \end{bmatrix} = \begin{bmatrix} \oint\limits_{\partial E_k} \phi_i \phi_j^p n_x \mathrm{d}\ell \\ \oint\limits_{\partial E_k} \phi_i \phi_j^p n_y \mathrm{d}\ell \end{bmatrix} \qquad \mathbf{N}_{ij} = \begin{bmatrix} N_{xij} \\ N_{yij} \end{bmatrix} = \begin{bmatrix} \iint\limits_{E_k} \frac{\partial \phi_j}{\partial x} \phi_i^p \mathrm{d}\ell \\ \iint\limits_{E_k} \frac{\partial \phi_j}{\partial y} \phi_i^p \mathrm{d}\ell \end{bmatrix} \tag{11}$$

and

$$M_{ij} = \iint\limits_{E_k} \phi_i \phi_j \mathrm{d}x\mathrm{d}y \quad R_{ij} = \oint\limits_{\partial E_k} \phi_i \mathbf{n} \cdot \nabla\phi_j \mathrm{d}\ell \quad D_{ij} = \iint\limits_{E_k} \nabla\phi_i \cdot \nabla\phi_j \mathrm{d}x\mathrm{d}y \tag{12}$$

The continuity equation is projected onto the pressure interpolation function ϕ_i^p and is given in matrix notation as:

$$\sum_{j=1}^{16} \mathbf{N}_{ij} \cdot \mathbf{u}_j(t) = 0 \tag{13}$$

Each of the matrices given in (11) and (12) are calculated by the function *matrix_sk.m*. In order to compute the integrals needed, the function *product.m* uses GLL quadrature to evaluate the integrals exactly. The vector $P = [-1, -\sqrt{1/5}, \sqrt{1/5}, 1]$ stores the 1D GLL points, and $w = [1/6, 5/6, 1/6, 5/6]$ stores the weights for the quadrature. Using (7) and (8) the integrals are evaluated over the domain $[-1,1] \times [-1,1]$ using the tensor products $P \otimes P$ and $w \otimes w$. The domain interpolation nodes were chosen to be the 16 GLL points in order to be able to integrate the polynomials exactly. For example, M_{ij} is evaluated by:

$$\iint_{E_k} \phi_i(x,y)\phi_j(x,y)\mathrm{d}x\mathrm{d}y = J\int_{-1}^{1}\int_{-1}^{1} \phi_i(\xi,\eta)\phi_j(\xi,\eta)\mathrm{d}\xi\mathrm{d}\eta = J\sum_{i,j=1}^{4} w_i w_j \phi(P_i,P_j)\phi(P_i,P_j) \tag{14}$$

The Jacobean of the transformations (7) and (8) is $J = (x_2^{E_k} - x_1^{E_k})(y_2^{E_k} - y_1^{E_k})/4$ and is the same constant for every element. Hence the matrices in (11) and (12) are the same for every element and so are only calculated once. We note that these are the solutions for a given element E_k and so must be combined into a global system for our whole domain. The function *matrix_sk.m* also accomplishes this assembly process and returns the global matrices to *stokesU.m*. This assembly process is outlined in section 2.5.

2.4 Time Discretisation

The spectral element has reduced the system of continuous partial differential equations given by (2)-(4) to a discretized system of ordinary differential equations given by (10) and (13). We can now apply a Crank-Nicolson discretization for the ODE to obtain a scheme which can be iterated through time. For each element we obtain the system:

$$\frac{\rho}{\Delta t}\sum_{j=1}^{16} M_{ij}(\mathbf{u}_j^{n+1} - \mathbf{u}_j^n) = \sum_{j=1}^{16}\mathbf{Q}_{ij}p_j^{n+1} + \sum_{j=1}^{16}\mathbf{N}_{ji}p_j^{n+1} - \frac{\mu}{2}\sum_{j=1}^{16} R_{ij}(\mathbf{u}_j^{n+1} + \mathbf{u}_j^n) - \frac{\mu}{2}\sum_{j=1}^{16} D_{ij}(\mathbf{u}_j^{n+1} + \mathbf{u}_j^n) \tag{15}$$

The continuity equation is evaluated at time $n+1$ to produce:

$$\sum_{j=1}^{16}\mathbf{N}_{ij}\cdot\mathbf{u}_j^{n+1} = 0 \tag{16}$$

Along with the initial and boundary conditions, the above is a closed system which can be used to solve for the unknown velocity and pressure coefficients at time $n+1$. Solving for these coefficients is done in the context of the global system, and hence left to the next section. Essentially the time stepping scheme amounts to taking a centered difference for the derivative and averaging the velocity for the last two terms.

2.5 Assembly of the Global System and Final Solution

By handling each element separately over the domain, we obtain N_e systems of equations for each u_x, u_y and p. Simply solving over each element and adding the N_e functions together will likely produce a discontinuity at the shared element edges. For example in figure 2, the edge containing nodes 13,14,15 and 16 may be assigned function values by element one which do not agree with those assigned by element two. The assembly process is instead used to guarantee a continuous solution over the domain using global matrices assembled from the local element matrices. With the help of the connectivity matrix, *matrix_sk.m* assembles the global matrices and returns those to *stokesU.m* and not the element matrices listed in (10) and (11). The components of the 16×16 element matrices which are associated with shared nodes are added together with the values from other elements which share these nodes. For simplicity, consider the following discretization using only 4 nodes per element:

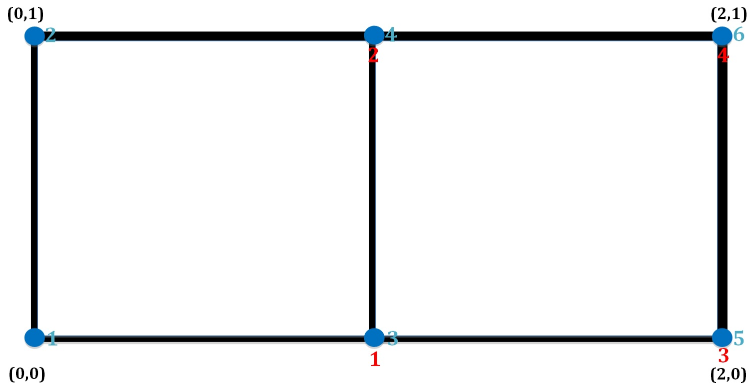

Figure 5: Example discretization using only corner nodes
Left: Element 1, Right: Element 2

and suppose the element mass matrices are given by:

$$\mathbf{M}^{E_1} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \qquad \mathbf{M}^{E_2} = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} \tag{17}$$

The connectivity matrix defines the node mapping functions such that for element two: $F_2(1) \to 3$, $F_2(2) \to 4$, $F_2(3) \to 5$ and $F_2(4) \to 6$. Using this map on the indices of the two matrices and adding the corresponding entries produces the global mass matrix for the entire domain:

$$\mathbf{M}^G = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 \\ a_{31} & a_{32} & a_{33}+b_{11} & a_{34}+b_{12} & b_{13} & b_{14} \\ a_{41} & a_{42} & a_{43}+b_{21} & a_{44}+b_{22} & b_{23} & b_{24} \\ 0 & 0 & b_{31} & b_{32} & b_{33} & b_{34} \\ 0 & 0 & b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} \tag{18}$$

A similar assembly process is applied to all the element matrices in (11)-(12) to produce the global system. Element matrices can be independently constructed, inter-cell communication is not required until the assembly of the global solution. This is essential for the scalability of spectral element dynamical cores -such as the one in CAM-SE- on massively parallel machines.

For completeness, we state the final global system solved by *stokesU.m* for the system of equations given by (2)-(4) as:

$$\begin{bmatrix} \frac{\mu}{2}(\frac{2}{\nu\Delta t}\mathbf{M}^G + \mathbf{D}^G + \mathbf{R}^G), & \mathbf{0}, & -\mathbf{N}_x^{G\prime} - \mathbf{Q}_x^G \\ \mathbf{0}, & \frac{\mu}{2}(\frac{2}{\nu\Delta t}\mathbf{M}^G + \mathbf{D}^G + \mathbf{R}^G), & -\mathbf{N}_y^{G\prime} - \mathbf{Q}_y^G \\ \mathbf{N}_x^G, & \mathbf{N}_y^G, & \mathbf{0} \end{bmatrix} \mathbf{X} \tag{19}$$

$$= \begin{bmatrix} \frac{\mu}{2}(\frac{2}{\nu\Delta t}\mathbf{M}^G - \mathbf{D}^G - \mathbf{R}^G), & \mathbf{0} \\ \mathbf{0}, & \frac{\mu}{2}(\frac{2}{\nu\Delta t}\mathbf{M}^G - \mathbf{D}^G - \mathbf{R}^G) \\ \mathbf{0}, & \mathbf{0} \end{bmatrix} \mathbf{U}^n \tag{20}$$

where

$$\mathbf{X} = \left[u_{x_1}^{n+1}, u_{x_2}^{n+1}, ...u_{x_{Ng}}^{n+1}, u_{y_1}^{n+1}, ..., u_{y_{N_g}}^{n+1}, p_1^{n+1}, ..., p_{N_e}^{n+1}\right]' \tag{21}$$

$$\mathbf{U}^n = \left[u_{x_1}^n, u_{x_2}^n, ..., u_{x_{Ng}}^n, u_{y_1}^n, ..., u_{y_{N_g}}^n\right]' \tag{22}$$

The velocity and pressure coefficients in $\mathbf{X}$ are solved for in every time step for a set of initial and boundary conditions. The next section presents some custom scenarios of unsteady Stokes flow which is solved by the MATLAB code.

3 Unsteady Stokes Flow Examples

Examples are presented in this section on a $[0, 2] \times [0, 2]$ rectangular domain for a range of boundary conditions. In order to plot the functions, *nodal_coord.m* assigns the coefficients which have been obtained for the unknown functions to the correct Lagrange polynomial. It then evaluates these continuous functions at the resolution specified and passes the values to be plotted. The initial state of the fluid is at rest in all cases in order to make clear the impacts of the boundary conditions on the motion of the fluid. In all cases we have chosen $\mu = 1$ and $\rho = 1$, though other values can easily be set. Accompanying the text here are *.avi* files which show the time progression of system where both velocity vector fields and surface plots of the scalar speed fields are included. Lower resolutions for the videos were chosen for the sake of computation time, but high resolution images of the steady state are included here and in separate *.jpg* files. Also included is the MATLAB code for each video with the appropriate initial and boundary conditions specified.

3.1 Example 1: Accelerated Left Boundary

Fluid inside a closed cavity is subject to a moving left wall boundary which is linearly accelerated to from $0ms^{-1}$ to $1ms^{-1}$ in a $10s$ time span. We can imagine a moving plate attached to the left end of the cavity being accelerated vertically on the y-axis. The steady state solution is given in figure 6 in the form of the velocity vector field (left) and the speed plot (right).

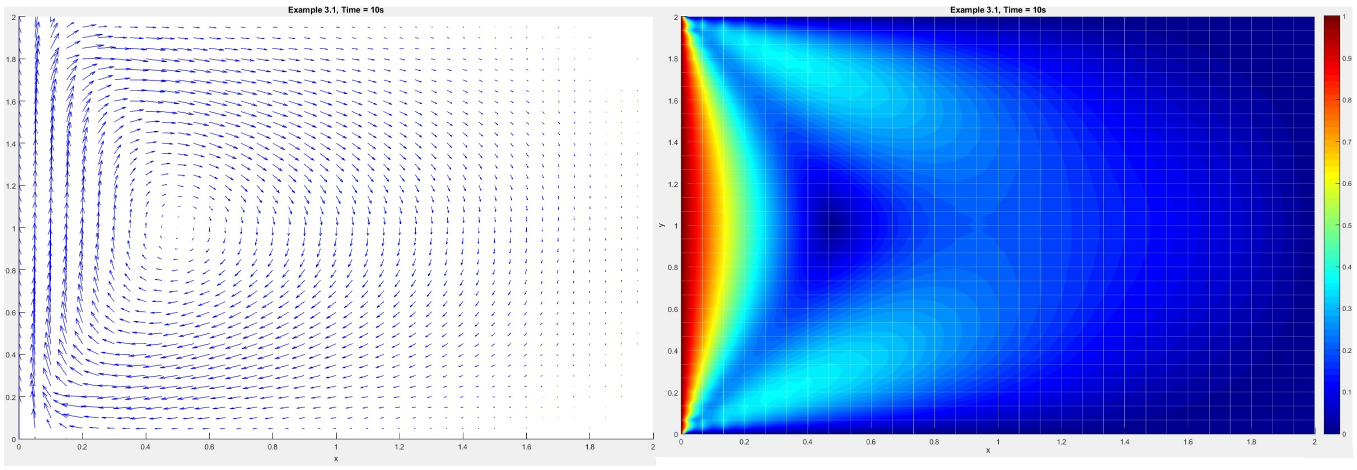

Figure 6: Accelerated left boundary in closed cavity
Left: Velocity field, Right: Scalar field of fluid speed

The solid boundary is specified by applying the no-slip and no-through boundary conditions at all the walls except for the left wall where only the no-through boundary condition is applied. In other words, at the top, bottom and right walls we set $u_x = 0$ and $u_y = 0$, and set $u_x = 0$ at the left wall. The wall is accelerated by setting $u_y = t/10$ at the left wall. Note that there is no fluid escaping, the vectors which extend outside the domain all represent the velocity of a point inside the cavity. A vortex clearly forms in the fluid and an 'eye of the storm' develops where the fluid is relatively stationary.

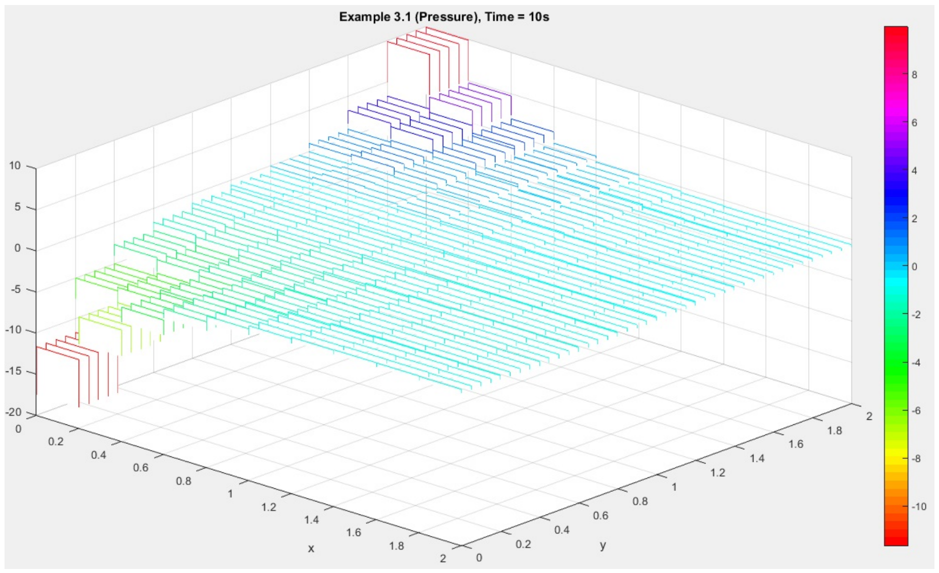

Figure 7: Pressure field for accelerated left wall

In figure 7 we plot the pressure function $p_j(10)\phi_j$ for $j = 1, 2 ... N_e$. The pressure field is inversely symmetric across the x-axis. The fluid is forced up into the top left corner of the cavity and away from the bottom right corner, creating high and low pressure zones respectively. Away from these regions the pressure field is relatively constant.

3.2 Example 2: Accelerated Left Boundary With Free Boundary Conditions

The boundary conditions are now modified to allow for the fluid to enter and exit the cavity. Holes are placed in the boundary by not specifying any boundary conditions at these locations. In figure 8 we denote these gaps by a green rectangle, in this case placed at the bottom left corner and at the right wall of the boundary.

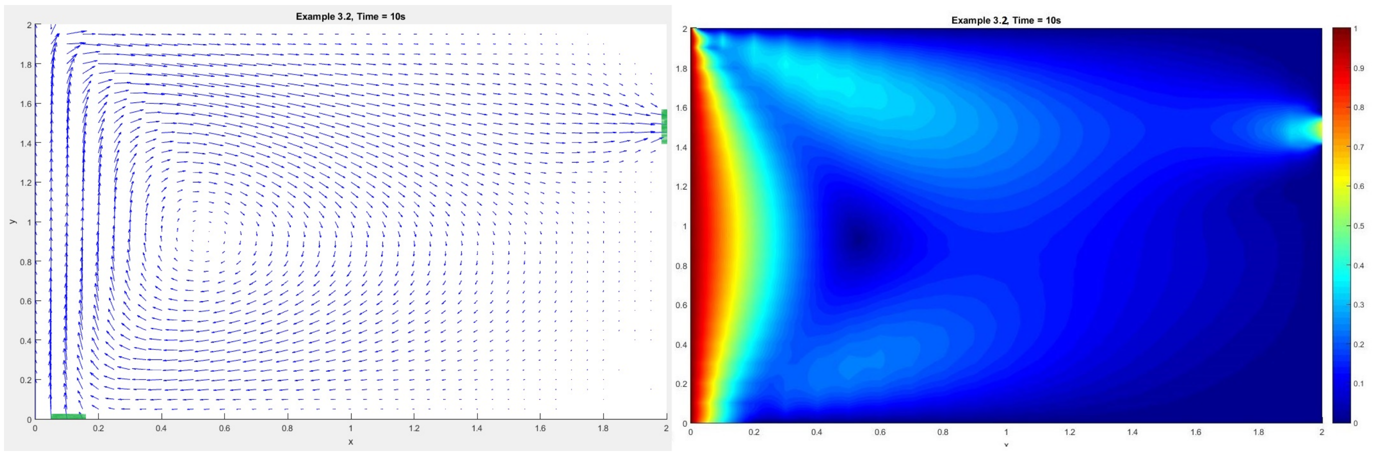

Figure 8: Accelerated left wall with gaps in boundary
Left: Velocity field, Right: Scalar field of fluid speed

The moving left boundary forces fluid into the cavity and since our fluid is incompressible, the fluid exits the cavity at the right wall gap at a rate to avoid compression. The center of the vortex is slightly moved downwards as compared to example 3.1. The gap in the right wall is chosen to be relatively small in order to force the fluid to exit the cavity at a relatively high velocity. Also, we note that the fluid in the upper cavity now moves faster than that at the lower half.

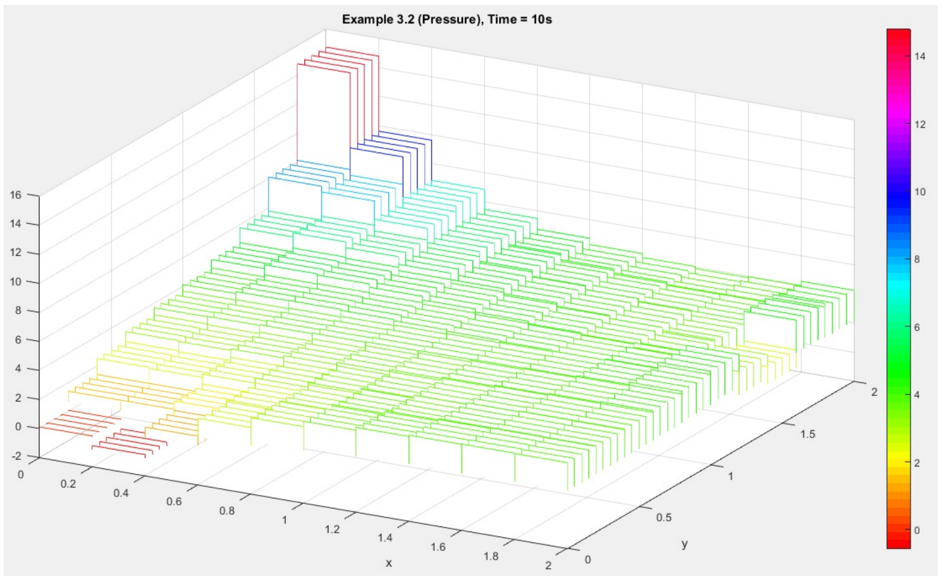

Figure 9: Pressure field for accelerated left wall with gaps in boundary

Since fluid is allowed to flow into the cavity from the bottom left corner, the pressure gradient in this region is much smaller than that of the top left corner in figure 9. We also see a low pressure buildup around the region where the fluid exits the cavity on the right wall. The low pressure forces the fluid to leave at a rate consistent with the incompressibility condition.

3.3 Example 3: Flow Around a Solid Boundary

Next consider the case of a jet of fluid entering the cavity from the left wall and an open gap allowing fluid to escape on the right wall, as depicted in figure 10. In addition, an obstacle is placed in the path of the fluid altering its trajectory. Fluid enters the cavity at an increasing rate of $u_x = t/10$.

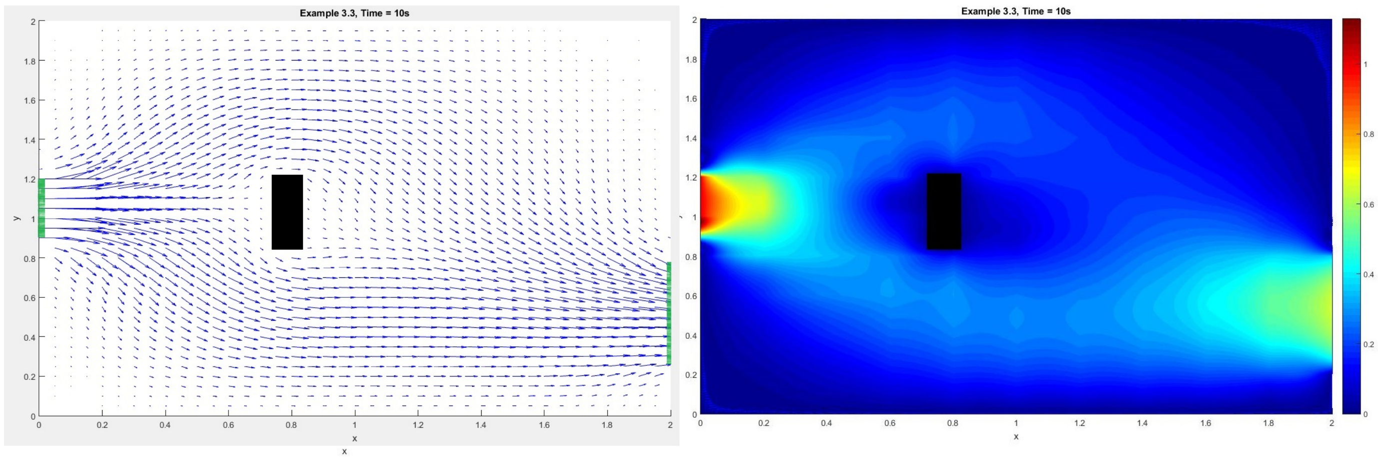

Figure 10: Fluid entering cavity with obstacle
Left: Velocity field, Right: Scalar field of fluid speed

In order to create an obstacle in the fluid, we apply boundary conditions to nodes which are not located on the boundary of the domain, i.e. not on a wall of the cavity. Similar to example 3.1, no-slip and no-through boundary conditions are applied to the nodes which make up the obstacle. The gap in the right wall is larger than the gap on the left wall, hence we note that the velocity of the fluid exiting the cavity is less than the velocity of the fluid entering, as expected.

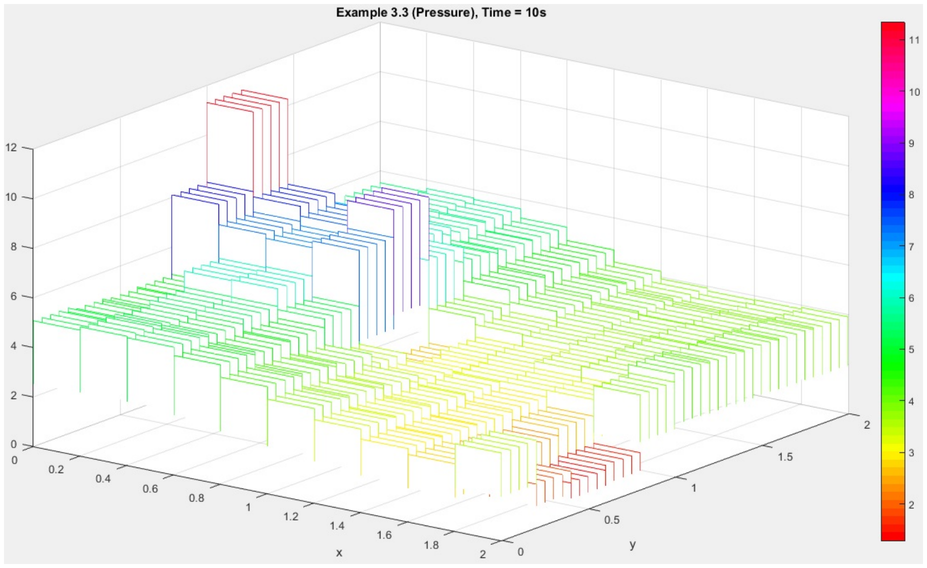

Figure 11: Pressure field for fluid entering cavity with obstacle

We note the higher pressure at the front of the barrier and the decrease on the other side in figure 11. Since the gap in the left wall is larger than that at the right, the fluid enters the cavity at a higher pressure than it exits in order for the fluid to remain uncompressed inside the cavity.

References

Pozrikidis, C. Introduction to Finite and Spectral Element Methods Using MATLAB. Boca Raton: Chapman and Hall/CRC, 2005. Print.