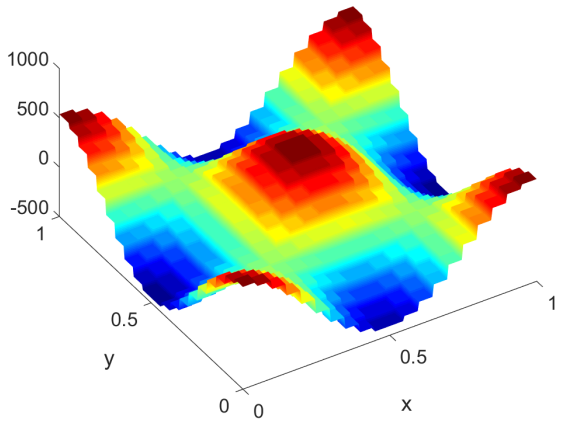
Applications of Discrete Exterior Calculus on Exact Conservation FEM

Feng Ling*

*University of Texas at Austin.
e-mail: FLing@utexas.edu

This paper surveys the applications of discrete exterior calculus on solving linear elliptic PDEs with local conservation features. A code for Poisson's equations and 2D Stoke's flow is constructed in this paradigm, and is tested in comparison with known results to study its numerical properties.

1 Motivation/Introduction



In many applications of Finite Element Methods, such as fluid flows or elasticity, we always desire some kind of conservation property over our domain of interest to comply with the physical laws. Classically, we have the so-called Mixed Finite Element Methods, which solves the problem by simply introducing extra intermediate independent variables. However, such fixes almost feels arbitrary and ad-hoc, not to mention it is still hard to generalize to arbitrary geometry, globally or locally at element level.

Under the formalism of discrete exterior calculus, we can obtain a exact conservation mimetic method that works well with the principles of IsoGeometric analysis, where we can obtain local exact conservation while maintaining complicated geometry and basis functions that aligns with the Computer Aided Design side of engineering. This is an important incentive for further research, as it eliminates the effort needed to reconcile the difference in geometry representation from Computer Aided Designs to analysis, especially since this process often end up costing more resource than the actual analysis. Furthermore, we also want to retain the same geometry because many physical applications such as precise description of the fluid structure interface, non-linear phenomena like transition to turbulence and shell buckling analysis are extremely sensitive to small deviations.

In this paper, we survey the theoretical basis of this exact conservation finite element methods and develop a MATLAB code constructed in this paradigm to test and demonstrate its numerical

properties when solving linear elliptic PDEs. We will also briefly experiment on the scaling properties of this algorithm by translating it to C++/MPI to solve problems at a larger scale/ finer mesh.

2 Complex of Differential Forms on a Manifold

First we will introduce the concept of differential forms. A differntial k-form ω can be defined as a map which associates to each point $x \in \Omega$ a manifold space an element $\omega_x \in \mathrm{Alt}^k T_x\Omega$. Here $T_x\Omega$ is the tangent space of Ω at the point x, and Alt^k stands for alternating k-linear maps to $\mathbb{R}$ the real numbers. One way to quickly get an intuitive grasp on this operation is that a differential k-form spits out a value given k vector fields on some space Ω and its sign alternates depending on the order of the input vector fields[1].

This space admits an obvious basis given a basis (coordinates $x^1, \ldots, x^n$) on our space Ω. Specifically, we know that all differential k-forms are generated by the forms $\mathrm{d}x^I$ where I is a multi-index, a set of coordinate directions in Ω, and $|I| = k$[2].

Given two differential forms α, β, there is a natural binary operation called the wedge product that combines the differential forms, i.e. we have $\wedge : \bigwedge^k \Omega \times \bigwedge^l \Omega \to \bigwedge^{k+l} \Omega$. The key property of the wedge product is that it is anti-symmetric, i.e. $\alpha \wedge \beta = -\beta \wedge \alpha$.

One more structure for the differential forms on a finite dimensional space is that we have a correspondence between k-forms and $(n-k)$-forms[3]. This duality is realized by the Hodge star operator $\star : \bigwedge^k \to \bigwedge^{n-k}$. By definition, we have $\star\star\,\alpha^k = (-1)^{k(n-k)} s\alpha^k$ for any k-form α. Here n is the dimension of our space, and s is the signature of the inner product on our space (sign of the determinant of the inner product tensor). Since we would be working in Euclidean space only, s always equals 1.

From this dependence on some integer value k, one can see that differential forms admit a "$\mathbb{Z}-$grading," i.e. a correspondence somehow between the integers and all the differential forms on some space. If we can define a meaningful operation that transform say any k-form into the next piece $k+1$-form that behaves nicely (what we need here is the notion of exactness i.e. image of the map is contained in the kernel, or the twice composed map is the zero map), we can obtain a so-called "complex" of differential forms.

This operation is dubbed to be the differential operator $\mathrm{d} : \bigwedge^k \Omega \to \bigwedge^{k+1} \Omega$. For a k-form of the form $\omega = f_I \mathrm{d}x^I$, it is defined by the following relation

$$\mathrm{d} f_I \mathrm{d}x^I = \sum_{i=0}^{k+1} \partial_{x^i}(f_i) \mathrm{d}x^i \wedge \mathrm{d}x^I$$

Here I is a multi index and ∂_{x^i} is precisely the usual notion of partial derivative with respect to the coordinate x^i.

The differential operator is linear (over C^∞ functions), satisfies Leibniz rule, commutes with the wedge product, is metric free (commutes with the pullback operator on the underlying

[1] For a detailed treatment of this material still focusing on applications in FEM, see Arnold et. al. (2006).

[2] The basis differential forms can be think of as "covectors" where we have $\mathrm{d}x^i \frac{\partial}{\partial x^j} = \delta^i_j$, where $\frac{\partial}{\partial x^j}$ is the unit tangent vector in x^j direction on the tangent space at x, and δ^i_j the Kronecker delta function.

spaces). Furthermore we can check its exactness since the anti-commutativity of the wedge product ensures that $\mathrm{d}x^i \wedge \mathrm{d}x^i = 0 \, \forall i$, implying that $\mathrm{d} \circ \mathrm{d} = 0$.

But perhaps more interestingly, the reason why one would like to work with differential forms on manifolds is that we have a very nice theorem that generalizes the fundamental theorem of calculus. Namely given an orientation[4], we can define an integration operation on differential forms which behaves nicely globally (it also commutes with pullback of spaces), and satisfy the following theorem

$$\int_\Omega d\omega = \int_{\partial\Omega} \omega$$

.

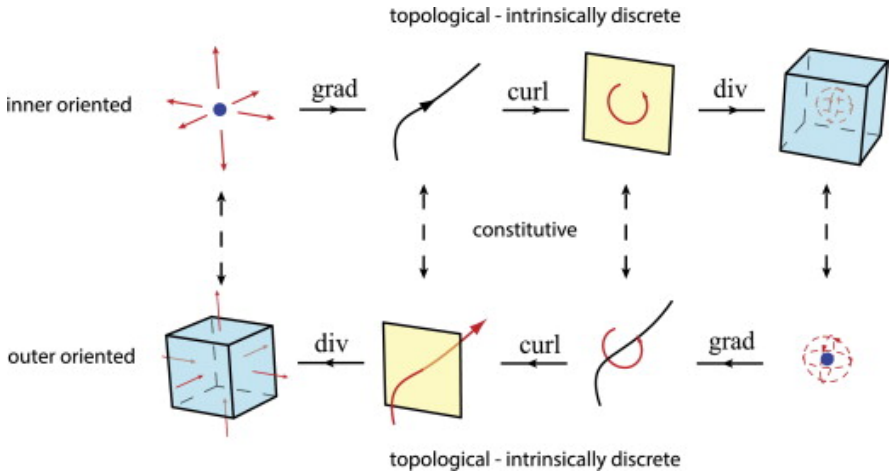


Figure 1: Graphical representation of the cochain complex and differential operators on $\mathbb{R}^3$. [Adapted from Hiemstra et. al. (2013)]

In addition to the differentials, we have another similar operator acting on k-forms, named the codifferential, which reduces the degree of differential forms, i.e. we have $\mathrm{d}^* : \bigwedge^k \Omega \to \bigwedge^{k-1} \Omega$, and it is defined to satisfy the relation

$$\star \mathrm{d}^* = (-1)^k \mathrm{d} \star$$

From the above expression (and the naming convention), it is almost obvious that this operator is a natural dual to our differential. In fact on the L^2 space of functions on Ω equipped with the normal inner product, we have the adjoint relation

$$(\alpha^k, \mathrm{d}^* \beta^{k+1})_\Omega = (\mathrm{d}\alpha^k, \beta^{k+1})_\Omega - \int_{\partial\Omega} \alpha^k \wedge \star \beta^{k+1}$$

This in fact generalizes the integration by parts operation that we usually carry out in ordinary vector calculus.

[4]For our purposes, we can just think of them as the intuitive notion of directions used in our every day Euclidean spaces. For a more rigorous treatment of orientation as a component of the top exterior product of the underlying tangent bundle of our manifold, see the standard reference in differential topology by Guillemin and Pollack.

The last operator we will introduce here is the harmonic Laplacian operator $\Delta : \bigwedge^k \Omega \to \bigwedge^k \Omega$ via the definition

$$\Delta = \mathrm{dd}^* + \mathrm{d}^*\mathrm{d}$$

This operator turns out to be exactly our familiar one in ordinary PDE theory. Note that for any 0-form, since its codifferential is always trivial, Δ reduce to $\mathrm{d}^*\mathrm{d}$. Similarly for any n-form, $\Delta = \mathrm{dd}^*$.

3 Discretization and the Finite Element Method

The above notions are all built on the C^∞ space of functions on our physical space Ω. In reality, we will have to work with finite dimensional function spaces, and will need to discretize the theory.

To obtain finite dimensional equivalent operators, we will do the minimal thing which is projecting down such that all the relations between our operators remain intact (i.e. the diagrams still commute). Luckily, algebraic topology gives us that most of the interesting characteristics of our theory are in fact topological invariants, we in fact do not need to worry about losing too much information.

3.1 Incidence Matrices

Given the topology of the target domain, we can derive the discrete differential operators e.g. discrete `grad`, `curl`, and `div`, easily through difference relations given a choice of orientation. This can be seen easily from Figure 1 above, as the differential operator (e.g. `grad`) is the differential connection between two near by dimensions of simplices (e.g. between nodes and edges). To calculate the actual difference relation, we just follow through the induced orientation and count the number of direction changes. See Figure 2 below.

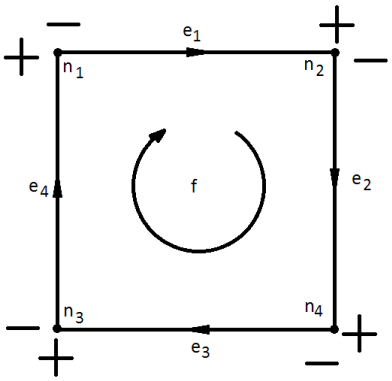


Figure 2: To calculate the difference relation of simplices (e.g n_3 and n_4) on a given simplex σ (e.g. e_2), we just count whether the simplices in question are aligned if we induced the orientation of σ there. If yes, we put down $+1$. If no, we write -1. If the simplex is not directly connected to σ, we know the answer is zero. (So we get 0 for n_3 and $+1$ for n_4.)

Completing this process will give us the appropriate matrix operators between each adjacent dimensions. These are called the *incidence Matrices*, and they are usually denoted $E_{k,k+1}$ for some k. The desired differential operators is in fact $D_{k+1,k} = E_{k,k+1}^T$ the transpose.

3.2 Basis Functions

The idea behind choosing basis functions to approximate functions on each element is exactly as in conventional Galerkin finite element methods. But in order to utilize our theory and keep things metric free, we need the basis functions to not obstruct the commuting properties of our differential operators, i.e. if we choose a basis for 0-forms, we need to make sure their derived basis on higher forms (e.g. edges) works with our still satisfy same commuting diagrams with the discrete differential operators.

By Hiemstra et. al. (2013), it is apparent that such basis are exactly the integral ones. In other words, the basis functions need to be *cochain interpolants*. For nodal interpolating basis functions like the classical Lagrange polynomials, this means that we need to scale its derivative functions such that they integrates to unity at each edge to obtain the desired basis functions for the 1-simplices (edges). As long as our basis satisfies this integral condition[5], we can always build a compatible discretization framework. This means that more CAD friendly functions such as B-splines or NURBs can be used directly for finite element analysis, bringing a critical piece to the puzzle of IsoGeometric analysis.

For multidimensional basis, we can simply use tensor products of univariate nodal and edge functions. For the code and solutions presented below, we use hat functions $N(x)$ for nodes and scaled constant functions $M(x)$ for edges (i.e. first order Lagrangian interpolant adjusted to be integral).

3.3 Local Mass Matrix and Assembly

The assembly of the mass matrix goes exactly same as in conventional FEM. We use quadrature rules to obtain the integral of our basis function over each element using a pull back map from the physical domain to the parent element $[-1, 1] \times \cdots \times [-1, 1] \in \mathbb{R}^n$.

3.4 Boundary Conditions

Similar to conventional FEM, the boundary conditions will be applied in two different fashion. For strong (Dirichlet) boundary conditions, we employ the same tactic as in FEM and directly substitute solutions into the matrix equations to reduce the degree of freedom. For weak (Natural/Neumann) conditions, we calculate the appropriate right hand side load vectors using our generalized integration by parts, also completely analogous to conventional FEM.

4 Benchmark Problems and Solutions

In this section, we will focus on how to exactly formulate our exact conservation FEM solutions to several benchmark problems.

[5] see Equation (49) in Hiemstra et. al. (2013).

4.1 0-form Poisson's Equation

we would like to find the solution to the equation

$$\Delta u = \mathrm{d}^*\mathrm{d}u = f$$

where u, f are 0-forms on 2D domain Ω. Here the other part of the Laplacian dd^* is omitted because we have $\mathrm{d}^*\alpha = 0$ for any 0-forms.

This problem can be solved directly.

Taking inner product with test functions w (0-form), we get

$$(w, \mathrm{d}^*\mathrm{d}u)_\Omega = (w, f)_\Omega$$

Performing integration by parts for differential forms,

$$(\mathrm{d}w, \mathrm{d}u)_\Omega - \int_{\partial\Omega} w \wedge \star\mathrm{d}u = (w, f)_\Omega$$

Next we approximate the forms in finite dimension spaces

$$u_h(\xi^1, \xi^2) = \sum_{i,j} \mathrm{u}_{i,j} P_{i,j}(\xi^1, \xi^2)$$
$$w_h(\xi^1, \xi^2) = \sum_{i,j} \mathrm{w}_{i,j} P_{i,j}(\xi^1, \xi^2)$$

where $P_{i,j} = N_i(x^1)N_j(x^2)$ are the basis functions. Here $\mathrm{u}_{i,j}, \mathrm{w}_{i,j}$ are the nodal/edge values for each finite dimensional projections.

Substituting and rearranging terms give us

$$-\mathrm{D}_{(0,1)} \cdot \mathrm{M}_{(1)} \cdot \mathrm{D}_{(1,0)} u = \mathrm{M}_{(0)}(f)$$

where $\mathrm{D}_{(1,0)}$ is the discrete matrix representation of curl and $\mathrm{D}_{(0,1)} = \mathrm{D}_{(1,0)}^T$ is the dual operator. Furthermore, $\mathrm{M}_{(1)}$ is the mass matrix for the 1-form basis functions integrated using (Gaussian) quadrature over the parent element domain. Specifically,

$$\mathrm{M}_{(1)} = \sum_{i,j} \int_\Omega L_i L_j \det J dx^1 \wedge dx^2$$

where $L^1_{i,j} = M_i(\xi^1)N_j(\xi^2)$ and $L^2_{i,j} = N_i(\xi^1)M_j(\xi^2)$ are the basis functions on edges. Note that we are getting mass matrix for 1-forms because our final inner product is $(\mathrm{d}\omega, \mathrm{d}u)_\Omega$, where both arguments are 1-forms.

4.2 2-form Poisson's Equation

The theory of differential forms interestingly allows us to generalize the pedestrian Poisson's equation into a slightly less familiar form. Here we would like to find the solution to the equation

$$\Delta u = \mathrm{d}\mathrm{d}^* u = f$$

where u, f are 2-forms on a 2D domain Ω. Now the other part of the Laplacian $\mathrm{d}^*\mathrm{d}$ is omitted because we have $\mathrm{d}\alpha = 0$ for any 2-forms in a 2-dimensional space.

Introducing the intermediate 1-form p

$$\Rightarrow \begin{cases} \mathrm{d}^* u = p \\ \mathrm{d}p = f \end{cases}$$

Taking inner product with test 1- and 2-forms v, w, we get

$$\begin{cases} (v, \mathrm{d}^* u)_\Omega = (v, p)_\Omega \\ (w, \mathrm{d}p)_\Omega = (w, f)_\Omega \end{cases}$$

Performing integration by parts

$$\begin{cases} (\mathrm{d}v, u)_\Omega - \int_{\partial\Omega} v \wedge \star u = (v, p)_\Omega \\ (w, \mathrm{d}p)_\Omega = (w, f)_\Omega \end{cases}$$

Next we approximate the forms in finite dimension spaces

$$\begin{aligned} u_h(\xi^1, \xi^2) &= \sum_{i,j} \mathrm{u}_{i,j} S_{i,j}(\xi^1, \xi^2) d\xi^1 \wedge d\xi^2 \\ p_h(\xi^1, \xi^2) &= \sum_{i,j} \mathrm{p}^1_{i,j} L^1_{i,j}(\xi^1, \xi^2) d\xi^1 + \sum_{i,j} \mathrm{p}^2_{i,j} L^2_{i,j}(\xi^1, \xi^2) d\xi^2 \\ v_h(\xi^1, \xi^2) &= \sum_{i,j} \mathrm{v}^1_{i,j} L^1_{i,j}(\xi^1, \xi^2) d\xi^1 + \sum_{i,j} \mathrm{v}^2_{i,j} L^2_{i,j}(\xi^1, \xi^2) d\xi^2 \\ w_h(\xi^1, \xi^2) &= \sum_{i,j} \mathrm{w}_{i,j} S_{i,j}(\xi^1, \xi^2) d\xi^1 \wedge d\xi^2 \end{aligned}$$

where $P_{i,j} = N_i(\xi^1) N_j(\xi^2)$, $L^1_{i,j} = M_i(\xi^1) N_j(\xi^2)$, and $L^2_{i,j} = N_i(\xi^1) M_j(\xi^2)$ are the basis functions. Here $\mathrm{u}_{i,j}, \mathrm{w}_{i,j}, \mathrm{p}^k_{i,j}, \mathrm{v}^k_{i,j}$ are the nodal/edge values for each finite dimensional projections.

Here we can choose an appropriate reordering of the basis function for both P_i and L_i.

Substituting and rearranging terms give us

$$\begin{cases} \mathrm{D}_{(1,2)} \mathrm{M}_{(2)} \cdot \mathbf{u} - \mathrm{M}_{(1)} \cdot \mathbf{p} & = \int_{\partial\Omega} v \wedge \star u \\ \mathrm{M}_{(2)} \mathrm{D}_{(2,1)} \cdot \mathbf{p} & = \mathrm{M}_{(2)}(f) \end{cases}$$

where $\mathrm{D}_{(1,0)}$ is the discrete matrix representation of divergence and $\mathrm{D}_{(0,1)} = \mathrm{D}^T_{(1,0)}$ is the dual operator. And we have $\mathrm{M}_{(i)}$ being the mass matrix for the i-form basis functions integrated using Gaussian quadrature as usual. Specifically,

$$\mathrm{M}_{(1)} = \sum_{i,j} \int_\Omega L_i L_j g^{i,j} \det J d\xi^1 \wedge d\xi^2$$

$$\mathrm{M}_{(2)} = \sum_{i,j} \int_\Omega S_i S_j \frac{1}{\det J} d\xi^1 \wedge d\xi^2$$

$$\mathrm{M}_{(0)}(f) = \sum_{i} \int_\Omega f P_i^k \det J d\xi^1 \wedge d\xi^2$$

The determinant of the jacobian of the pullback between parent element and physical element falls out just like what would have happen under conventional vector calculus calculations, but with more clarity and grace. :)

4.3 Stoke's Flow

Our final problem of interest is to find the solution to the 2D incompressible stationary stokes flow problem in vorticity-velocity-pressure differential forms on a 2D domain Ω

$$\begin{aligned} \Delta u + \mathrm{d}^* p &= f \\ \mathrm{d}u &= 0 \end{aligned}$$

where u is the velocity 1-form, f 1-form of body forces. p 2-form pressure.

If we introduce a vorticity 0-form such that $\omega = \mathrm{d}^* u$, combined with the fact that $du = 0$ reduces the Laplacian to $\Delta = \mathrm{d}\mathrm{d}^*$, and $\mathrm{d}^* = (-1)^k \star^{-1} \mathrm{d} \star$, we have

$$\begin{aligned} \mathrm{d}\omega + \star^{-1} \mathrm{d} \star p &= f \\ \mathrm{d}u &= 0 \\ \omega &= - \star^{-1} \mathrm{d} \star u \end{aligned}$$

And the Hodge star on the basis forms are

$$\begin{aligned} \star 1 &= \mathrm{d}x \wedge \mathrm{d}y \\ \star \mathrm{d}x &= \mathrm{d}y \\ \star \mathrm{d}y &= -\mathrm{d}x \\ \star(\mathrm{d}x \wedge \mathrm{d}y) &= 1 \end{aligned}$$

Given solutions

$$\begin{aligned} u &= \mathrm{u}^x \mathrm{d}x + \mathrm{u}^y \mathrm{d}y \\ p &= \mathrm{p}\, \mathrm{d}x \wedge \mathrm{d}y \end{aligned}$$

we can calculate the corresponding vorticity and body force by simple substitution.

Substituting into the stokes system gives us

$$\begin{aligned}
f &= -\mathrm{d}\star^{-1}\mathrm{d}\star u + \star^{-1}\mathrm{d}\star p \\
&= -\mathrm{d}\star^{-1}\mathrm{d}(\mathrm{u}^x\mathrm{d}y - \mathrm{u}^y\mathrm{d}x) + \star^{-1}\mathrm{dp} \\
&= -\mathrm{d}\star^{-1}(\partial_x\mathrm{u}^x\mathrm{d}x\wedge\mathrm{d}y - \partial_y\mathrm{u}^y\mathrm{d}y\wedge\mathrm{d}x) + \star^{-1}(\partial_x\mathrm{p}\mathrm{d}x + \partial_y\mathrm{p}\mathrm{d}y) \\
&= -\mathrm{d}(\partial_x\mathrm{u}^x + \partial_y\mathrm{u}^y) - \partial_x\mathrm{p}\mathrm{d}y + \partial_y\mathrm{p}\mathrm{d}x \\
&= -\partial_x\partial_x u^x\mathrm{d}x - \partial_y\partial_x u^x_{xy}\mathrm{d}y - \partial_x\partial_y u^y\mathrm{d}x - \partial_y\partial_y u^y\mathrm{d}y - \partial_x\mathrm{p}\mathrm{d}y + \partial_y\mathrm{p}\mathrm{d}x \\
&= (-\partial_y\partial_x u^x_{xy} - \partial_y\partial_y u^y - \partial_x\mathrm{p})\mathrm{d}y + (-\partial_x\partial_x u^x - \partial_x\partial_y u^y + \partial_y\mathrm{p})\mathrm{d}x
\end{aligned}$$

Note that we have $\omega = -(\partial_x\mathrm{u}^x + \partial_y\mathrm{u}^y)$ as an intermediate result.

To put this problem into finite elements formulation, we can again do the usual inner product with test function in L^2 space and then apply suitable integration by parts, just as above. The result is a system

$$\begin{aligned}
-(\alpha,\omega)_\Omega + (\mathrm{d}\alpha, u)_\Omega &= \int_{\partial\Omega}\alpha\wedge\star u \\
(\beta,\mathrm{d}\omega)_\Omega + (\mathrm{d}\beta, p)_\Omega &= (\beta, f)_\Omega \\
(\gamma,\mathrm{d}u)_\Omega &= 0
\end{aligned}$$

where α, β, γ are our test 0-, 1-, 2- forms. Discretize we get

$$\begin{aligned}
-\mathrm{M}_{(0)}\cdot\boldsymbol{\omega} + \mathrm{D}_{(0,1)}\mathrm{M}_{(1)}\cdot\mathbf{u} &= \int_{\partial\Omega}\alpha\wedge\star u \\
\mathrm{M}_{(1)}\mathrm{D}_{(1,0)}\cdot\boldsymbol{\omega} + \mathrm{D}_{(1,2)}\mathrm{M}_{(2)}\cdot\mathbf{p} &= \mathrm{M}_{(1)}(f) \\
\mathrm{M}_{(2)}\mathrm{D}_{(2,1)}\cdot\mathbf{u} &= 0
\end{aligned}$$

where as usual, $D_{(k,l)}$ are our discrete differential operators relating simplices of dimension k simplices to that of dimension l, and M_i are the mass matrices for $i-$forms. Note that the boundary term at the right hand side of the first equation is weakly enforcing the boundary tangential velocities.

5 Results and Discussion

This section will discuss the results of solving specific cases (specified forcing functions) of the previously discussed problems using a MATLAB implementation of our FEM under various mesh resolutions.

5.1 Manufactured Solutions for Poisson's Equations

Using the forcing function $F = -2\pi^2\cos\pi x\cos\pi y$, we can solve both the 0-form and 2-form Poisson's equations using the forms $f = F$ or $f = F\mathrm{d}x\wedge\mathrm{d}y$ respectively.

The numerical solution for the potential u is then compared with the analytical one, $u = \cos \pi x \cos \pi y$ (as well as the intermediate 1-form $p = -\pi \cos \pi y \sin \pi x \mathrm{d}x - \pi \cos \pi x \sin \pi y \mathrm{d}y$ for the 2-form Poisson's equation) at various grid resolutions to determine the convergence characteristics of our method.

In Figure 3 below, we can see that the 0-form equation have a convergence rate of h^{-2} where h is the side length of the rectangular element used in our grid.

For the 2-form equation (see Figure 4), the additional solution for the intermediate 1-form p resulted in a convergence rate of h^{-1} for both u and p. All of which are consistent with our theory[6].

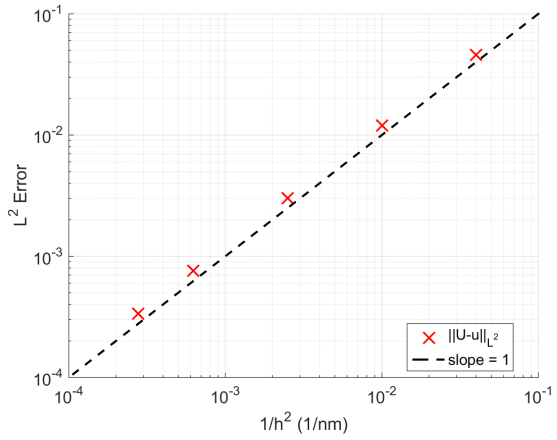
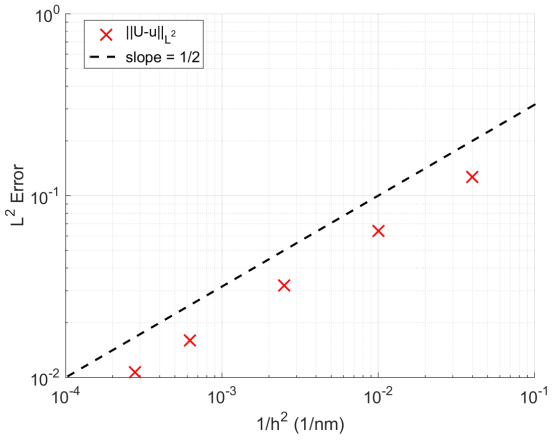
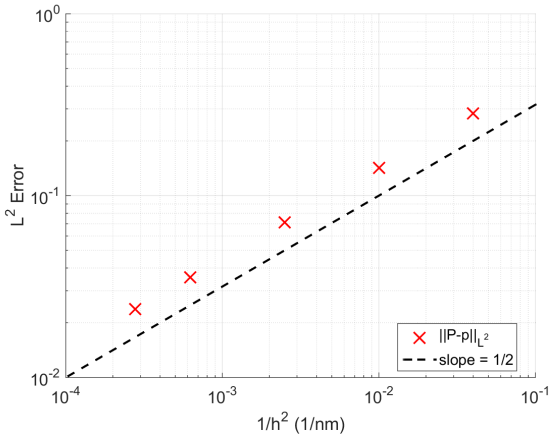


Figure 3: L^2 error convergence results for u from 0-form Poisson's Equation



(a) L^2 error for u



(b) L^2 error for p

Figure 4: L^2 error convergence results from 2-form Poisson's Equation

[6]The seemingly decrease in error convergence rate when we switch from 0- to 2-form is a result of the order of the basis function in use. For nodes (0-forms, such as u in 0-form Poisson's equation), we have bivariate linear functions, which is in fact a tensor product of linear polynomials resulted in some quadratic terms. For edges (1-forms, such as p) and faces (2-forms, such as u in 2-form Poisson's equation solution), we only have linear and constant basis functions, respectively.

5.2 Lid-driven cavity flow

If we assume trivial boundary flow flux and body forces along with a uniform tangential flow on the upper edge of our (unit) square domain Ω to our linear Stoke's flow problem, we obtain the most rudimentary form of lid-driven cavity flow problem with trivial Reynold's number.

In Figure 5 below, it is evident that our solution approximates a convergent solution stably even at very coarse grid resolutions to the best of the abilities of the chosen low-order basis functions. If we have used more complicated basis such as the B-splines, as done in Hiemstra et. al. (2013), we can see even better approximation results at lower orders. And comparing with the literature (Sahin and Owens, 2003) indicates that the solution presented here are indeed converging to the correct result.

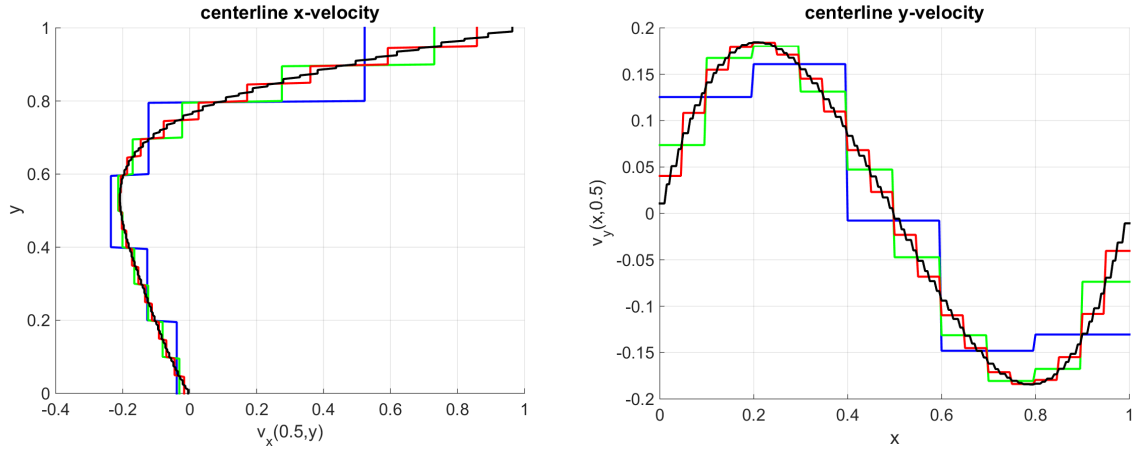


Figure 5: Lid-driven cavity flow solution convergence
The grid resolution used here are 5x5, 10x10, 20x20, and 80x80 shown in blue, green, red, and black respectively.

The actual solutions obtained at various grid resolutions (5x5, 10x10, 20x20, 80x80) are presented in Figure 6 to 9 on the next few pages. There are several interesting characteristics In particular, one should note that for the bottom right subfigure where the divergence of the velocity field are plotted, we are consistently seeing very minimal amount of noise even locally around each element regardless of the coarseness of our grid resolution. This is the key point of our exact conservation finite element methods: we can conserve some quantity ($\nabla \cdot \mathbf{v}$ in this case) at a local level in a very robust sense.

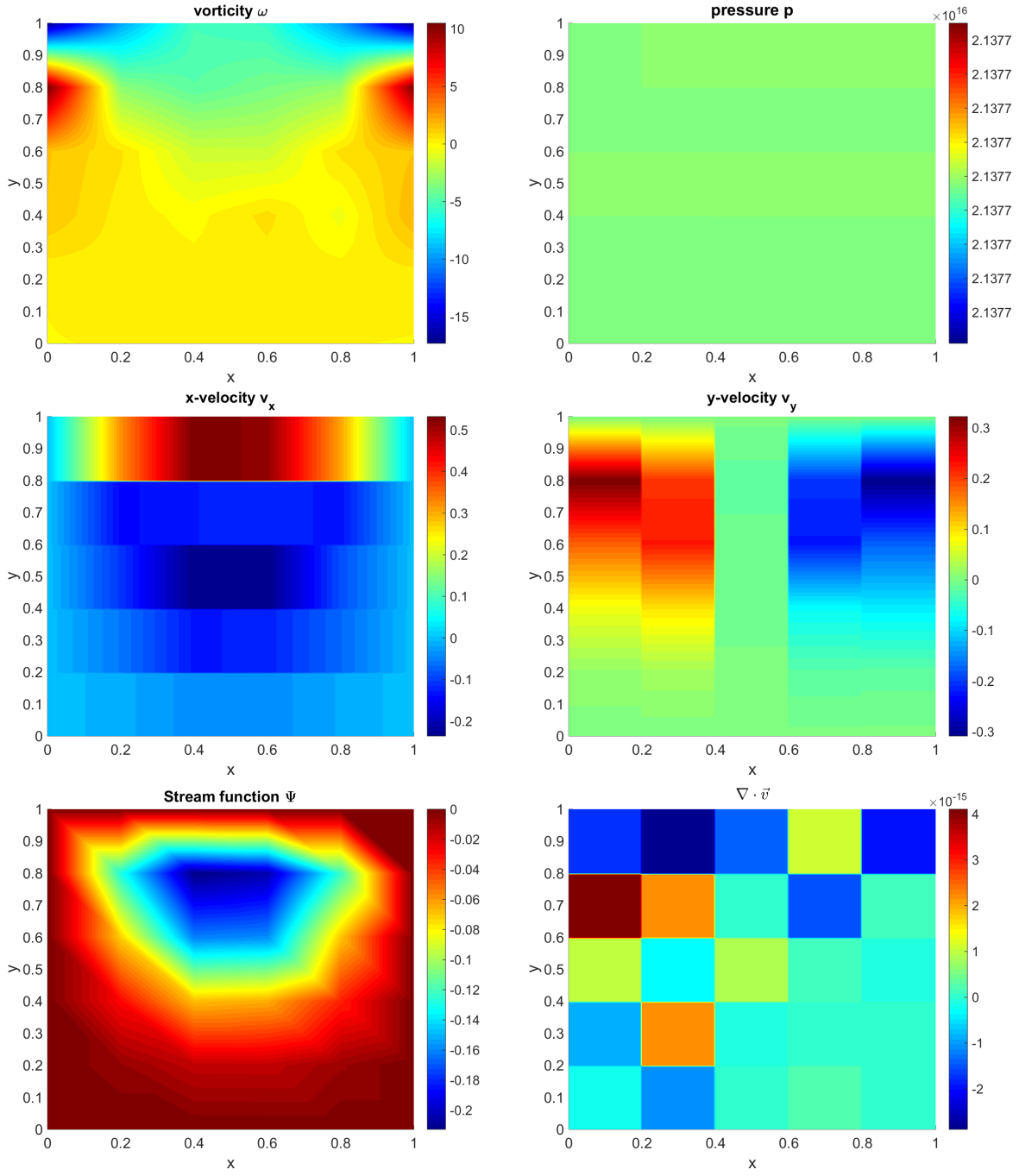


Figure 6: Lid-driven cavity flow solution on 5x5 grid

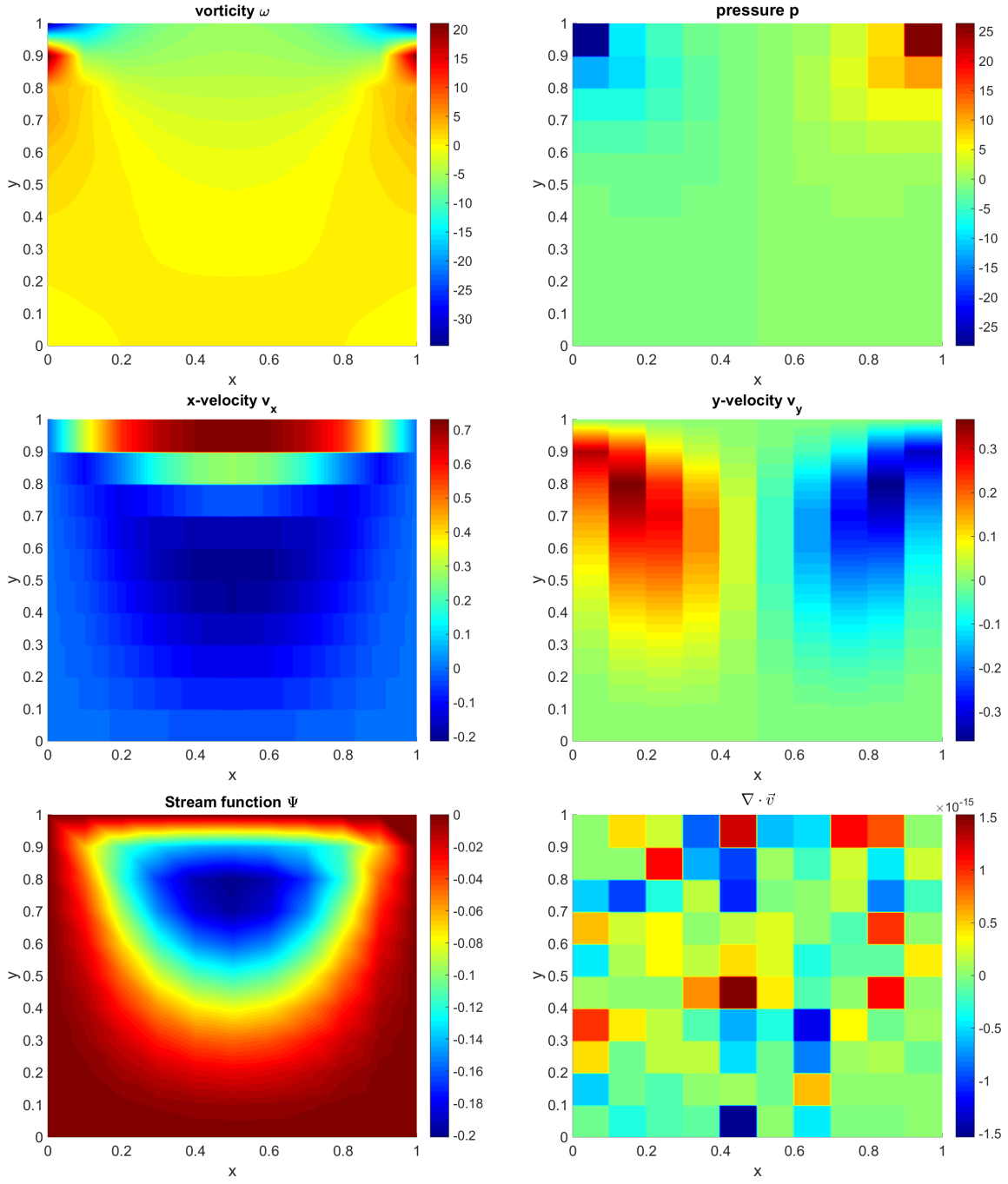


Figure 7: Lid-driven cavity flow solution on 10x10 grid

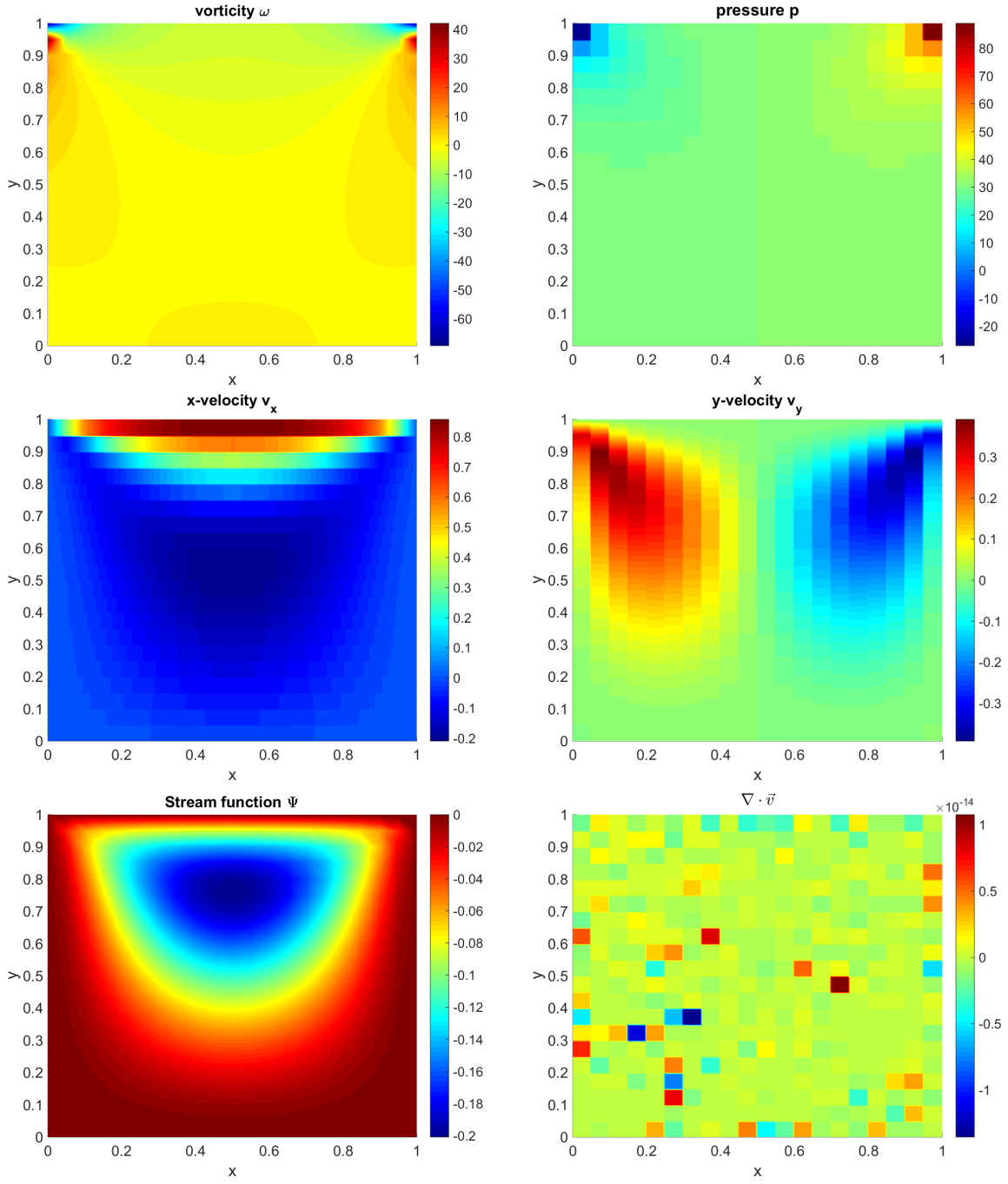


Figure 8: Lid-driven cavity flow solution on 20x20 grid

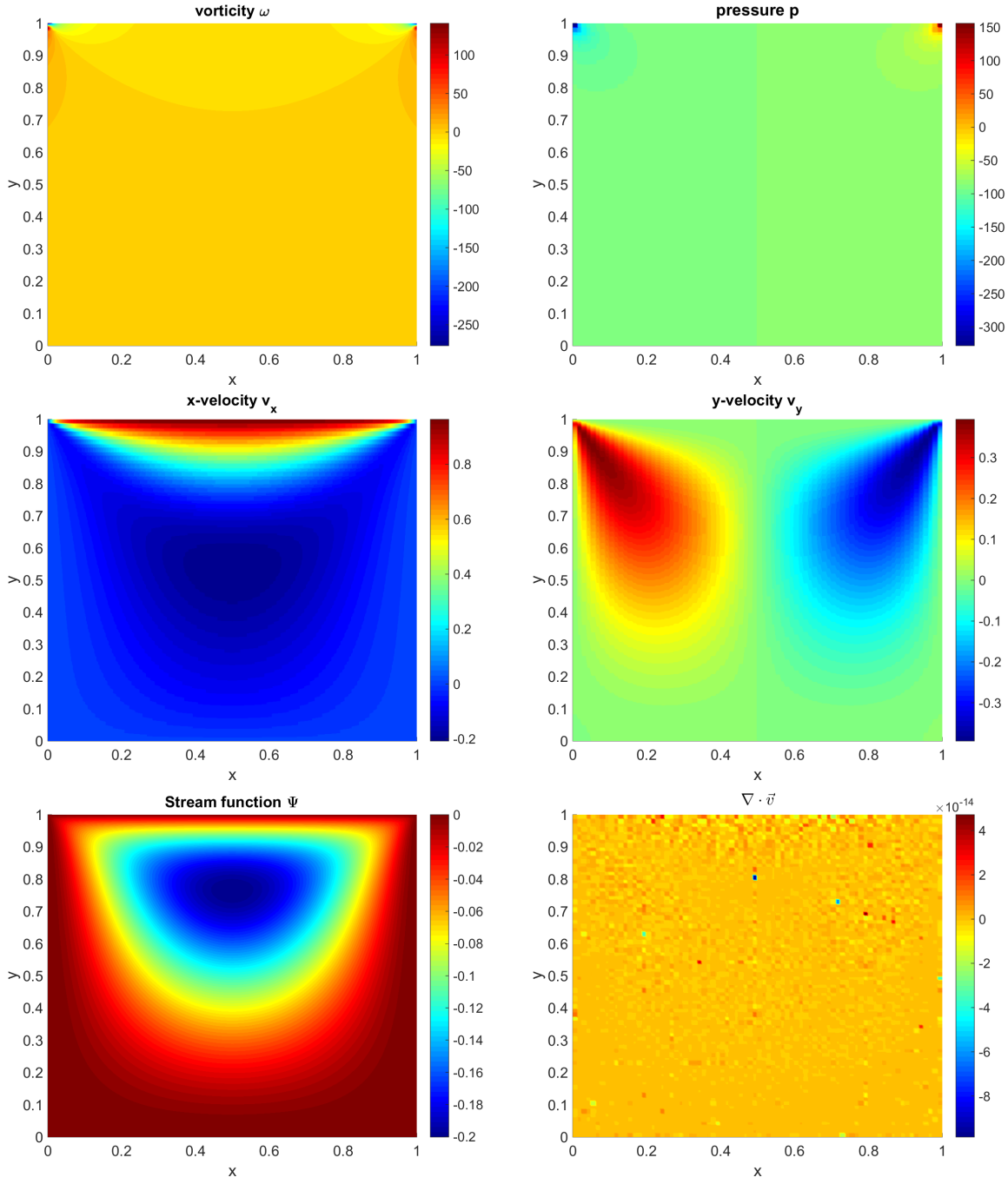


Figure 9: Lid-driven cavity flow solution on 80x80 grid

5.3 Parallelization properties

In order for any FEM algorithms to become practical, it is very important to know that the methods can be successfully extended for parallel processing so that we can tackle problems

at larger scale. Thus we analyzed the serial MATLAB code used to solve the 2-form Poisson's equation above for its performance to develop a parallel version coded in C++ using MPI.

The serial program is separated into seven different stages, namely the initialization and setup (construction of initial discretized domain), computing of the difference matrices (discrete differential operators), building the local (element-wise) mass matrices (for all 0-, 1-, and 2-forms in our 2D case), insertion of local mass matrices into the global mass matrices, other activities for the assembly process (this is mostly the conversion of the full matrices into their sparse representations), enforcement of the 2-form boundary condition, and finally solving the resulting sparse linear system. The performance of each stage is measured separately (See Figure 10 below) to guide us in designing an effective parallel solutions.

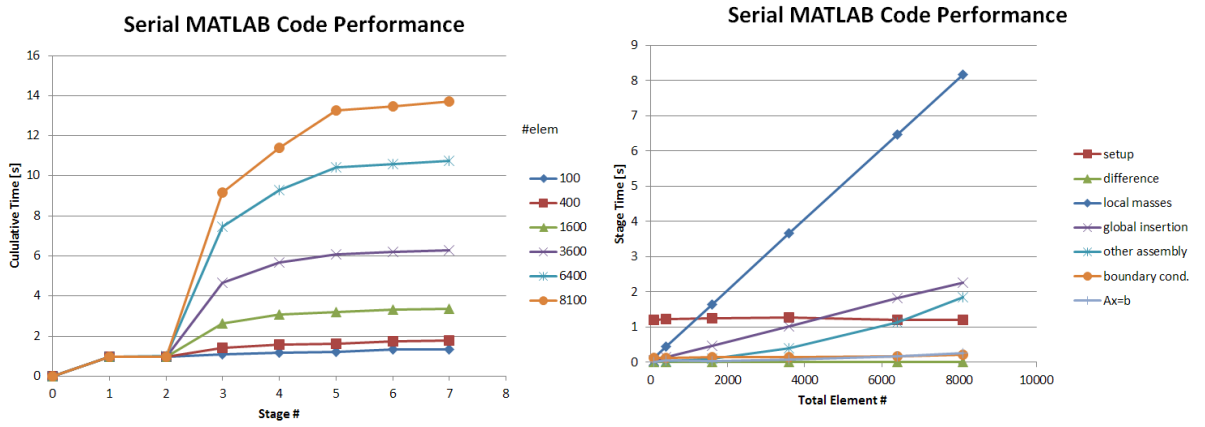


Figure 10: Time analysis of the serial MATLAB program
Running on Intel Core i5-4300M 2.60GHz QuadCore

From the above analysis, it is clear that the initialization, computation of difference matrices, enforcing of boundary conditions, and the final solution of sparse linear system all have almost constant time performance regardless of the grid resolution[7]. Thus it is most wise for us to first test and improve the situation of local mass matrix computation and insertion to the global mass matrices using parallelization techniques[8].

There are many ways to organize our construction of the parallelized computation of mass matrices. First, in a shared memory model where each task/thread has direct access to the final global matrices waiting to be assembled (using *openMP*), we can simply subdivide the total domain into different threads and then have each thread insert their result into the global

[7]This situation has some theoretical basis, so it is not too implementation (MATLAB) dependent. Here the initialization is more or less the computation of node coordinates in our domain, which is only copying two 1D array in different directions to get a 2D grid. The discrete differential operators are represented by highly structured sparse matrices that only depends on the topology of our space. The boundary conditions only grow on the order of $\mathcal{O}(n+m)$, where n, m are number of elements in each x-, y- directions, while the mass matrices grow on the order of $\mathcal{O}(nm)$. The solution of the final sparse linear system could be fast due to the additional structure resulted in our elementary assumptions of the toy problem, as well as the highly optimized nature of MATLAB.

[8]Here the sparse matrix conversion issue is unfortunately ignored due to the scope (time) of this project.

matrices. The problem with this approach is that we might have a race condition between adjacent elements, since adjacent elements share common nodes and edges. One way to combat this problem is to make sure that each thread only operates on non-adjacent elements at any given time. One way to achieve this is to subdivide the $n \times m$ grid into rectangular regions based on the total number of threads. Then if we iterate on each region for each thread, they would never try to write to the same element of the global matrix (See Figure 11 below).

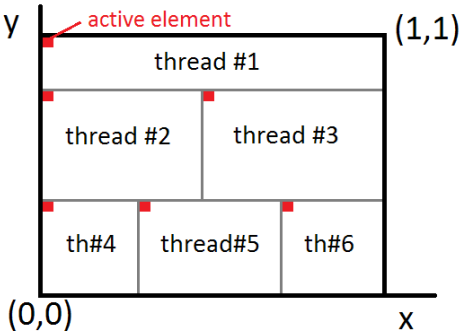


Figure 11: Example domain subdivision

Under the multi-executables paradigm (i.e. *MPI*), an immediate naïve method would be similar to the bulk scheduling done in Lab 10 employing the MPI collectives. Specifically, we have a root process in charge of task scheduling and have access to the final global mass matrices. Then when there are free processors, the root processor would send the element coordinate in need of computation to all the free processors until none is left. Once the work is done, we can collect the local mass data to write the final matrices and repeat the cycle. The first trouble is the same as before: we might have to use reduce (padded with zeros for irrelevant nodes/edges) if we send out consecutive elements to free processors in one cycle. Of course we can bypass the issue just like before.

Next we have to worry about task scheduling issues. We need to make sure that we do not manually serialize our program by forcing too many worker processor to wait before the new round of tasks are generated and issued. i.e. we need to make sure each worker receive enough work at each scheduling cycle. In a word, we need to balance our program so that it is neither purely computation limited (root global matrix insertion processor stays idle) nor "bandwidth" (IPC) limited (local mass matrix computing worker processors stays idle).

Consequently, we need to determine whether the transmission of all the entries of each local mass matrices would be too costly. For a rectangular element with the bivariate hat functions for nodal basis, linear slopes for edges, and constants for faces, we need to consider 4 nodes, 2 x-direction edges, 2 y-direction edges, and 1-face. This yields a 4×4 matrix, four 2×2 matrices, and a scalar[9], a total of 33 doubles per element in six contiguous parts[10]. This means that we

[9]For higher degree basis functions, which might be required for certain applications, these numbers increase quickly. e.g. for degree 2, we require one 9×9, four 6×6 matrices, and four scalars, which sums to 229 doubles in 6 parts.

[10]In implementing dynamic 2D arrays, we need to pay attention on how to allocate the memory in order to

have a complexity of $P(4\alpha + 25\beta)$ for each cycle. This in fact is quite high given that we are making many assumptions to simplify our local computation, which leads to a drop in the γ term. One solution to help balance our efficiency is to consider computing multiple adjacent elements on each worker processor at once and transmit the resulted partial global mass matrices to the root process in charge of file I/O streams at once. Yet another solution would be to mimic the shared-memory model by creating local copies of the global mass matrices from the beginning, and only use a reduce procedure to communicate and collect the final result.

There is also the problem of how to subdivide the entire work domain. For simplicities' sake, we just use a sequential coloring of a given step size when assigning separate them into strips of length $n \times \Delta m$ to accommodate the row-major format of matrix allocation in C. But our discussions should hold their grounds in general.

Following the above discussions, a MPI Overall, the MPI parallelization improved the execution speed drastically. However, a major concern is that this speed up might be mostly a result of the translation of the programming languages rather than that of parallelization itself[11]. This could happen since the more rigid structure of a compiled language like C++ requires many functions to be reduced to its most rudimentary form. But at least, a rewritten MATLAB algorithm, in a rather strict semantical sense, equivalent to our MPI code still achieves a slow runtime (see Figure 12).

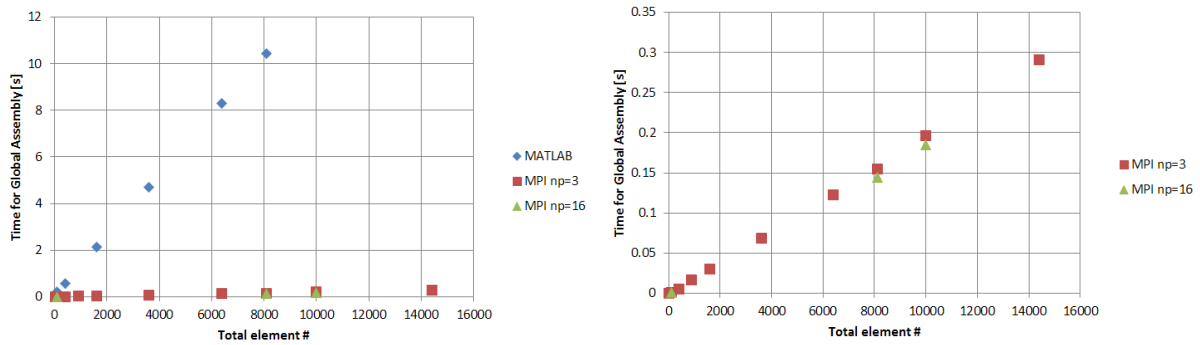


Figure 12: MPI parallel assembly result

Another important observation, demonstrated by the graph on the right, is that as our elementary communication complexity analysis showed, the increase in communication traffic between the worker processors and the master root processors drastically prohibited further speed up when we increase the number of processors from 3 to 16. This result indicates that we indeed have a dominating α and β terms, which scales at least linearly with the number of processors and occupy the total wall clock time.

We should finally note that, to avoid reinventing the wheels, for any implementation with real life applications in mind, we should take advantage of the highly developed modern libraries and

have correct contiguous packing of our data to take full advantage of MPI send/receive commands. The idea is that for every 2D array `A[i][j]`, we create a one dimensional array `B` of size $n_1 \times n_2$ elements and have `A[i]` point to the appropriate chucks of `B`. See *Ge Bolai, 2008*.

[11]For example, most of the (small) matrix vector products are reduced to simple scalar multiplications of rearranged elements of 1D arrays.

techniques, the prominent ones being the PETsc library[12], sparse matrix storage and computation techniques, BLAS-type fast matrix library, plus hybrid solutions between openMP, MPI, and even MATLAB. It is truly a prototype not yet capable of fully representing the parallel computing world, and a full scale analysis on its scalability should be done with more time.

6 Conclusion

We have shown that under the theory of differential forms and discrete exterior calculus, we can reformulate the mixed Galerkin Finite Element Methods to solve more general linear elliptic PDEs with excellent local conservation properties even at very coarse grid and very simple basis functions. The version of the exact conservation FEM implemented and studied can also be suitably extended to use on curved geometries with curved basis functions.

Furthermore, it is shown that our algorithm can also be reasonably parallelized similar to the conventional methods. However, to really investigate how it would behave in parallel setting, we do need to test our methods under a more difficult domain geometry and topology, element shapes, and basis functions.

7 Next Steps

There are many more directions for further explorations on this subject. In the spirit of the original paper by Hiemstra et. al. (2013), one extension to our investigation is to implement our exact conservation FEM with curved B-spline basis functions for more complicated geometry. It would also be desirable to actually implement everything for 3D flow problems, which might require further translation and parallelization of our code into C++/MPI without restricting its functionalities.

Another direction would be to explore how well would our method work with different parent element shapes and basis, such as the popular triangular meshes with Whitney basis functions.

It would also be interesting to see if we can further develop the theory to incorporate more complicated geometry processing research, such as ones about discrete connections on surfaces by Crane and other geometry processing research laboratories.

Last but not least, it would greatly satisfy the author himself if this method can be implemented to solve (at least a simplified axial-symmetric version) of the reentry heat flow problem arisen in his senior design class. Of course this would require first exploring the ability to adapt our method to compressible flows or even solutions of certain nonlinear problem.

8 Acknowledgement

I would like to thank René Hiemstra for his utmost patience on teaching me the fundamentals about this subject and helping me code and debug the algorithms despite his own busy schedule.

[12] e.g. in *Victor Eijkhout, Intro to HPC, pg. 443* we see that this global matrix computed in small chunks locally alogrithm can be very easily implemented in PETsc.

References

1. René Hiemstra, D. Toshniwala, R.H.M. Huijsmansb, and M.I. Gerritsma, (2013) "High Order Geometric Methods with Exact Conservation Properties," *Journal of Computational Physics* **257B**, pp. 1444-1471

2. René Hiemstra, (2011) "IsoGeometric Mimetic Methods," *TU Delft* MS. Sci. Thesis.

3. Douglas N. Arnold, Richard S. Falk, Ragnar Winther, (2006) "Finite Element Exterior Calculus, Homological Techniques, and Applications," *Acta Numerica*, pp. 1-155

4. Mehmet Sahin and Robert G. Owens, (2003) "A novel fully implicit nite volume method applied to the lid-driven cavity problem," *International Journal for Numerical Methods in Fluids* **42**:5777 (DOI: 10.1002/fld.442)

5. Keenan Crane, (2013) "Digital Geometry Processing with Discrete Exterior Calculus," *SIGGRAPH Course Notes* Updated March 2015.

6. Victor Eijkhout, Edmond Chow, Robert van de Geijin, (2015), "Introduction to High Performance Scientific Computing," 2nd edition.

7. Ge Baolai, (2008) "Parallel Numerical Solution of PDEs with Message Passing," *The University of Western Ontario*