Galerkin/Least Squares Finite Element Method for Fluid Flow Problems

Kameswararao Anupindi*

ME697F Project Report – April 30, Spring 2010

Abstract. Standard Galerkin finite element method [3] augmented with least square stabilization is known as Galerkin/least squares (GaLS) finite element method [4]. In this project we study and apply GaLS method for solving fluid flow problems. We discuss the construction and application of least squares stabilization for 1D, 2D convection - diffusion equations and for 2D Navier Stokes equations. GaLS method provides stability without altering the test function space. The fact that test function space is unaltered makes least squares stabilization to be called as *consistent* stabilization. Stability and consistency together drive the solution to convergence. GaLS was implemented in FreeFem++ and 2D Navier Stokes equations are solved for lid driven cavity and flow past a square cylinder problems. Lid driven cavity simulations are compared with the solutions obtained from Acusolve.

Key words: Galerkin finite element method, least squares, fluid flow

1 Introduction

Computational difficulties and shortcomings of the Galerkin finite element method have lead to the development of alternate finite element methods. The objective in developing alternate finite element methods has been to avoid the two main problems encountered with standard finite element methods [3]. The two main problems with standard finite element method are (1) generation of wiggles or oscillations in the solution, and (2) limiting the choice of test function spaces for pressure and velocity in order to satisfy the Ladyshenskaya - Babuška - Brezzi (LBB) stability condition. GaLS [4] is a consistent stabilization method which avoids both the above problems [6] [2]. In this project we discuss the contruction and application of GaLS method for 1D, 2D convection - diffusion problem and for solving 2D Navier - Stokes equations using FreeFem++ [5]

*Graduate Student, School of Mechanical Engineering, Purdue University, West Lafayette, IN 47907. (kanupind@purdue.edu).

2 Galerkin/Least Squares Method

2.1 Construction of least squares term

Consider a differential operator **D** acting on a vector of unknowns **u** with a vector of external force **f**

$$\mathbf{Du} = \mathbf{f} \tag{2.1}$$

Let us define a quadratic functional in terms of the L_2 norm of the equation residual

$$\mathbf{I}(\mathbf{u}) = \frac{1}{2}\|(\mathbf{Du} - \mathbf{f})\|^2 \tag{2.2}$$

For **u** to be a minimizer of the functional **I(u)**

$$\lim_{t\to 0} \frac{d}{dt}\mathbf{I}(\mathbf{u} + \mathbf{tv}) \equiv \mathbf{(Du\text{-}f,\ Dv)} \tag{2.3}$$

(Du-f, Dv) is called the least squares term, because we have minized the L2 norm of the equation residual to get this. We note that the differential operator **D** directly enters the least squares term.

Now, for the Eq. 2.1 the standard Galerkin method would give the weak form as follows:

$$(\mathbf{u}^h, \mathbf{v}^h) = \mathbf{L}(\mathbf{v}^h) \tag{2.4}$$

In Eq. 2.4 $(\mathbf{u}^h, \mathbf{v}^h)$ is a bi-linear inner product and $\mathbf{L}(\mathbf{v}^h)$ is linear term. The weighting functions $\mathbf{v}^h$ and the trial functions $\mathbf{u}^h$ belong to a restricted space $\mathbf{V}^h$.

In Galerkin/least squares method one augments the weak form of standard Galerkin method with the least squares term coming from the residual of the original differential equation as derived above. Hence, the Galerkin/least squares formulation for the differential equation 2.1 is as given below:

$$(\mathbf{u}^h, \mathbf{v}^h) + \int_\Omega \tau \mathbf{D}\mathbf{v}^h(\mathbf{D}\mathbf{u}^h - \mathbf{f})d\Omega = \mathbf{L}(\mathbf{v}^h) \tag{2.5}$$

As the space of trial functions is unaltered, the scheme is still consistent. The extra least squares term supplies stability in such a way that wiggles are suppressed. Consistency and stability together enable convergence of $\mathbf{u}^h$ to the exact solution. The parameter τ is chosen so that it gives optimal convergence properties.

2.2 1D - Convection Diffusion Equation

Let us now consider the 1D - convection diffusion equation in order to apply Galerkin/least squares method for solving it. The boundary value problem for 1D - convection diffusion is given by:

$$-\epsilon u_{xx} + \beta u_x = 0 \tag{2.6}$$

where $0 \leq x \leq 1$, with boundary conditions $u(0) = 0, u(1) = 1$
The unknown u in the Eq. 2.6 can be thought of as a scalar transported by diffusion (ϵ being the diffusion coefficient) and imposed with a convective velocity of β. Application of standard Galerkin method to Eq. 2.6 would result in the following discrete weak form of the equation.

$$\epsilon(u_x^h, v_x^h) + \beta(u_x^h, v^h) = 0, \qquad \forall v^h \in V^h \tag{2.7}$$

The Galerkin/least squares formulation of the Eq. 2.6 would append the standard Galerkin formulation Eq. 2.7 with the least squares term, as follows:

$$\epsilon(u_x^h, v_x^h) + \beta(u_x^h, v_h) + \tau \int_\Omega (-\epsilon v_{xx}^h + \beta v_x^h)(-\epsilon u_{xx}^h + \beta u_x^h) d\Omega = 0 \tag{2.8}$$

If we assume piece wise linear elements then the second order derivatives appearing in Eq. 2.8 vanish and the equation reduces to

$$(\epsilon + \tau\beta^2)(u_x^h, v_x^h) + \beta(u_x^h, v_h) = 0 \tag{2.9}$$

Where the parameter τ is chosen to be $\tau = \frac{h}{2\beta}\sqrt{\left(\frac{\alpha^2}{9+\alpha^2}\right)}$
$\alpha = Pe/2$, and $Pe = \beta h/\epsilon$ is Peclet number.

From the Eq. 2.9 we can see that addition of least squares term essentially behaves as adding an artificial diffusion to the standard Galerkin scheme.

2.3 2D - Convection Diffusion Equation

Formulation of Galerkin/least squares method for 2D - convection diffusion equation is very similar to the one done for 1D - convection diffusion equation above. Let us consider the 2D - convection diffusion equation given below:

$$-\epsilon\nabla^2\mathbf{u} + \beta\nabla\mathbf{u} = 0 \tag{2.10}$$

Where ϵ and β are the diffusion coefficient and known convective velocity.

Applying standard Galerkin method and adding least square term to the weak formulation we get :

$$\epsilon(\nabla\mathbf{u^h}, \nabla\mathbf{v^h}) + (\beta\nabla\mathbf{u^h}, \mathbf{v^h}) + \tau(-\epsilon\nabla^{\mathbf{2}}\mathbf{v^h} + \beta\nabla\mathbf{v^h})(-\epsilon\nabla^{\mathbf{2}}\mathbf{u^h} + \beta\nabla\mathbf{u^h}) = 0 \tag{2.11}$$

Here we note that all the terms that are multiplied by τ come from the addition of least squares term and the rest of the terms appearing in Eq. 2.11 are nothing but the standard Galerkin weak formulation terms.

2.4 2D Navier - Stokes Equations

Let us consider 2D incompressible Navier - Stokes equations and applying standard Galerkin method would lead to the weak form as shown below:

$$\nu \int_\Omega \nabla \mathbf{u} : \nabla \mathbf{v} + \int_\Omega (\mathbf{u}.\nabla \mathbf{u}).\mathbf{v} - \int_\Omega p(\nabla.\mathbf{v}) = \int_\Omega \mathbf{f}.\mathbf{v} \tag{2.12}$$

$$\int_\Omega q(\nabla.\mathbf{u}) = 0 \tag{2.13}$$

Where the Eq. 2.12 is the weak form of the momentum equation and the Eq. 2.13 is the weak form of the continuity equation. And $\mathbf{v}$ and $\mathbf{q}$ are the weighting functions for velocity $\mathbf{u}$ and pressure $\mathbf{p}$ respectively.

Adding the least squares term we get the Galerkin/least squares formulation of Navier - Stokes equations as below:

$$\nu \int_\Omega \nabla \mathbf{u} : \nabla \mathbf{v} + \int_\Omega (\mathbf{u}.\nabla \mathbf{u}).\mathbf{v} - \int_\Omega p(\nabla.\mathbf{v}) + \int_\Omega R_{GLS} d\Omega = \int_\Omega \mathbf{f}.\mathbf{v} \tag{2.14}$$

Where R_{GLS} is given by

$$R_{GLS} = \tau(\mathbf{u}.\nabla \mathbf{v} + \nabla q - \nu \nabla^{\mathbf{2}} \mathbf{v})(\mathbf{u}.\nabla \mathbf{u} + \nabla p - \nu \nabla^{\mathbf{2}} \mathbf{u}) \tag{2.15}$$

We can express the above equation in a short form as

$$R_{GLS} = \tau D(\mathbf{v}, q) D(\mathbf{u}, p) \tag{2.16}$$

Where D is the incompressible differential operator.
The parameter τ for steady state problems is given by

$$\tau = \left[\left(\frac{2||\mathbf{u}||}{h} \right)^2 + \left(\frac{4\nu}{h^2} \right)^2 \right]^{-1/2} \tag{2.17}$$

For unsteady problems τ is given by

$$\tau = \left[\left(\frac{2}{dt} \right) + \left(\frac{2||\mathbf{u}||}{h} \right)^2 + \left(\frac{4\nu}{h^2} \right)^2 \right]^{-1/2} \tag{2.18}$$

3 Results

Using GsLS we run various test cases in 1D and 2D as follows:

3.1 1D - Convection Diffusion

The 1D convection diffusion problem as described in Eq. 2.6 is solved using *myfem1d.f90* and piece-wise linear finite elements are used so the second derivative terms vanish in the formulation. In this problem because of high Peclet number we see that standard Galerkin method produces oscillations in the solution. Where as the GaLS method produces a solution which is smoother and without wiggles as shown in Fig. 1. The GaLS solution is closer to exact solution. Also, the number of elements is taken as 11 in both the standard Galerkin and GaLS method cases. Taking more number of elements makes the GaLS solution to more accurately coincide with the exact solution.

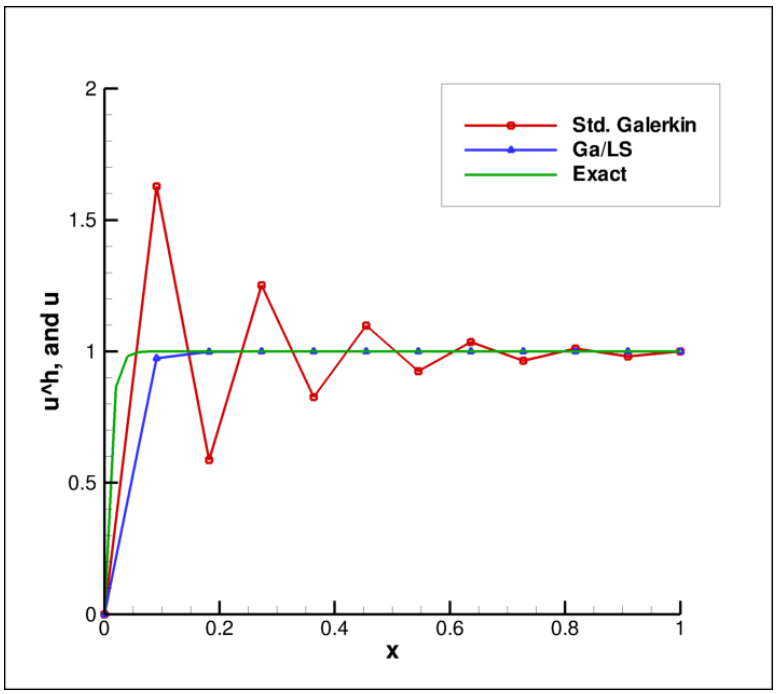

Figure 1: 1D convection diffusion solved using standard Galerkin and GaLS

3.2 2D - Convection Diffusion

In the 2D convection diffusion problem we solved the Eq. 2.11 on a square domain with the following boundary conditions.

$$u(x,-1) = x, \qquad u(x,1) = 0$$
$$u(-1,y) = -1, \qquad u(1,y) = 1$$

with a vertical wind $\beta = (0, 1)$, and $\epsilon = 1/200$. Solving this problem using standard Galerkin method would lead to wiggles in the solution very similar to the 1D problem. GaLS method produces a smooth solution and is shown in Fig. 2. FreeFem++ was used for running the 2D convection diffusion problem.

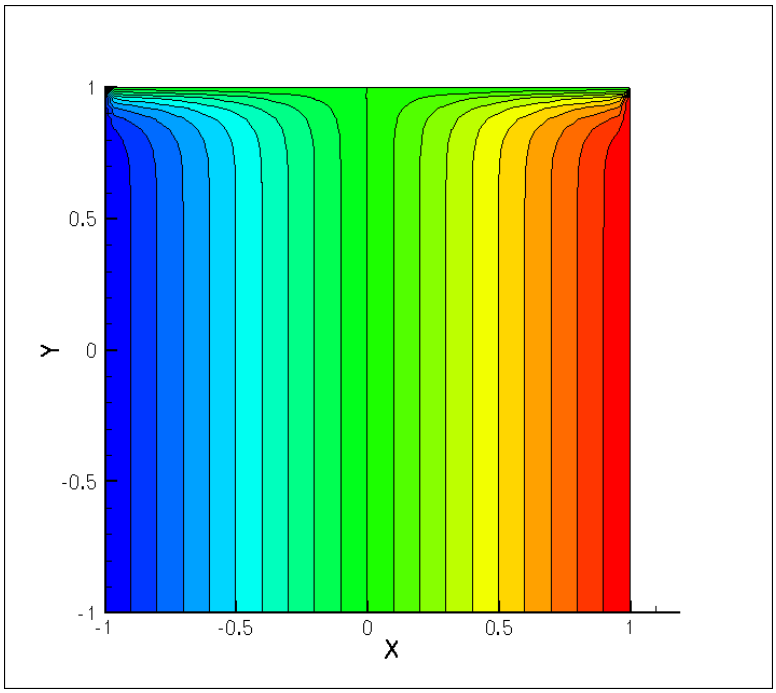

Figure 2: 2D convection diffusion equation solved using GaLS

3.3 2D Navier - Stokes Equations

Next, we solve 2D Navier Stokes equations as described in Eq. 2.14 by incorporating the least squares term to the weak formulation in FreeFem++. Two problems were considered, flow in a lid driven cavity and flow over a square cylinder placed in a channel.

3.3.1 Flow in a lid driven cavity

For the flow in a lid driven cavity, a square domain of $(-1, -1)$ to $(1, 1)$ was considered and the top wall of the square is given a velocity of 1 unit/sec. The viscosity of fluid is varied to reproduce varies Reynolds numbers of the flow. This problem was run using FreeFem++ and Acusolve. In FreeFem++ we run the problem in a 2D domain. In Acusolve we took three layers of cells in the z direction and *slip* boundary conditions were imposed on these front and back walls so as to simulate a 2D problem. The domain in both the cases (using FreeFEM++, and Acusolve) is discretized using equal number of triangular elements. The streamlines are shown plotted in Fig. 3 for a Re = 100 for the cases run in Acusolve and FreeFem++. In FreeFem++ P2 elements were taken for both velocity and pressure spaces.

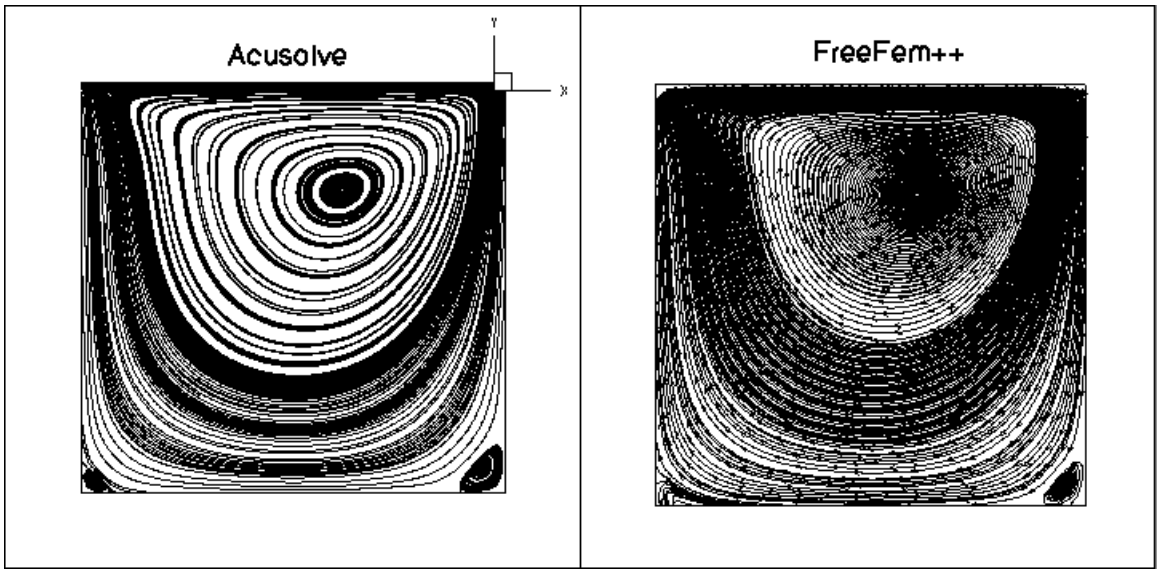

Figure 3: Streamlines for Re = 100

Further the u-velocity variation along the vertical centerline of the lid driven cavity is shown plotted in Fig. 4 Here the results obtained are compared to the literature values of Ghia et al. We can see that for a low Reynolds number of 100 the results obtained from FreeFem++ and Acusolve match closely to the published results.

By increasing the Reynolds number to 400, the lid driven cavity case was re-run. The streamlines obtained for a $Re = 400$ are shown plotted in Fig. 5. The streamlines seem to be similar between the Ga/LS FreeFem++ and the Acusolve in terms of the main recirculation eddy and in capturing the corner eddies. But a plot of u-velocity on the vertical centerline of the driven cavity as shown in Fig. 6 reveals that Ga/LS FreeFem++ do not match very well with the published results, but the Acusolve results are closely matching the published results.

3.3.2 Flow over a square cylinder placed in a channel

Next, we simulate the flow over a square cylinder placed in a rectangular channel. The geometry of the flow domain is shown in Fig. 7. Where D, the side of the square cylinder is taken as the length scale and the total length of the channel $L = 50D$. The height of the channel $H = 8D$, and the inlet length of the center of the square cylinder from the inlet is $L_{in} = 12.5D$. A parabolic velocity profile which represents a fully developed flow profile is imposed on the inlet. The mesh that is considered is shown in Fig. 8. This problem was run using FreeFem++ for Reynolds numbers ranging from $Re = 1, 10, 20, 30, 40$ and 50. In this case also P2 elements were chosen for discretizing both velocity and pressure spaces. The streamlines are shown plotted in Fig. 9 for various Reynolds numbers. We can see that as the Reynolds number is increased from $Re = 1$ to $Re = 50$ the recirculation length behind the square cylinder gradually increases. The variation of recirculation length of the eddies

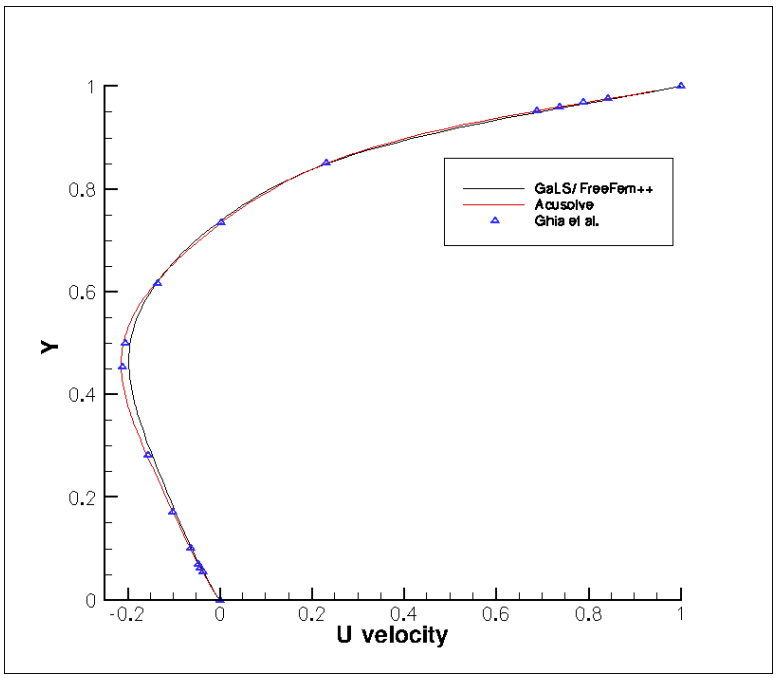

Figure 4: Comparison of U velocity along vertical center line for Re = 100

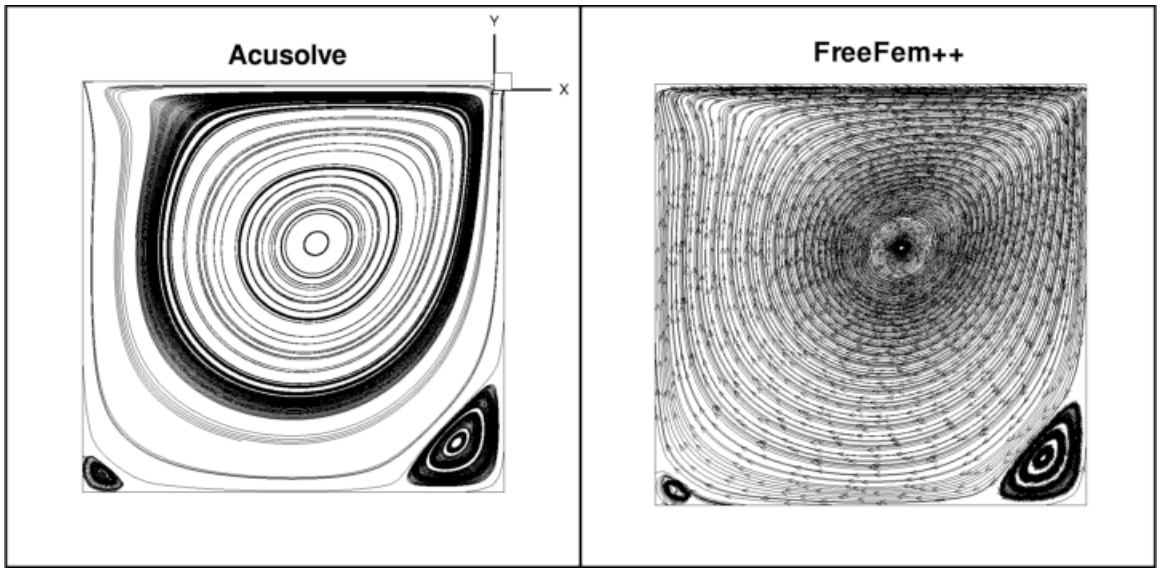

Figure 5: Streamlines for Re = 400

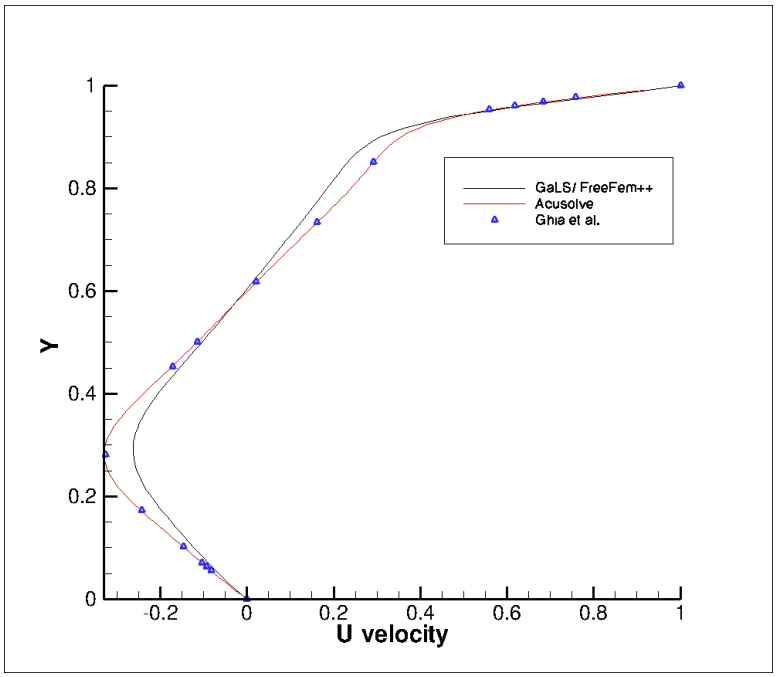

Figure 6: Comparision of U velocity along vertical center line for Re = 400

with Reynolds number is shown plotted in Fig. 10. We can see that it matches closely with the finite volume results of Breuer et al. [1] from the literature.

4 Conclusions

Construction of Galerkin/least squares method was discussed for convection - diffusion and Navier - Stokes equations and it is applied to solve the same. The results obtained with GaLS for a $Re = 100$ in a lid driven cavity matches closely with the published values and also with the results obtained from Acusolve. For higher $Re = 400$ the GaLS results have deviated but Acusolve results still match at this Reynolds number. For flow over a square cylinder the recirculation length is matching closely with that of literature.

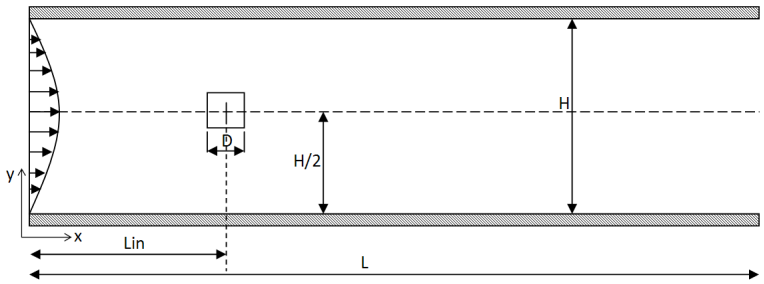

Figure 7: Channel geometry showing various dimensions

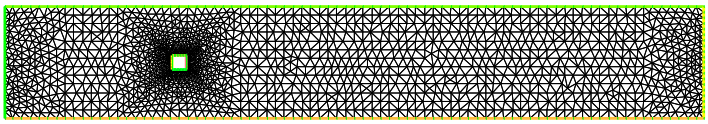

Figure 8: Mesh used for square cylinder in a channel problem

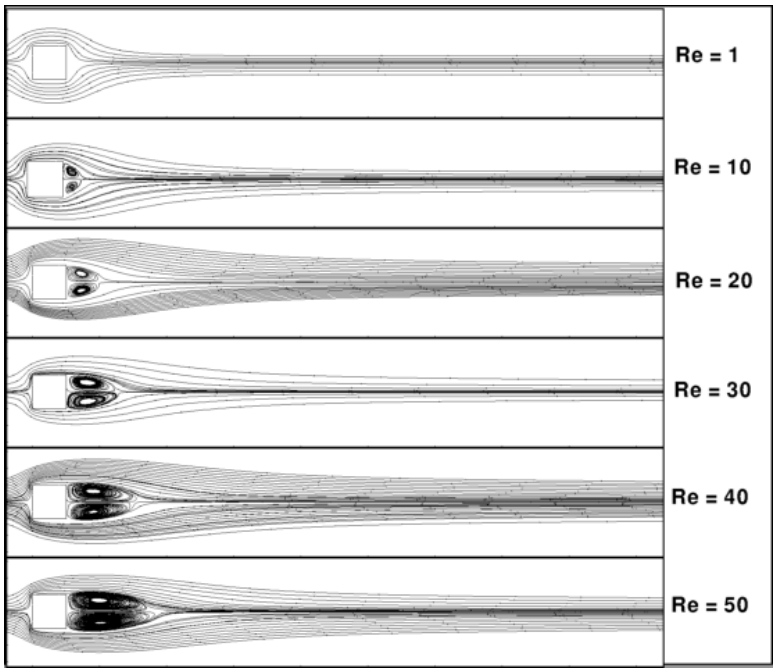

Figure 9: Streamlines for various Reynolds numbers

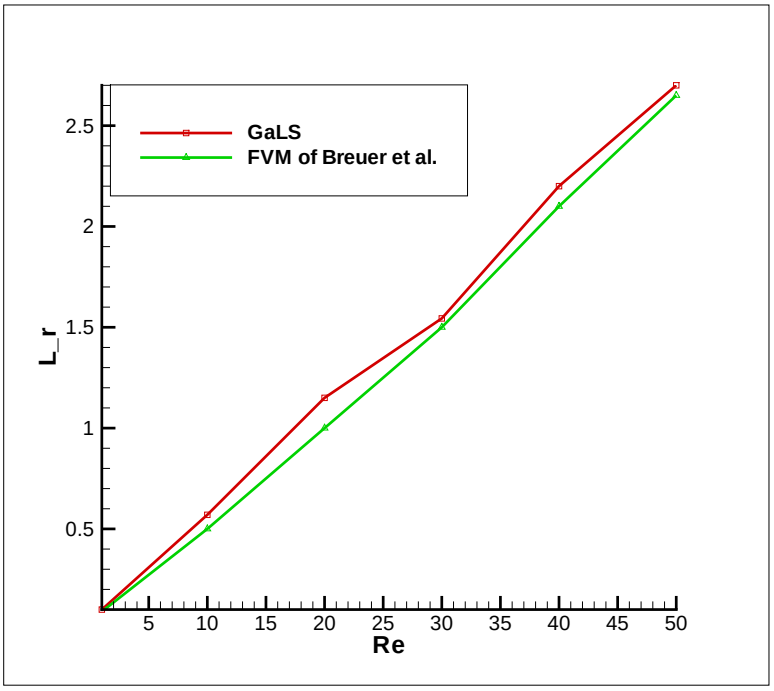

Figure 10: Variation of recirculation length with Re

5 Acknowledgements

I would like to thank *Dinesh Shetty* for his help on FreeFem++ related queries and *Jeffrey R. Kennington* for teaching me Acusolve.

References

[1] M. Breuer, J. Bernsdorf, T. Zeiser, and F. Durst, *Accurate computations of the laminar flow past a square cylinder based on two different methods: lattice-boltzmann and finite volume*, Intl. Journal of Heat and Fluid Flow, 21 (2000), pp. 186–196.

[2] J. Donea, *Finite Element Methods for flow problems*, John Wiley, 2004.

[3] H. Ellman, D. Silvester, and A. Wathen, *Finite Elements and Fast Iterative Solvers*, Oxford Science Publications, 2005.

[4] A. Ern and J.-L. Guermond, *Theory and Practice of Finite Elements*, Springer, 2004.

[5] F. Hecht, O. Pironneau, and J. Morice, *FreeFem++ reference manual.*

[6] D. G. Roddeman, *Some aspets of artificial diffusion in flow analysis.*

6 Code

```
real d = 1.0, L = 50.0*d, H = 8.0*d;
real xl = 0.0, xr = L, yb = 0.0, yt = H;
real lin = L/4.0;
real bbottom = (H/2.0 - d/2.0);
real btop = (H/2.0 + d/2.0);
real bleft = (lin - d/2.0);
real bright = (lin + d/2.0);

// define channel boundaries
border bot(t = 0, xr){x = t; y=yb; label=1;}
border outlet(t = 0, yt){x = xr; y = t; label = 2;}
border top(t = 0, xr){x = xr-t; y = yt;label = 3;}
border inlet(t = 0, yt){x = 0; y = yt-t;label = 4;}

// define box boundaries
border bb(t = -d/2.0, +d/2.0){x = lin+t; y = bbottom; label = 5;}
border br(t = -d/2.0, +d/2.0){x = bright; y = H/2.0 + t; label = 6;}
border bt(t = +d/2.0, -d/2.0){x = lin+t; y = btop; label = 7;}
border bl(t = +d/2.0, -d/2.0){x = bleft; y = H/2.0 + t; label = 8;}

int m = 80, n = 20, s = 10;

mesh Th = buildmesh(inlet(n) + bot(m) + outlet(n) +
                    top(m) + bb(-s) + br(-s) + bt(-s) + bl(-s));

plot(Th, wait = 0, ps ="mesh.eps");

int itmax = 100;
real nu=1.0/50.0;

func inflow = (1.0/16.0)*y*(H-y);

fespace Vh(Th,P2);
fespace Qh(Th,P2);

Vh u1,u2,u1old,u2old,v1,v2, ht;
Qh p,q;

macro umod() (abs(sqrt(u1old^2 + u2old^2))) //
macro beta() (umod == 0.0 ? 1.0 : umod) //
```

```
ht = hTriangle;

macro alpha() (umod*ht/(2.0*nu)) //
//macro tau() ((ht/(2.0*beta))*(sqrt(alpha^2/(9.0+alpha^2)))) //

macro tau() (1.0e-4/(sqrt((2.0*umod/ht)^2 + (4.0*nu/ht^2)^2))) //

solve stokes([u1,u2,p],[v1,v2,q]) =
int2d(Th)(nu*dx(u1)*dx(v1)+nu*dy(u1)*dy(v1)
+ nu*dx(u2)*dx(v2)+ nu*dy(u2)*dy(v2)
- p*q*(0.000001)
- p*dx(v1) - p*dy(v2) - q*(dx(u1)+dy(u2)))
- int1d(Th,2) (p/nu*v1)    // Neumann (7.7)
+ on(1,3,5,6,7,8,u1=0,u2=0) + on(4,u1=inflow,u2=0);

//plot(p,[u1,u2],fill=1, wait=1);

u1old=u1;
u2old=u2;

problem nse ([u1,u2,p],[v1,v2,q]) =
int2d(Th)(nu*dx(u1)*dx(v1) + nu*dy(u1)*dy(v1)
        + nu*dx(u2)*dx(v2) + nu*dy(u2)*dy(v2)
        + u1old*dx(u1)*v1 + u2old*dy(u1)*v1 //    x-convection term
        + u1old*dx(u2)*v2 + u2old*dy(u2)*v2 //    y-convection term
//        - p*q*(0.000001)
       - p*dx(v1) - p*dy(v2) - q*(dx(u1)+dy(u2)))

//first set of three terms (u.grad(v) * (u.grad(u) + grad(p) - nu*D(u)))
+ int2d(Th)
        (tau*((u1old*dx(u1)*u1old*dx(v1)) +
        (u1old*dx(u1)*u2old*dy(v1)) +
        (u2old*dy(u1)*u1old*dx(v1)) +
        (u2old*dy(u1)*u2old*dy(v1)) +

        (u1old*dx(u2)*u1old*dx(v2)) +
        (u1old*dx(u2)*u2old*dy(v2)) +
        (u2old*dy(u2)*u1old*dx(v2)) +
        (u2old*dy(u2)*u2old*dy(v2)) ))

+ int2d(Th)
        (tau*(dx(p)*(u1old*dx(v1)+u2old*dy(v1)) +
        dy(p)*(u1old*dx(v2) + u2old*dy(v2))))
```

```
- int2d(Th)
        (tau*nu*( (dxx(u1)+dyy(u1))*(u1old*dx(v1)+u2old*dy(v1)) +
        (dxx(u2)+dyy(u2))*(u1old*dx(v2) + u2old*dy(v2))))

//second set of three terms (grad(q) * (u.grad(u) + grad(p) -nu*D(u)))
+ int2d(Th)
        (tau*(dx(q)*(u1old*dx(u1)+u2old*dy(u1)) +
        dy(q)*(u1old*dx(u2) + u2old*dy(u2))))
+ int2d(Th)
        (tau*(dx(p)*dx(q) + dy(p)*dy(q)))
- int2d(Th)
        (tau*nu*(dx(q)*(dxx(u1) + dyy(u2)) +
        dy(q)*(dxx(u2) + dyy(u2))))

//third set of three terms (-nu*D(v) *(u.grad(u) + grad(p) - nu*D(u)))

- int2d(Th)
        (tau*nu*( (dxx(v1)+dyy(v1))*(u1old*dx(u1)+u2old*dy(u1)) +
        (dxx(v2)+dyy(v2))*(u1old*dx(u2)+u2old*dy(u2))))

- int2d(Th)
        (tau*nu*( (dxx(v1)+dyy(v1))*(dx(p)) +
        (dxx(v2)+dyy(v2))*(dy(p))))

+ int2d(Th)
        (tau*nu*nu*((dxx(v1)+dyy(v1))*(dxx(u1)+dyy(u1)) +
        (dxx(v2)+dyy(v2))*(dxx(u2)+dyy(u2))))

- int1d(Th,2) (p/nu*v1)    // Neumann (7.7)
+ on(1,3,5,6,7,8, u1=0,u2=0) + on(4,u1=inflow,u2=0);

for (int it=0; it<itmax; it+=1)
{
    cout << "iteration =" << it << endl;
    nse;
 //   cout << "tau =" << tau << endl;
    u1old=u1;
    u2old=u2;
//    plot([u1,u2],fill=1, wait=0);
}

// plot(p,[u1,u2],fill=1, wait=0);
```

```
string outName="square_cylinder.dat";
string titleName="Solution";
include "tecPlotOut.edp";
```