Vortex Dominated Flows: A High-Order, Conservative Eulerian Simulation Method

Josh Bevan

April 28, 2015

Abstract

A high-order, conservative Eulerian method will be presented for the simulation of vortex dominated inviscid fluid flows. The primitive variable incompressible Euler equations are recast in the velocity-vorticity form to explicitly enforce conservation of vorticity. The advection of the vorticity is then calculated via a two-step process: the velocity field is first determined by evaluation of the Biot-Savart integral, and then a line-based discontinuous Galerkin (DG) Eulerian spatial discretization scheme is applied to accurately advect the vorticity field. The accuracy and convergence of this method is examined for test cases where an analytical solution exists, as well as more challenging test cases which lack an analytical solution. The influence the velocity field discretization has on the performance of the method is of particular interest.

Acknowledgments

Contents

List of Tables

List of Figures

1 Introduction

1.1 Velocity-Vorticity equation

Direct solution of Navier-Stokes is impractical for many fluid problems and where possible simplifications should be made; one such possibility is for vortex dominated flows. It is possible to recast Navier-Stokes from a primitive variable form (u, v, p) to a velocity-vorticity (u, v, ω) form. This has several advantages: explicit conservation of vorticity, elimination of pressure, and reduction of required simulated DOFs to just those that form the vorticity support. For inviscid and incompressible flows the vorticity transport equation is:

$$\frac{\partial \omega}{\partial t} + u \cdot \nabla \omega - \omega \cdot \nabla u = S(x, t) \tag{1.1}$$

Traditionally simulation techniques take advantage of Helmholtz and Kelvin's theorems; vorticity exists as material surfaces, lines, etc. and therefore a Lagrangian approach is natural. Typically the vorticity is discretized into vortex particles, lines, or sheets at which point advection becomes trivial, one merely needs to calculate the velocity field at each particle and step forward in time [38, 35, 40]. Several monographs exist that enumerate the details and advantages of Lagrangian vortex methods [1, 2, 3].

1.2 Lagrangian Methods

There are several variations of Lagrangian methods depending on how the vorticity is discretized; point vortex methods were the earliest with the work of Rosenhead [4] in 2D with singular point vortices, which was later built upon and improved [5, 6, 7, 8, 9].

Higher dimensional discretizations take the form of vortex line [10, 11, 12, 13], sheets [14, 15, 16], and volumes [17, 18, 19].

The velocity field due to the vorticity is usually calculated from solving the Poisson problem, or through inversion of the Poisson problem via evaluation of the Biot-Savart integral. There are several challenges with evaluation of the velocity: boundary conditions in the Poisson solution method, and singularities in the Biot-Savart volume integral method. Typically Lagrangian point vortex codes de-singularize the kernel [33, 34]. Direct summation for the velocity field has a $\mathcal{O}(N^2)$ complexity. Tree-codes [32] and Fast Multipole Methods (FMM) [31] are common ways of achieving a more efficient.

The convergence of vortex methods was proven early on in 2D [24] and was later extended to 3D [25].

1.3 Lagrangian Method Difficulties

Lagrangian point methods present several problems. Point disorganization can occur as the fluid evolves, this typically requires temporary meshing to recondition the discretization. This has been handled with various methods; recalculation of the quadrature weights at each time step [20, 21], regridding/rezoning [22], and remeshing [23] among them. Additionally, most Lagrangian approaches are limited to low order; careful point locations must be chosen and maintained through frequent remeshing etc. in order to achieve high-order convergence [31].

1.4 Existing Eulerian codes

In comparison, far less literature exists for Eulerian approaches. Chief among them is the work of Brown et al. [26] who adapted the velocity-vorticity approach to a low order Finite Volume (FV) solver. Later they were able to extend their method

to be accelerated via a FMM [27]. However, like most FV approaches, to extend to high order solution approximations requires extended stencils that ultimately limit the geometrical freedom of the mesh. Steinhoff et al. also used an Eulerian approach, but solved a modified form of the inviscid Euler equations with "vorticity confinement" rather than the velocity-vorticity equation [36].

1.5 Proposed Method

1.5.1 Alternate Advective Methods

The desire to resolve fine vortical structures motivates the need for a high order solver. Considering the challenges associated with achieving a high order Lagrangian method and the preliminary success of Brown et al.'s low order method, it is natural to attempt a high order Eulerian vorticity-velocity method, but what of the spatial discretization choice?

Finite difference methods suffer from similar problems as FV with extended stencils, as well as not being explicitly conservative. A finite element approach is ill-suited to the hyperbolic nature of vorticity advection. Spectral methods are promising, but for sparse vorticity domains are less-efficient due to the global support of the harmonic bases. However, discontinuous Galerkin (DG) methods are a natural choice; they are conservative, able to take advantage of vorticity sparseness, are well-suited to handle advection via intelligent choice of a numerical flux function, and have bases with local/compact support.

For domains free of impinging bodies a hexahedral mesh with a tensor product grid of interpolation points is convenient to implement. This permits the use of a line-DG [**?**] approach that considerably simplifies multi-dimensional cases by allowing reuse of 1D methods. A 2D domain will be considered to evaluate whether the method is practical, as well as to limit solution times to those that are reasonable on

a workstation. This has the advantage of removing the vortex stretching term and reducing the vorticity to a scalar. The resultant partial differential equation (PDE) takes on the familiar form of a scalar conservation law.

1.5.2 Velocity Field Fidelity

Unlike Lagrangian methods where advection is trivial, or Brown's method which is at low order, the necessary fidelity of the velocity field to maintain a particular global convergence rate that one would expect from the order of the vorticity field is not immediately obvious. The method by which one recovers the velocity field is decoupled from the advective solver, which has no a priori knowledge or assumptions about the quality of the velocity field approximation.

To maintain maximum flexibility for investigative purposes and to remove as much approximation error as possible the velocity field for validation of the underlying method is calculated via direct evaluation of the BS integral.

1.6 Thesis Structure

In Chapter 2, a brief overview of the underlying theory of the velocity-vorticity formulation, velocity evaluation, the Biot-Savart kernel and the discontinuous Galerkin method are covered. The extension of one-dimensional DG methods to 2D via a line-DG style method will also be reviewed.

In Chapter 3 the methodology taken to construct a high-order Eulerian velocity-vorticity is covered. Formation of the semi-discrete system, along with the development of some specialized tools is addressed. A high-order velocity evaluation method is laid out, as well as some computation-saving modifications. Explicit time-stepping via a Runge-Kutta method optimized for the spectra of the DG operator is briefly reviewed. Finally a test plan is laid out for validation and investigation of the overall solver and method.

In Chapter 4 results of the validation tests, convergence studies, and performance of the possible improvements and modifications is presented. Additionally runtime performance of the solver is examined, dependent on solver parameters, tolerances, order, etc.

In Chapter 5 the successful performance of the method and solver is presented and discussed. Additionally an un-implemented extension to the velocity evaluation via a modified Fast Multipole Method is presented. Chapter 7 points in the direction of possible expansion of the method to include additional physical behaviors and configurations, as well as several next steps that should be taken to improve the solver for practical solution of larger problems beyond what is currently possible.

Finally, Chapter 8 presents the specific implementation details necessary to implement the solver practically and efficiently. Several Matlab functions and routines are presented that either makeup the solver itself, or necessary auxiliary tools that are tangentially needed.

2 Theory

2.1 Navier-Stokes: Velocity-vorticity form

If the quantity *vorticity* is defined as

$$\omega = \nabla \times u \tag{2.1}$$

Then the traditional form of the Navier-Stokes equations can be recast, assuming an inviscid incompressible flow

$$\frac{\partial \omega}{\partial t} + u \cdot \nabla \omega - \omega \cdot \nabla u = S(x,t) \tag{2.2}$$

There are several benefits to the recast form; first, it explicitly conserves vorticity. The vorticity is recoverable from the primitive variable Navier-Stokes, but the diffusive nature of most upwinded advection means coherent vortical structures are quickly smeared. Also, frequently the distribution of the vorticity is quite sparse. The primitive variable form requires the velocity and pressure be evaluated over the whole domain.

In comparison, the velocity-vorticity form only requires velocity to be evaluated in areas of non-zero vorticity. In areas where there is no vorticity, nothing needs to be advected. The sparser and more concentrated the vorticity, the larger the gap in performance between the two forms; this is especially true in 3D domains where the scaling from surfaces to volumes gives even greater opportunity for empty local regions in the domain.

Additionally, the primitive-variable form requires the pressure be solved via a separate elliptic equation. The velocity-vorticity form is absent the pressure and

automatically ensures satisfaction of continuity.

Restricting to examining 2-D distributions of vorticity, several simplifications can be made. The originally vectorial vorticity becomes a scalar quantity, all vorticity is directed normal to the plane. As a result, the vortex stretching term in (2.2) becomes zero. The only non-zero component of ω is in the z-direction, however the gradient of the velocity field is zero in the z-direction, so the product is therefore zero. The result is

$$\frac{\partial \omega}{\partial t} + u \cdot \nabla \omega = S(x,t) \tag{2.3}$$

or if instead the second term is expressed in terms of the flux of the vorticity (where $f_i(\omega) = u_i\,\omega$):

$$\frac{\partial \omega}{\partial t} + \frac{\partial f}{\partial x_i} = S(x,t) \tag{2.4}$$

2.2 Velocity Field Evaluation: The Biot-Savart Integral

We shall decide that ω is the quantity of interest in the solution method. However, if this is the case then the question must be posed: how does one determine the velocity field? For an incompressible flow

$$\nabla^2 u = -\nabla \times \omega \tag{2.5}$$

If inverted, the Biot-Savart integral is obtained

$$u(x^*) = \int_\Omega K(x^*, x) \times \omega(x) dx \tag{2.6}$$

with the singular Biot-Savart kernel

$$K(x^*, x) = \frac{-1}{4\pi} \frac{x^* - x}{|x^* - x|^3} \tag{2.7}$$

There are several important points to note that are a consequence of this inversion. First, rather than solving the Poisson equation for the entirety of the domain we

choose to evaluate the velocity at some subset. For the purposes of advecting the velocity we will only need velocities near the vorticity itself. However, if the number of required velocity evaluation points is roughly proportional to the N DOFs of our vorticity approximation, then it might be expected for the velocity calculations to scale as $\mathcal{O}(N^2)$.

There are numerous methods to reduce the computational complexity of similar N-body type problems to $\mathcal{O}(NlogN)$ or $\mathcal{O}(N)$. Cyclic reduction [42], tree-codes [32, 43], and FMM [44] are all possibilities. However, because the velocity field calculation method is essentially decoupled from the discretization of the PDE, there is no a priori assurance that the velocity calculated is of sufficient fidelity to ensure convergence of the overall method (let alone convergence at the order one might expect based on solely the discretization). For maximum flexibility in investigating this dependency of overall convergence on the calculated velocity field we shall eschew more efficient techniques so that we can directly control the fidelity of the velocity field.

The second point to consider regarding the Biot-Savart integral is the singular nature of the exact kernel. Lagrangian point vortex methods have the benefit that the singularity and it's associated non-physical velocities occur in a relatively small region; they assume that the advective effect of the point vortex on itself is negligble. Generally, self-influence merely results in solid body rotation. Additionally they will frequently de-singularize the kernel by introducing a core function. The classical one used is the Rosenhead-Moore kernel [33, 34]:

$$K(x^*, x) = \frac{-1}{4\pi} \frac{x^* - x}{(|x^* - x|^2 + \sigma^2)^{3/2}} \tag{2.8}$$

The high-order Eulerian approach taken here however means that the Biot-Savart integral diverges everywhere within any of the extended vorticity patches thanks to self-influence. The approach taken by Brown [27] was to use the Rosenhead-Moore

kernel, choosing a core size such that the maximum velocity occurred on the face of the finite volume unit. This is constructible a priori because the vorticity is taken as constant across the volume, as is typical in a FV approach. If the vorticity is spatially varying however, the choice of core size is more troublesome. This will be examined in greater detail in the next chapter.

2.3 Discontinuous Galerkin

In order to solve (2.9) we adopt a method-of-lines approach. We will first spatially discretize the system to obtain the semi-discrete system, then we use an explicit time discretization method to march forward in time. Note that (2.9) has the form of a scalar conservation law, with ω being the conserved quantity. The velocity field that advects the conserved quantity has been calculated by evaluation of the Biot-Savart integral for the current timestep.

Our spatial discretization is at best an approximate solution to (2.9), that we shall denote as $\widetilde{\omega}(x,t)$. Some residual will remain for the approximate solution of the PDE at any time t

$$\frac{\partial \widetilde{\omega}}{\partial t} + \frac{\partial f}{\partial x_i} = R(x) \tag{2.9}$$

where we have shown a 1-D case and omitted vorticity sources for simplicity.

The discontinuous Galerkin (DG) approach was introduced by Reed and Hill to solve the steady-state neutron transport equation in 1973 [28] and later extended to general linear hyperbolic systems by Lesaint and Raviart [29]. It attempts to approximately satisfy the PDE in the following way: Seek the best approximation in a finite vector test space $\mathbb{W}_h$ in the L^2 norm sense. Minimize the L^2 norm by an orthogonal projection of the residual onto $\mathbb{W}_h$. Form a complete basis for $\mathbb{W}_h$ with a set of test functions $\phi_j \in \mathbb{W}_h$, such that the orthogonal projection satisfies:

$$\int_\Omega R(x)\phi_j \, dx = 0 \quad \text{for all } j \tag{2.10}$$

Substituting the residual for our conservation PDE yields:

$$\int_\Omega \frac{\partial \widetilde{\omega}}{\partial t}\,\phi_j\,dx + \int_\Omega \frac{\partial f(\widetilde{\omega})}{\partial x}\,\phi_j\,dx = 0 \quad \text{for all } j \tag{2.11}$$

Choose the space of piecewise polynomials for the finite dimensional approximation space, and decompose the global domain into K elements where the local approximation space $\overset{k}{\mathbb{V}}_h$ has basis functions $\overset{k}{\psi}_i(x)$ over the local domain $x \in [x_L, x_R]$ (henceforth we drop the element index k for brevity, except when describing two distinct elements that must be distinguished). Note that continuity of vorticity is not enforced across elements, and that the local approximation spaces for each element are independently defined. Finally, take the Bubnov-Galerkin choice of setting the test space to be the same as the approximation space so that $\phi_j = \psi_i \in \mathbb{V}_h$ if $i = j$. The local Mth order approximation to vorticity takes the form

$$\omega(x,t) \approx \widetilde{\omega}(x,t) = \sum_{i=0}^{M} a_i(t)\psi_i(x) \tag{2.12}$$

where a_i is some set of coefficients that are to be determined. The elemental approximating PDE is then

$$\sum_{i=0}^{M}\left[\frac{da_i(t)}{dt}\int_{x_L}^{x_R}\psi_i(x)\,\phi_j(x)\,dx\right] + \int_{x_L}^{x_R}\frac{\partial f(\widetilde{\omega})}{\partial x}\,\phi_j\,dx = 0 \tag{2.13}$$

To reduce the smoothness requirements on the flux, integrate the second term by parts to yield

$$\sum_{i=0}^{M}\left[\frac{da_i(t)}{dt}\int_{x_L}^{x_R}\psi_i(x)\,\phi_j(x)\,dx\right] + f\phi_j(x)\Big|_{x_L}^{x_R} - \int_{x_L}^{x_R} f(\widetilde{\omega})\,\frac{d\phi_j(x)}{dx}\,dx = 0 \tag{2.14}$$

The local solution offers no way of recovering the global solution, until one realizes that the flux at the element boundaries is multiply defined between elements. The global solution is allowed to be piecewise discontinuous across boundaries, so there is no guarantee that the flux between neighboring elements would agree. To resolve this (and at the same time recover a global solution) define a numerical flux analogous

to a Finite Volume Method that takes as input the vorticity at the adjacent element boundaries. We will use an upwind flux, where defining the average as $\{\{\omega^+\}\} = \frac{\omega^+ + \omega^-}{2}$ and the jump as $[[\omega]] = \omega^+ - \omega^-$, yields

$$\hat{f}_{upwind}(x^+, x^-) = u\{\{\widetilde{\omega}\}\} + \frac{|u|}{2}[[\widetilde{\omega}]] \tag{2.15}$$

We would also like to map any element to a computational element by means of a mapping $x = g(X)$ and it's inverse $X = G(x)$. If we set the domain of our computational element to be $X \in [-1, 1]$ and the element size to be $\Delta x = x_R - x_L$, then the mapping is

$$x = g(X) = \frac{X+1}{2}\Delta x + x_L \quad , \quad X = G(x) = \frac{2(x - x_L)}{\Delta x} + 1 \tag{2.16}$$

Applying a change of variables using the mapping to (2.17), accounting for the numerical flux, and being sure to include the determinant of the Jacobian matrix that results from the mapping ($J = g'(X) = \Delta x/2$) in the first integral:

$$\frac{\Delta x}{2}\sum_{i=0}^{M}\left[\frac{da_i}{dt}\int_{-1}^{1}\psi_i\,\phi_j\,dX\right] + \hat{f}\phi_j\Big|_{x_L}^{x_R} - \int_{-1}^{1} f(\widetilde{\omega})\,\frac{d\phi_j}{dX}\,dX = 0 \tag{2.17}$$

(Note: no determinant of the Jacobian matrix appears in the second integral because one of the basis functions has been differentiated. When the chain rule is applied during the change of variables an extra $1/g'(X)$ appears that cancels out with the $g'(X)$ that occurred from the change of variables on the integral).

We have purposely left the flux in the second integral unevaluated in terms of the basis functions, as well as the choice of basis functions left as undecided. We defer these choices to a more in depth investigation in Methodology. We also have developed the equations past the initial PDE in a 1-D form. We will shortly show that the basis coefficients are chosen to be nodal values. The 2-D solution will be recovered following the Line-DG methodology [30] where each element will be composed of a tensor product of two 1-D discretizations.

3 Methodology

3.1 Method Specific DG Choices

We now seek to take the general form of (2.17) and customize it to suit our needs. There are several choices to be made, the space of the basis, the physical meaning of the basis coefficients, and how it is formed on a 2D domain. We shall take the common choice of the space of polynomials for our basis. Additionally we shall choose our basis coefficients to be local nodal values that our basis interpolates.

One could be tempted to choose a modal basis, but for simplicity of extension a nodal basis allows one to readily expand 1D basis functions to 2D by means of a tensor product of two 1D bases. If the two 1D basis are linearly independent, then the tensor product spans $\mathbb{R}^2$. Furthermore, if the bases are orthogonal to each other than each can be treated separately from each other. To illustrate this we must choose our basis functions; a readily specified choice for interpolation is the Lagrange basis

$$\psi_i(x) = \ell_i(x) = \prod_{\substack{p=0 \\ p \neq i}}^{M} \frac{x - x_p}{x_i - x_p} \tag{3.1}$$

where M is the order of the basis and the set of points x_p are the $M+1$ interpolation points. The interpolatory property of the Lagrange basis means that $\ell_i(x_j) = \delta_{ij}$, so that the value of the function at the interpolation points forms the expansion coefficients

$$f(x) \approx \sum_{i=0}^{M} y_i \ell_i(x) \tag{3.2}$$

We can now use the developed 1D basis in a tensor product to form our 2D basis

$$f(x,y) \approx \left[\sum_{j=0}^{L} z_j \ell_j(y)\right] \times \left[\sum_{i=0}^{M} z_i \ell_i(x)\right] = \sum_{j=0}^{M}\sum_{i=0}^{M} z_{ij}\ell_j\ell_i = \sum_{j=0}^{M} z_{ij}\ell_j \sum_{i=0}^{M} \ell_i \qquad (3.3)$$

In general the tensor basis could have unequal order along each direction, but for simplicity we use an equal order.

An important consequence of the tensor basis means that the two basis directions are uncoupled except for at the interpolatory nodes at the intersection of each direction's basis. This is the central idea in a line-DG approach, we have turned what would be a 2D problem into two 1D problems of the form of (2.17). The 2D time evolution of the overall system is the linear combination of the 1D evolutions; notably the instantaneous rate of change of a particular node is the sum of the rates from each direction. Heuristically, a node can be thought of as a differential element with each direction's basis acting on the respective "face" of the node.

We can now substitute our basis into (2.17) to get a particular direction's PDE

$$\frac{\Delta x}{2}\sum_{i=0}^{M}\left[\frac{dy_i}{dt}\int_{-1}^{1} \ell_i\,\ell_j\,dX\right] + \hat{f}\ell_j\Big|_{x_L}^{x_R} - \int_{-1}^{1} f(\widetilde{\omega})\,\ell_j'\,dX = 0 \qquad (3.4)$$

There are two more points to be made regarding method specific choices, the interpolation nodes and the flux interpolation.

3.2 Interpolation Node Choice

It is well known that for even moderate orders equispaced interpolation nodes are a poor choice that will suffer from Runge's Phenomenon. The common solution is to use nodes that are the roots of an orthogonal polynomial family; Chebyshev, Legendre, and Legendre-Lobatto are all common choices. At first one may be tempted to choose the Lobatto points as they include nodes on the extremes of the domain, which are needed for the boundary flux evaluation. There is a minor problem with this however.

We would ideally like to collocate our quadrature nodes with our interpolation points, so that they can be reused for any numerical integration. However a Gauss-Lobatto quadrature rule is $2N-1$ order accurate. This means if we have the product of two Nth order polynomials that share the same nodes, a collocated Lobatto quadrature rule is not exact. While this is not outright impermissible, there is another point to consider.

If the interpolation points are the roots of a single orthogonal polynomial, then the associated Lagrange bases of differing order are orthogonal. Consider the inner product of two Lagrange polynomials that share the same interpolation nodes which are the roots of the Legendre polynomials, one may rearrange terms to obtain

$$\begin{aligned}
&\int \ell_i(x)\ell_j(x)dx = \int \frac{1}{c_1}\prod_{p\neq i}^{M}(x-x_p)\frac{1}{c_2}\prod_{q\neq j}^{M}(x-x_q)\ dx \\
&= \frac{1}{c_3}\int \Big[(x-x_0)...(x-x_{i-1})(x-x_{i+1})...(x-x_M)\Big]\prod_{p\neq j}(x-x_p)\ dx \\
&= \frac{1}{c_3}\int \Big[(x-x_0)...(x-x_{i-1})\boldsymbol{(x-x_i)}(x-x_{i+1})...(x-x_M)\Big]\frac{1}{\boldsymbol{(x-x_i)}}\prod_{p\neq j}(x-x_p)\ dx \\
&= \frac{1}{c_3}\int P_M(x)\pi_{M-2}(x)\ dx = 0
\end{aligned} \tag{3.5}$$

where P_M is the Mth order Legendre polynomial and $\pi(x)$ is a $M-2$th order arbitrary polynomial.

The Legendre polynomial is a result of the complete polynomial being reconstructed from it's roots one by one. Since the Legendre polynomials form a complete basis $\pi(x)$ can be expressed in terms of a sum of them, all lower order than P_M and all orthogonal to it as well, so the integral is zero. For the inner product of the same Lagrange basis for a point i, one simply recovers the associated quadrature weight w_i

so that in general

$$\int \ell_i(x)\ell_j(x)dx = \delta_{ij}w_j \tag{3.6}$$

3.3 Flux Interpolation

It may seem somewhat pedantic to worry about the nature of the flux interpolant if one thinks only in the case of collocated interpolation points for the vorticity and velocity. After all, although one may think of the interpolant as either the interpolation of the product of the velocity and vorticity, or the product of the interpolations of velocity and vorticity; the resultant interpolant is identical at the nodes

$$I_N f(x_i) = I_N(u(x_i)\omega(x_i)) = I_N(u(x_i))I_N(\omega(x_i)) \tag{3.7}$$

Furthermore, if we perform a Gauss-Legendre quadrature with collocated nodes, the quadrature is exact and the integral of either choice is equal

$$\int f\ dx \approx \sum_i^M u_i\omega_i w_i = \sum_i^M u_i \sum_j^M \omega_j \int \ell_i\ell_j \tag{3.8}$$

The interpolation of the product may be order M, while the product of the interpolations is order $2M$, but the integral of either is equal; deciding between the two options would seem to be moot. Consider however the case where we wish to integrate the product of the interpolated flux and the derivative of the weighting function. We are fortunate that the interpolation of the product is order M and the derivative of the weighting basis M-1, the collocated Gauss-Legendre quadrature is exact

$$\int I_N(f)\,\ell_j'(x)\ dx = \sum_i^M u_i\omega_i\ell_j'(x_i) \tag{3.9}$$

There is an important decision we have implicitly made here however! By choosing the quadrature of this form we have implicitly chosen to use the interpolant of the product. The two different flux interpolation choices no longer result in the same

integral value. Importantly this means if we expect to need the full interpolatory order N of both the velocity and the vorticity in the stiffness integral, we will not be able to recover them. Instead we will recover the Nth order interpolant of them.

To retain the full order of each quantity, as well as suffer no quadrature error, we would have to instead evaluate the integral of the form

$$\begin{aligned}\int u(x)\omega(x)\ell'_j(x)\ dx &\approx \int \sum_i^M u_i\ell_i(x) \sum_k^M \omega_k\ell_k(x)\ \ell'_j(x)\ dx \\ &= \sum_i^M u_i \sum_k^M \omega_k \int \ell_i(x)\ell_k(x)\ell'_j(x)\ dx\end{aligned} \tag{3.10}$$

We will do just this in the next section.

3.4 Modified Quadrature

To retain full interpolatory order of both the velocity and vorticity in the stiffness integral we seek a set of quadrature weights derived from

$$\sum_i^M u_i \sum_k^M \omega_k\ w_{unknown} = \sum_i^M u_i \sum_k^M \omega_k \int \ell_i(x)\ell_k(x)\ell'_j(x)\ dx \tag{3.11}$$

note that we might alternatively wish to find something that looks more like the Gauss-Legendre quadrature rule

$$\sum_i^M u_i \sum_k^M \omega_k \sum_q^M \ell'_j(x_q)\ w_{unknown2} \tag{3.12}$$

However we can actually go about it better keeping the derivative weighting basis inside the integral! The derivative of the weighting basis can be derived analytically by use of logarithmic differentiation

$$f' = f\,[ln(f)]' \rightarrow \ell'_j(x) = \ell_j(x) \sum_{\substack{p=0 \\ p\neq j}}^M \frac{1}{(x - x_p)} \tag{3.13}$$

the resultant form can now be included in the integral.

The integral of the Lagrange basis functions associated with each interpolation times the derivative of the weighting basis must equal the quadrature weight associated that particular integrand, that is rather than M+1 quadrature weights associated with the M+1 nodes of the interpolation of the product, we have the associated $N(M+1)^2$ weights (where N is the number of possible weighting bases, for Galerkin methods N=M of course)

$$\int \ell_i(x)\ell_k(x)\ell'_j(x)\,dx = \mathbf{W}_{ikj} \tag{3.14}$$

The integral can be done numerically by means of a normal Gauss quadrature of order at least $3M-1$ to be exact. These weights are pre-calculated at the start of the solver and saved for re-use.

The resultant modified quadrature now takes the form

$$\int u(x)\omega(x)\ell'_j(x)\,dx \approx \mathbf{u}_i^T\mathbf{W}_{ikj}\,\boldsymbol{\omega}_k \tag{3.15}$$

for each jth weighting basis function.

This may seem to incur a high cost, until one considers several facts. First, normal Gauss-Legendre quadrature requires one to perform the stiffness integral for each weighting basis already, so the j dimension of the quadrature matrix is no worse than before. Secondly, one is able to extract *more* information out of the *same* number of interpolation points and achieve superior convergence of the stiffness integral than the standard Gauss-Legendre rule. Third, by including the derivative of the weighting basis in the integral for the quadrature weight matrix, we integrate it exactly a priori. It never appears in the later quadrature of the stiffness integral except implicitly in the quadrature weight matrix itself.

Finally, and most interestingly the order and definition of the Lagrange basis functions for the vorticity and velocity interpolations are *decoupled*. It is not necessary for the order of the velocity interpolation to be equal to the vorticity. Indeed, the

set of interpolation nodes don't even have to be of the same *kind*. It is completely permissible for example to have the vorticity nodes be the Gauss-Legendre nodes, and the velocity nodes to be Lobatto nodes.

This has serious advantages for the overall method of the thesis, both in terms of minimizing interpolation and quadrature error, as well as giving excellent flexibility in the solver. Normally to vary the order of the velocity interpolation one would need to re-interpolate at the vorticity nodes to recover the flux interpolant. This adds additional interpolation error, as well as potential aliasing problems. This additional interpolation step would also end up requiring about the same number of operations as using the larger quadrature weight matrix, making the two about equal in terms of computational cost. Instead one is free to choose the velocity order freely and the solver adapts seamlessly.

There is an additional computational savings as well. Although we would prefer to use the Legendre points for the vorticity (to preserve the diagonal mass matrix structure that we shall discuss in a coming section), we would like to use Lobatto points for the velocity. Ordinarily we would have to evaluate the boundary velocities and then leave them unused in the stiffness integral with normal Gauss-Legendre quadrature; both the vorticity and velocity would have to share the same nodes or face re-interpolation costs. This adds wasted velocity evaluations which are the most expensive operation. One might be tempted to only evaluate the velocity at the Legendre points and extrapolate the velocity at the boundaries, but there is no guarantee that the two interpolated velocities from adjacent elements would agree; this would present obvious challenges in the boundary flux evaluation. However by using the modified quadrature matrix one can freely mix node sets, meaning the boundary velocity values are reused in the stiffness integral.

One other point remains worth mentioning regarding the modified quadrature matrix, the maximum order polynomial that can be integrated exactly. Similiar to

some composite quadrature rules the order depends on whether the number of nodes is odd or even. An even node basis for one of the bases results in a $2M + 1$ order accurate method, odd node number results in a $2M$ order accurate method (still sufficient to exactly integrate the product of two Mth order interpolants). A script in the thesis code repository is available that validates the convergence claims and order of accuracy of the modified quadrature method.

3.5 Overall Solver Structure

It is worth briefly mentioning the overall structure of the solver, as it motivates some of the machinery developed and provides a useful overview of the detailed portions of the following sections. The preamble contains various solver configuration parameters and initializes most of the persistent data objects. It also initializes quadrature information which in turn is used to initialize the quadrature weight matrices needed for the order of the vorticity and velocity. The quadrature matrix is modified in place to incorporate the Jacobian as well as the inverse Mass matrix entry so that these do not needed to be multiplied by after the fact.

Most data is stored in one of two different formats, either a column-major form that matches Matlab's memory layout to optimize indexing performance, or a "stream-wise" format. The stream-wise terminology shouldn't be confused with any physical phenomenon, rather it relates to the tensor composition of the code. As the x and y direction problems are largely decoupled, each is treated independently, with a "stream" being a particular 1D basis along the coordinate direction. Adjacent streams are parallel 1D basis functions that share a boundary node. For an example layout of x and y streams see Figure 3.1. The next section of the solver creates element-stream-boundary-node associativity. These are used to freely select the appropriate sections of data associated with a particular hierarchy. Most of these are simply a column-wise index list, some are specifically shaped so that the returned data from

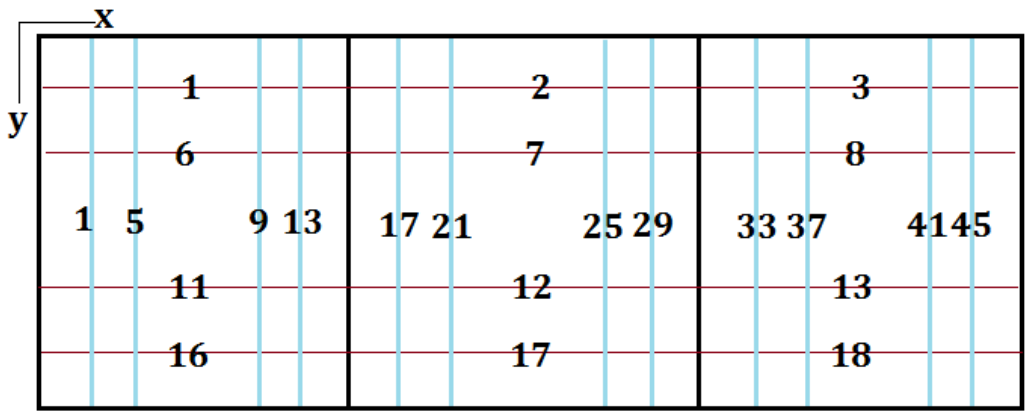

Figure 3.1: "Stream-wise" numbering for elements 1, 5, and 9 in a 4x5 element global domain.

the index selection is the correct shape for reuse (ND matrix etc). This is also where the coordinates of the vorticity nodes, velocity boundary nodes, and velocity internal element nodes are specified.

The next section consists of a cell list of functions that specify the initial conditions of the system. The resultant initial condition is the linear superposition of all the selected generating functions. The interpolation vectors for the surface flux are constructed, again incorporating the Jacobian and inverse matrix entry like in the quadrature weight matrix to avoid element-wise multiplications later. Finally the "global" kernel evaluation matrices are constructed for later re-use (these will be described in greater detail in a following section).

At this point the remainder of the code occurs inside the main time marching loop. An explicit Runge-Kutta routine is used with an inner loop used for stage-stepping. Depending on the solver setup, the velocity evaluation code is either executed every time step, or every stage. The sum of the vorticity in each element is computed and those above a threshold are added to element-wise and stream-wise masks. Each element in the mask is treated as a source of vorticity and the velocity from that element is added to the current running total for the global velocity. Total element and element boundary velocities are formed for later use in the semi-discrete system.

The DG advection code always occurs within the RK stage loop. Vorticity is extrapolated at the element boundary. Boundary conditions are applied and the element boundary fluxes are evaluated. The stiffness term is evaluated for the current stage's vorticity and velocity configuration. At this point the semi-discrete system is complete and the stage's rate of change for all nodes is computed for each direction and the two are added together. The new vorticity is computed for the stage and both the time-stepping and stage-stepping loops wrap.

There is also some additional post-processing type routines that run that save simulation data for later use, and can also update in realtime displaying current simulation health statistics (L^2 norm etc.).

3.6 Mass Matrix

As discussed in the Interpolation Node Choice section, the set of Legendre roots for the vorticity interpolation points results in

$$\int \ell_i(x)\ell_j(x)dx = \delta_{ij}w_j \tag{3.16}$$

This proves particular useful for the mass term, meaning the choice of Legendre nodes for interpolation results in a diagonal mass matrix that is trivially invertible. (3.4) is then simplified to

$$\frac{d\widetilde{\omega}_j}{dt}\frac{w_j\Delta x}{2} + \hat{f}\ell_j\Big|_{x_L}^{x_R} - \int_{-1}^{1} f(\widetilde{\omega})\,\ell_j'\,dX = 0 \tag{3.17}$$

One could instead choose the set of Lobatto points to avoid extrapolating the vorticity at the element boundaries, but this would result in a full mass matrix that when inverted necessitates a matrix-vector product which is about as computationally expensive as extrapolating the boundary vorticity.

3.7 Use and Abuse of Structured Tensor Mesh

The cases considered all are free of impinging geometry and have no bias known a priori, as such a constant-size structured mesh of squares is used. It is worth noting that this is by no means required, an un-structured mesh is easily treatable via a mapping to a canonical computational element [30]. The structured mesh merely affords ease of implementation, as well as simplification of element-stream-boundary-node indexing.

Likewise, the tensor product basis used for the vorticity can be exploited for computational and ease-of-implementation gains. The tensor product means that the 2D integral of the vorticity times the Biot-Savart kernel can be reduced by a matrix-vector and vector-vector product by pre-multiplying the vorticity of a source by the outer product of the Gauss-Legendre weights component-wise, which is then reused for each kernel-vorticity integration.

$$\int \omega \times K \, dx \approx \boldsymbol{w}^T(\boldsymbol{\omega}_E . * \boldsymbol{K})\boldsymbol{w} \rightarrow (\boldsymbol{w}\boldsymbol{w}^T . * \boldsymbol{\omega})\boldsymbol{K} \tag{3.18}$$

where ".*" indicates element-wise multiplication per Matlab convention. It's worth noting that the pre-multiplied product and the kernel are reshaped to be row and column vectors respectively. If there are many kernels to be integrated for a particular vorticity source the computational savings are appreciable.

3.8 Stiffness Matrix

We can apply the results for the modified quadrature rule from (3.15) to the purpose of evaluating the stiffness integral we have left unevaluated thus far. Recall that for maximum flexibility the velocity interpolation order is independent of the vorticity order. In addition the Lobatto points are chosen for the velocity to reuse the boundary velocities. At first glance it might seem actually counter productive to choose a

different order or set of nodes for the velocity. To avoid needing "off-stream" interpolation the velocity nodes should be co-linear with the vorticity streams. The result for a 5th order velocity and vorticity field is shown in Figure 3.2.

It would appear we actually need double the velocity evaluation points! How could this be a savings over collocated velocity points? The non-square tensor product of the Lobatto nodes with the Legendre nodes means that the x-stream velocity nodes will never coincide with the y-stream velocity nodes.

The answer is that the velocity is evaluated component-wise. Along the x-stream we only require u_x and along the y-stream we only require u_y. We do not need need the other velocity component at any particular velocity node, so though they may not coincide we don not miss out on any opportunities. The trade-off is that the velocity field is perhaps a bit harder to post-process, but if the result cuts in half the expected runtime of the most expensive portion of the method it seems a fair trade-off.

We can now happily apply the modified quadrature matrix to our elemental PDE

$$\frac{d\widetilde{\omega}_j}{dt}\frac{w_j \Delta x}{2} + \hat{f}\ell_j\Big|_{x_L}^{x_R} - \mathbf{u}^T \mathbf{W}_{--j}\,\boldsymbol{\omega} = 0 \tag{3.19}$$

rearranging to obtain the rate of change of the nodal vorticity

$$\frac{d\widetilde{\omega}_j}{dt} = \frac{2}{w_j \Delta x}\left[\mathbf{u}^T \mathbf{W}_{--j}\,\boldsymbol{\omega} - \hat{f}\ell_j\Big|_{x_L}^{x_R}\right] \tag{3.20}$$

This nodal equation now describes the stream's contribution to the time evolution of a particular node. This calculation is performed for all vorticity nodes on all x-streams and added to the rate of change of the matching vorticity nodes in the y-streams

3.9 Vortex Sparseness

One of the advantages of DG as an advection scheme is the compact local support of the vorticity. This means that for a large domain with only sparse vorticity one should be able to only consider those elements that contain vorticity. The sparseness

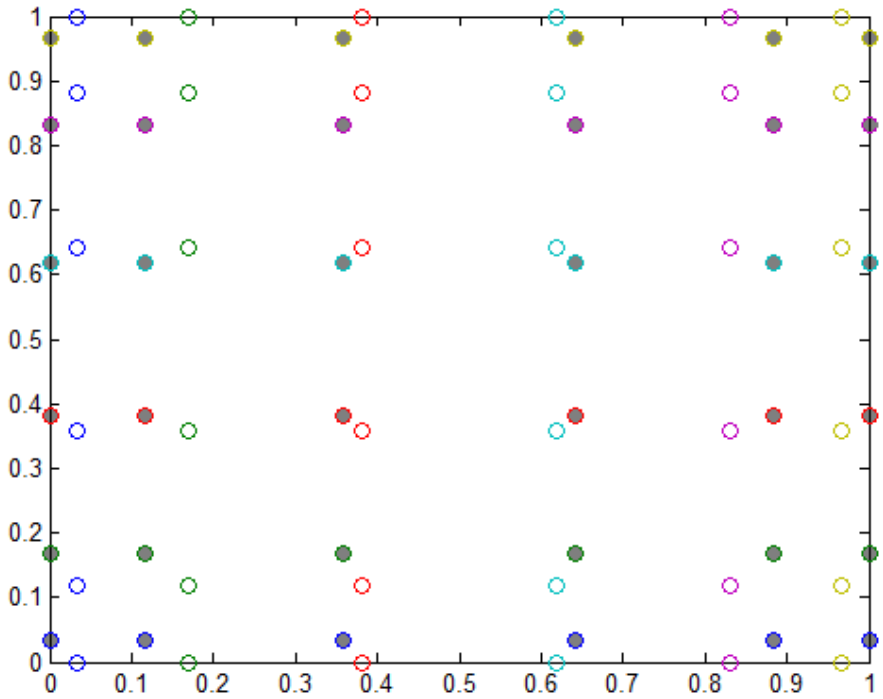

Figure 3.2: 5^{th} order Lobatto velocity nodes along x-streams (dark circles) and y-streams (empty circles).

can be applied in several ways. From the DG perspective, if an element is empty than there is nothing to advect and the element can be left out of evaluation entirely. The reality is that even small numerical errors in the boundary flux may lead to non-physical perturbations in the vorticity field in what should be otherwise empty elements. These elements can now now not be ignored, lest some instability result from not properly handling non-zero elements.

The reality is of course from a vorticity-velocity perspective these elements are unimportant, even if they are important from a consistency standpoint for the DG part of the code. Therefore a threshold value can be specified, elements with total vorticity less that the threshold value are ignored, and only those elements above the threshold are put in a mask. The masked elements are the ones considered as sources for the velocity evaluation. Additionally, only the boundary velocities for these unimportant elements are calculated and a reduced velocity order stiffness term is calculated for them.

The current state of the solver is bottlenecked by the velocity evaluation, so although some savings would be realized by maintaining true vorticity domain sparseness, the result is that it has little impact on the overall speed of the solver. The unmasked elements have only a marginal impact on the cost of the velocity evaluations. However empty elements do have one major effect on solver performance, this will be discussed in the next section.

3.10 Velocity Evaluation

For ease of implementation and to eliminate the effects of as most approximation error as possible on validation, the velocity is evaluated directly at each Lobatto velocity (only the boundary nodes for the unmasked elements) by means of the Biot-Savart integral. It is important to note that has $\mathcal{O}(N^2)$ complexity, and so the runtime rapidly reaches a limit of impracticality above a limit.

The velocity at a particular node is calculated by summing the net effect of each masked source element. A loop steps through each source element and adds it's contribution to all velocity nodes. The actual source-node pair evaluation is done via a regular Gauss-Legendre quadrature of order equal to the vorticity field. For the far field this performs as expected, with excellent convergence. However for elements directly neighboring the source element, as well as the source element itself, the nearly singular nature of the integral poses convergence problems for the quadrature.

This is unsurprising; the weighting function for the Legendre polynomials is simply $w_{Leg}(x) = 1$. As such, the product of the kernel and the vorticity near the source element is poorly interpolated by a polynomial. Ideally one would use a weighting function of a form similar to the singular portion of the kernel to form a new orthogonal basis, but here one is limited by the tensor basis. More freedom is required in the interpolation points to provide adequate leeway to generate desirable quadrature rules. The generation of general multivariable interpolation and quadrature bases is an area of active and relatively recent effort [45, 46, 47].

At first one might think that the predominant bottleneck in the scaling of the velocity evaluation is simply the evaluation of the Biot-Savart integral. While this may be the case in other codes, or other workstation setups, this actually only accounts for typically less than half the cost. The majority of the cost actually results from the assembly of the kernel values. One is presented with three options: calculate a new set of kernel values for each source element, form a "global" matrix of kernel values that one maps to the particular source element under consideration, or directly calculating and storing kernel values for every source-node pair.

The first choice is the most direct, but also the slowest. It is bottlenecked by computational cost; the number of operations to form the kernel actually exceeds those needed to evaluate the integral. The third method is the fastest, but bound by memory constraints which scale as Np^4K^4. Even a modest 5th order velocity method

with 20x20 elements (15k DOF) would require 1.8Gb of memory to store.

The second method is the one currently used in the code. The global domain is reflected about both axes for a central element and the kernel values are calculated for all elements in relation to the reference source. Depending on where the actual source element is in the true domain determines what portion of the "global" domain is selected with respect to the reference element, see Figure 3.3. This realizes significant computational savings from not having to repeatedly recalculate the kernel values, however it is memory access bottlenecked. Repeatedly selecting and manipulating non-contiguous blocks of memory can be slow. Even being careful to arrange as much data column-major as possible only delays the problem.

This also means otherwise "empty" elements affect performance by further fragmenting the global kernel matrix. At some point the sparseness of the domain is sufficient that the recalculation of the kernel for each source outperforms an overly fragmented global matrix. A hybrid method that uses as much memory as reasonable to store kernel values for the elements with the greatest vorticity, and calculates all others on-the-fly would be the most efficient use of resources and was considered but development time did not permit it's construction. Note that the memory bottleneck limitation is not an intrinsic property of the underlying method itself, but rather a consequence of the solver implementation and Matlab's underlying memory model.

3.11 Reduced Order Velocity (Global/Far-field)

The flexibility permitted by the modified quadrature matrix allows a choice of velocity order independent of the vorticity order. A reduced velocity order reduces the runtime of the solver significantly, but this comes with some potentially severe drawbacks as will be shown. The reduced order velocity field is most easily applied to unmasked elements, but can also be extended to far-field elements. The stiffness integral can be broken up into low and high order pieces with self, neighboring, and nearby elements

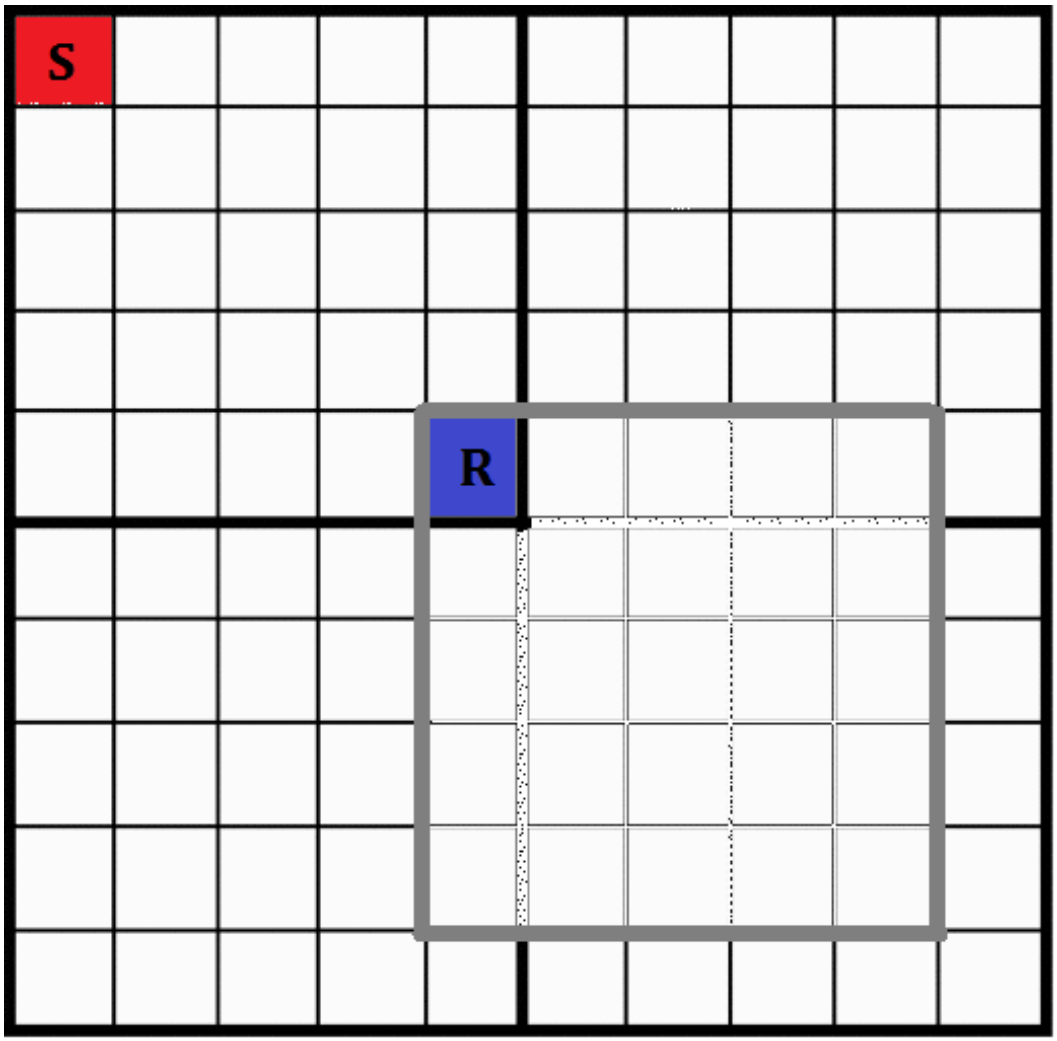

Figure 3.3: The "global" kernel matrix showing (R)eference source and actual (S)ource, the upper left quadrant is the actual full domain, the selected region with respect to (R) selects the correct values of the kernel for (S).

evaluated at high order, and slowly changing far-field elements at reduced velocity order.

In some senses a reduced order far-field component is an analogue to the cluster-cluster interactions of a fast multipole method. The implementation of the reduced order far-field evaluations is deferred to a later extension of the code that will fully implement a FMM-like evaluation.

3.12 Explicit Time-Stepping

With the semi-discrete system formed, one must now choose how to discretize in time. For simplicity an explicit time-stepping method will be used. The two chief categories are multi-step and multi-stage methods, typically either Adams-Bashforth or Runge-Kutta respectively. It is known that Forward Euler and AB2 are unstable for DG with an upwind flux due to it's dissipative nature. The choices permitted then are Runge-Kutta or a higher-order Adams-Bashforth.

The recent thesis work of Atcheson [48] presents an excellent overview of the optimal choice of time-stepping algorithm for a DG operator. In particular it was shown that Runge-Kutta methods outperformed AB3 and that all higher-order Adams-Bashforth schemes required far too restrictive of a time-step to satisfy the CFL condition to be efficient. Out of all the Runge-Kutta schemes considered [49] one of the competitive options was the "NRK14C" scheme developed by Niegemann et al. [50]. This is an optimized 4th order, 14 stage low storage scheme of similar structure to other 2N style schemes [51].

Notably, NRK14C permits a time-step 1.86 times as long per stage than the canonical RK4 assuming the limitation is the negative real component in the stability region of the method. This equates to a proportional runtime decrease for almost negligible implementation effort.

There is one further choice to be made regarding time-stepping: where to include

the velocity evaluation. It is reasonable to assume that in practice the numerical stiffness of the problem is due to the DG operator, not the velocity evaluation. This means one is free to either evaluate the velocity at each stage, or instead to evaluate at the start of each step.

Of course the time discretization error of the velocity field will be increased, but the runtime of the resultant method will be an order of magnitude faster. If the CFL condition is stringent enough to necessitate a small time-step compared to the evolution rate of the velocity, the velocity evaluation is moved outside of the stage loop. The effect on the overall convergence of the method is important to consider when making this choice.

3.13 Validation and Convergence of Proposed Methods

Of primary interest in the development of a new method is validation of it with common test cases, as well as an evaluation of convergence. Validation was performed in several ways; the 7th order polynomial test case by Perlmann [37] for a stationary vortex permits an analytical solution to compare against. Additionally, the test cases of the evolution of an elliptical vortex patch [39] as well as a system of patches [38] are considered and compared against. In each of these cases the conservation vorticity was also monitored.

Actual convergence rate of the method at various order vorticity fields, velocity fields, velocity evaluation step/stage-wise, effective core size, and long time integration also take advantage of the analytical solution available from the Perlmann test case. Qualitative assessment of the convergence rate dependence on the test parameters was also performed for the elliptical patch and system of patches.

Finally several interesting test cases are also presented to attempt to tax the method. First considered is the collision of two vortex pairs. The interaction and merging of collisions of this nature are known to generate very complex secondary

structures, and proved to be a useful discriminatory test for resolution of fine scale features. Finally a pair of "leap-frogging" vortex pairs are used to examine the long term stability of the method.

4 Results

4.1 Analytical Test Cases

.

.

.

4.2 Elliptical Blob

.

4.3 Arbitrary Patch

.

.

4.4 Vortex Pair

.

.

4.5 Vortical System

.

.

4.6 Reduced Order Velocity

.

.

4.7 Runtime Speed and Memory Footprint of Solver

5 Discussion

5.1 Analytical Validation of Model

Comparison of existing analytical solutions vs model (Euler vortex, 5th order poly, other Saffmann test cases)

.

5.2 Empirical Convergence of Method

.

5.3 Computational Efficiency

5.4 Extended Body Fast Multipole Formulation

6 Conclusion

7 Recommendations

7.1 Extension to 3D

7.2 Flux Limiters

7.3 Impinging Geometry and Boundary Conditions

7.4 Viscosity

7.5 Implicit and Local Time Stepping

7.6 Parallelizability

7.7 FMM Improvements

7.8 2D Non-classical Quadrature for Nearly Singular Integrands

8 Implementation and Algorithms

8.1 Modified Two-part quadrature

Vectorized Lagrange evaluation, Lagrange derivatives, etc

8.2 Binary Singleton Expansion Matrix-Vector Products

8.3 Lagrange Bases and Constructed Quadrature

8.4 Velocity Evaluation

8.5 Semi-discrete System Construction

8.6 Low-storage Stability Optimized Runge-Kutta

9 Literature Cited

Bibliography

[1] H. J. Lugt, Vortex Flow in Nature and Technology, Krieger Publishing Company, Malabar, FL, USA, 1983.

[2] P. G. Saffman, Vortex Dynamics, Cambridge Univ. Press, Cambridge, UK, 1992.

[3] C. G. Speziale, On the advantages of the vorticity-velocity formulation of the equations of fluid dynamics, J. Comput. Phys. 73 (1987) 476-480.

[4] L. Rosenhead, The formation of vorticies from a surface of discontinuity, Proc. Roy. Soc. London Ser. A 134 (1931) 170-192.

[5] G. Birkhoff, J. Fisher, Do vortex sheets roll up?, Rend. Circ. Math. Palermo, Ser. 2 8 (1959) 77-90.

[6] F. H. Abernathy, R. E. Kronauer, The formation of vortex streets, J.Fluid Mech. 13 (1962) 1-20.

[7] A. J. Chorin, P. S. Bernard, Discretization of a vortex sheet, with an example of roll-up, J. Comput. Phys. 13 (3) (1973) 423-429.

[8] K. Kuwahara, H. Takami, Numerical studies of two-dimensional vortex motion by a system of point vortices, J. Physical Society of Japan 34 (1) (1973) 247-253.

[9] R. G. Zalosh, Discretized simulation of vortex sheet evolution with buoyancy and surface tension effects, AIAA Journal 14 (11) (1976) 1517-1523.

[10] G. K. Batchelor, An Introduction to Fluid Dynamics, Cambridge University Press, 1967, 1973.

[11] W. T. Ashurst, E. Meiburg, Three-dimensional shear layers via vortex dynamics, J. Fluid Mech. 189 (1988) 87-116.

[12] J. E. Martin, E. Meiburg, Numerical investigation of three-dimensional evolving jets subject to axisymmetric and azimuthal perturbation, J. Fluid Mech. 230 (1991) 271-318.

[13] A. Leonard, Numerical simulation of interacting, three-dimensional vortex filaments, in: Proceedings of the IV Intl. Conference on Numerical Methods of Fluid Dynamics, no. 35 in Lecture Notes in Physics, Springer-Verlag, 1975, pp. 245-250.

[14] M. E. Agishtein, A. A. Migdal, Dynamics of vortex surfaces in three dimensions: Theory and simulations, Physica D 40 (1989) 91-118.

[15] O. M. Knio, A. F. Ghoniem, Three-dimensional vortex simulation of rollup and entrainment in a shear layer, J. Comput. Phys. 97 (1991) 172-223.

[16] O. M. Knio, A. F. Ghoniem, Vortex simulation of a three-dimensional reacting shear layer with in finite-rate kinetics, AIAA J. 30 (1) (1992) 105-116.

[17] G. Russo, J. A. Strain, Fast triangulated vortex methods for the 2D Euler equations, J. Comput. Phys. 111 (1994) 291-323.

[18] M. Carley, A triangulated vortex method for the axisymmetric Euler equations, J. Comput. Phys. 180 (2002) 616-641.

[19] S. A. Huyer, J. R. Grant, Solution of two-dimensional vorticity equation on a Lagrangian mesh, AIAA Journal 38 (5) (2000) 774-783.

[20] J. T. Beale, On the accuracy of vortex methods at large times, in: IMA Workshop on Computational Fluid Dynamics and Reacting Gas Flows, Springer-Verlag, 1988, p. 19.

[21] J. S. Marshall, J. R. Grant, Penetration of a blade into a vortex core: vorticity response and unsteady blade forces, J. Fluid Mech. 306 (1996) 83-109.

[22] H. O. Nordmark, Rezoning for higher order vortex methods, J. Comput. Phys. 97 (1991) 366-397.

[23] H. N. Najm, R. B. Milne, K. D. Devine, S. N. Kempa, A coupled Lagrangian-Eulerian scheme for reacting flow modeling, ESAIM Proc. 7 (1999) 304-313.

[24] O. H. Hald, V. D. Prete, Convergence of vortex methods for Euler's equations, Math. Comput. 32 (1978) 791-809.

[25] J. T. Beale, A convergent 3-D vortex method with grid-free stretching, Math. Comput. 46 (174) (1986) 401-424.

[26] R.E. Brown. Rotor Wake Modeling for Flight Dynamic Simulation of Helicopters. AIAA Journal, 2000. Vol. 38(No. 1): p. 57-63.

[27] A.J. Line, R.E. Brown. Efficient High-Resolution Wake Modelling using the Vorticity Transport Equation. in 60th Annual Forum of the American Helicopter Society. 2004. Baltimore, MD.

[28] W.H Reed and T.R. Hill. Triangular mesh methods for the neutron transport equation. 1973.

[29] Lesaint, P., and Pierre-Arnaud Raviart. On a finite element method for solving the neutron transport equation. Mathematical aspects of finite elements in partial differential equations 33 (1974): 89-123.

[30] P.O. Persson. A Sparse and High-Order Accurate Line-Based Discontinuous Galerkin Method for Unstructured Meshes. J. Comp. Phys., Vol. 233, pp. 414-429, Jan 2013.

[31] J. Strain. Fast adaptive 2D vortex methods. Journal of computational physics 132.1 (1997): 108-122.

[32] K. Lindsay, and R. Krasny. A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow. Journal of Computational Physics 172.2 (2001): 879-907.

[33] L. Rosenhead. The spread of vorticity in the wake behind a cylinder, Proc. Roy. Soc. London Ser. A 127, 590 (1930).

[34] D. W. Moore. Finite amplitude waves on aircraft trailing vortices, Aero. Quart. 23, 307 (1972).

[35] Moussa, C., Carley, M. J. (2008). A Lagrangian vortex method for unbounded flows. International journal for numerical methods in fluids, 58(2), 161-181.

[36] J. Steinhoff, D. Underhill, Modification of the Euler equations for "vorticity confinement": Application to the computation of interacting vortex rings, Phys. Fluids 6 (8) (1994) 2738-2743.

[37] M. Perlman. On the accuracy of vortex methods, J. Comput. Phys. 59 (1985) 200–223.

[38] J. Strain. 2D vortex methods and singular quadrature rules. Journal of Computational Physics 124.1 (1996): 131-145.

[39] P. Koumoutsakos. Inviscid axisymmetrization of an elliptical vortex, J. Comput. Phys. 138 (1997) 821–857.

[40] Koumoutsakos, P., Leonard, A. (1995). High-resolution simulations of the flow around an impulsively started cylinder using vortex methods. Journal of Fluid Mechanics, 296, 1-38.

[41] K. B. Southerland, J. R. P. III, W. J. A. Dahm, K. A. Buch. An experimental study of the molecular mixing process in an axisymmetric laminar vortex ring, Phys. Fluids A 3 (5) (1991) 1385–1392.

[42] Schumann, U., Sweet, R. A. (1976). A direct method for the solution of Poisson's equation with Neumann boundary conditions on a staggered grid of arbitrary size. Journal of Computational Physics, 20(2), 171-182.

[43] Barnes, J., Hut, P. (1986). A hierarchical O (N log N) force-calculation algorithm.

[44] Greengard, L., Rokhlin, V. (1987). A fast algorithm for particle simulations. Journal of computational physics, 73(2), 325-348.

[45] Gasca, Mariano, Thomas Sauer. Polynomial interpolation in several variables. Advances in Computational Mathematics 12.4 (2000): 377-410.

[46] Xu, Yuan. Minimal Cubature rules and polynomial interpolation in two variables. Journal of Approximation Theory 164.1 (2012): 6-30.

[47] Gautschi, Walter. Neutralizing nearby singularities in numerical quadrature. Numerical Algorithms 64.3 (2013): 417-425.

[48] Atcheson, Thomas. Explicit Discontinuous Galerkin Methods for Linear Hyperbolic Problems. Diss. Masters Thesis, Rice University. http://hdl. handle. net/1911/75120, 2013.

[49] Toulorge, Thomas, and Wim Desmet. Optimal Runge–Kutta schemes for discontinuous Galerkin space discretizations applied to wave propagation problems. Journal of Computational Physics 231.4 (2012): 2067-2091.

[50] Niegemann, Jens, Richard Diehl, and Kurt Busch. Efficient low-storage Runge–Kutta schemes with optimized stability regions. Journal of Computational Physics 231.2 (2012): 364-372.

[51] Carpenter, Mark H., and Christopher A. Kennedy. Fourth-order 2N-storage Runge-Kutta schemes. (1994).