VORTEX DOMINATED FLOWS: A HIGH-ORDER, CONSERVATIVE EULERIAN SIMULATION METHOD

BY

JOSH BEVAN

ABSTRACT OF A THESIS SUBMITTED TO THE FACULTY OF THE
DEPARTMENT OF MECHANICAL ENGINEERING
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE
UNIVERSITY OF MASSACHUSETTS LOWELL
2015

Thesis Supervisor: David J. Willis

Assistant Professor, Department of Mechanical Engineering

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 **viii**
List of Tables . . . viii

List of Figures **ix**
List of Figures . . . ix

1 Introduction **1**
1.1 Velocity-Vorticity equation . . . 1
1.2 Lagrangian Methods . . . 1
1.3 Lagrangian Method Difficulties . . . 2
1.4 Existing Eulerian codes . . . 2
1.5 Proposed Method . . . 3
1.5.1 Alternate Advective Methods . . . 3
1.5.2 Velocity Field Fidelity . . . 4
1.6 Thesis Goals and Structure . . . 4

2 Theory **6**
2.1 Navier-Stokes: Velocity-vorticity form . . . 6
2.2 Velocity Field Evaluation: The Biot-Savart Integral . . . 7
2.3 Kernel Desingularization . . . 8
2.4 Discontinuous Galerkin . . . 11

3 Methodology **15**
3.1 Preliminary Solver Overview . . . 15
3.2 Method Specific DG Choices . . . 16
3.3 Interpolation Node Choice . . . 16
3.4 Line-DG . . . 18
3.5 Flux Interpolation . . . 19
3.6 Modified Quadrature . . . 21

4 Implementation **25**
4.1 Overall Solver Structure . . . 25
4.2 Structured Tensor Mesh . . . 28
4.3 Boundary Conditions . . . 29
4.4 Mass Matrix . . . 30
4.5 Stiffness Matrix . . . 30
4.6 Velocity Evaluation . . . 32
4.6.1 Biot-Savart Kernel Calculations . . . 33
4.6.2 Convergence . . . 36

4.7 Vorticity Sparseness . . . 39
4.8 Explicit Time-Stepping . . . 39

5 Results **42**
5.1 Validating Test Cases . . . 42
5.1.1 Perlmann: Stationary Vortex . . . 42
5.1.2 Strain: Interacting Vortex Patches . . . 42
5.1.3 Koumoutsakos: Elliptical Vortex . . . 43
5.2 Perlmann: Stationary Vortex . . . 45
5.2.1 Comparison of Biot-Savart Kernel Effects . . . 45
5.2.2 Cutoff Radius Effects . . . 46
5.2.3 Stage vs. Step-wise Velocity Evaluation . . . 48
5.2.4 Reduced Order Velocity . . . 50
5.2.5 Convergence Rates of Various Orders . . . 51
5.2.6 Comparison of L^2 Error . . . 53
5.2.7 Conservation of Vorticity . . . 55
5.3 Strain: Interacting Vortex Patches . . . 56
5.3.1 Comparison with Originally Published Results . . . 56
5.3.2 Stage vs. Step-wise Velocity Evaluation . . . 58
5.3.3 Approximated Convergence Rates of Various Orders . . . 59
5.3.4 Comparison of L^2 Error . . . 61
5.3.5 Conservation of Vorticity . . . 62
5.4 Koumoutsakos: Elliptical Vortex . . . 62
5.4.1 Comparison with Originally Published Results . . . 62
5.4.2 Stage vs. Step-wise Velocity Evaluation . . . 64
5.4.3 Approximated Convergence Rates of Various Orders . . . 64
5.4.4 Comparison of L^2 Error . . . 66
5.4.5 Conservation of Vorticity . . . 67
5.5 Runtime Speed and Memory Footprint of Solver . . . 68

6 Discussion **69**
6.1 Validation . . . 69
6.1.1 Analytical . . . 69
6.1.2 Qualitative . . . 69
6.2 Convergence Rate . . . 71
6.2.1 Decay of Convergence Rate . . . 72
6.2.2 Effect of Flow Conditions . . . 73
6.2.3 Generation of Non-physical Artifacts . . . 73

7 Conclusion **74**
7.1 Summary . . . 74
7.2 Recommendations . . . 74
7.2.1 Extension to 3D . . . 74
7.2.2 Flux Limiters . . . 75
7.2.3 Impinging Geometry and Boundary Conditions . . . 75

7.2.4 Viscosity 75
7.2.5 Implicit and Local Time Stepping 76
7.2.6 Parallelizability 76
7.2.7 FMM Improvements 76
7.2.8 2D Non-classical Quadrature for Nearly Singular Integrands . 76

8 Bibliography **77**

Appendices **84**

A Notable Code and Algorithms **84**
A.1 Modified Two-part quadrature 84
A.2 Binary Singleton Expansion Matrix-Vector Products 85
A.3 Lagrange Bases and Constructed Quadrature 86
A.4 Velocity Evaluation 87
A.5 Semi-discrete System Construction 87
A.6 Low-storage Stability Optimized Runge-Kutta 88

List of Tables

3.1 Convergence of Modified Quadrature for a 7/9th order Method 24

4.1 Relative error in computed velocities for the self-influence of an element; 5th order quadrature method for a Gaussian vorticity distribution, K_{WL}, $\delta = 0.35h$ 36

4.2 Relative error in computed velocities for an element two elements away from the source; 5th order quadrature method for a Gaussian vorticity distribution, K_{WL}, $\delta = 0.35h$ 36

5.1 Interacting Vortex Patch Parameters 43

List of Figures

3.1 The splitting of nodal contributions along "line-wise" directions on a tensor grid. 19

4.1 "Line-wise" numbering for elements 1, 5, and 9 in a 5x4 element global domain. 29
4.2 5^{th} order Lobatto velocity nodes along x-lines (dark circles) and y-lines (empty circles). 31
4.3 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). 34
4.4 True product of example ω and K_δ. 37
4.5 Interpolated product of example ω and K_δ. 38
4.6 Error between true and interpolated products. 38

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 degrees of freedom 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 [44, 41, 46]. 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 [39, 40]. Direct summation for the velocity field has a $\mathcal{O}(N^2)$ complexity. Tree-codes [38] and Fast Multipole Methods (FMM) [37] are common ways of achieving a more efficient calculation.

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 [37].

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 [42].

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 [31] 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 [30] 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 Biot-Savart integral.

1.6 Thesis Goals and Structure

The overarching thesis goal is the development of a high-order solver capable of modeling inviscid incompressible vorticity-dominated flows in 2D. For simplicity only unconstrained flow free of solid bodies within the domain is considered in the present solver. In order to accomplish this several ancillary goals must be achieved: construction of a high-order advective solver capable of mixed order flux handling, and a high-order Biot-Savart evaluation routine that is as efficient as can be achieved within the bounds of a direct summation approach.

At it's conclusion the chief contributions will be: a complete high-order method for velocity-vorticity inviscid flow that integrates all necessary subsystems, validation of the solver and the underlying Eulerian vortex approach; and evaluation of the convergence, error, and performance of the method and solver.

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 [30] style method will also be reviewed.

In Chapter 3 the methodology taken to construct a high-order Eulerian velocity-vorticity is covered. Chief among the concerns are the spatial basis, choice of interpolation, and numerical integration of the interpolation.

Chapter 4 covers solver specific implementation details. An introductory overview of the solver structure is first laid out. Then, formation of the semi-discrete system, along with the development of some specialized tools are 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.

In Chapter 5 a test plan is laid out for validation and investigation of the overall solver and method. Results of the validation tests, convergence studies, and diagnostics are presented. Additionally runtime performance of the solver is examined, dependent on solver parameters, tolerances, order, etc.

In Chapter 6 discusses the successful validation of the method, as well as the high-order convergence rates achieved. The dependency of a successful solution upon flow conditions and generation of small non-physical artifacts are also discussed.

Chapter 7 concludes the thesis as well as suggests several next steps that should be taken to improve the generality and performance of the method.

Appendix A 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

The Navier-Stokes momentum equation is

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u} + \tfrac{1}{3}\,\mu\nabla(\nabla\cdot\mathbf{u}) \tag{2.1}$$

where u is the velocity field, p is the pressure field, and ρ is the density. If we restrict ourselves to incompressible flows and define the quantity *vorticity* as

$$\omega = \nabla\times\mathbf{u} \tag{2.2}$$

Then the traditional form of the Navier-Stokes equations can be recast

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

where we collect viscous generation of vorticity in S.

There are several benefits to the recast $\mathbf{u}$-ω form:

1. Explicit conservation of 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.

2. 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.

3. 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.

For 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 Eqn. (2.3) 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

$$\frac{\partial \omega}{\partial t} + u \cdot \nabla \omega = S(x,t) \tag{2.4}$$

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

$$\frac{\partial \omega}{\partial t} + \frac{\partial f}{\partial x_i} = S(x,t) \tag{2.5}$$

2.2 Velocity Field Evaluation: The Biot-Savart Integral

Since ω is the quantity of interest in the flowfield solution, the question must be posed: how does one determine the velocity field? For an incompressible 2D or 3D flow we can relate the velocity and vorticity by

$$\nabla^2 u = -\nabla \times \omega \tag{2.6}$$

If inverted, the Biot-Savart integral is obtained

$$u(x^*) = \int_\Omega K(x^*, x) \times \omega(x) dx \tag{2.7}$$

where x^* is the point we wish to evaluate the velocity, x is the coordinate in regions of non-zero vorticity, and $K(x^*, x)$ is the singular Biot-Savart kernel [36]

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

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 vorticity 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 degrees of freedom of our vorticity approximation, then it is expected the velocity calculations will 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 [48], tree-codes [38, 49], and FMM [50] are all possibilities. However, because the velocity field calculation method is decoupled from the PDE governing vorticity advection, 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.

2.3 Kernel Desingularization

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 negligible. Generally, self-influence merely results in solid body rotation. This is the approach

taken by early methods [4]. While easy to implement, it results in large, non-physical velocities when two point vortices approach near each other. The other issue with this approach is poor convergence and accuracy of the resulting velocity field off particle paths as described by Beale and Majda [36].

One may attempt to desingularize the Biot-Savart kernel by introducing a core function $\eta(^{z}/_{\delta})$. Heuristically, the point vortex is now replaced by a finite size vortex blob described by $\eta(r)$, with characteristic radius δ. Analytically, the point vortex is a delta function; core functions are desingularized approximations that converge to the delta function as $\delta \to 0$. Traditionally the core function is convolved with the Biot-Savart kernel to yield a desingularized kernel K_δ.

$$K * \eta(r) = K_\delta \tag{2.9}$$

The choice of a core function and a characteristic radius has important implications on the accuracy and convergence of a Lagrangian point vortex method. Choosing a cutoff radius too small and there is insufficient smoothing, too large and the vorticity discretization is spatially smeared. The choice of a core function is also nuanced: good approximations of the delta function should preserve both the volume and moments of the delta function [36].

Strictly positive core functions can only preserve the first moment, those preserving more may become negative in small regions. The more moments that are preserved, the higher order the convergence of the underlying method (assuming adequate choice of cutoff radius and sufficiently smooth flow). One may preserve all moments, permitting spectral convergence, with the caveat being the resulting core function does not have compact support [34]. This precludes the use of a tree-code of FMM. There are a wide range of desingularized kernels available, those tested in the solver are presented here.

The classical choice is the Rosenhead-Moore kernel [39, 40]. Note that strictly

speaking this kernel does not satisfy convergence proofs as it does not conserve any moments. It is technically a 0th order kernel.

$$K_{RM} = \frac{1}{2\pi}\frac{x^* - x}{|z|^2 + \delta^2} \tag{2.10}$$

where we have substituted $z = x^* - x$.

In [35] Winckelmans and Leonard derived a high-order algebraic kernel that preserves the first moment of the delta function, with second-order formal convergence.

$$K_{WL} = \frac{1}{2\pi}\frac{(z)(|z|^2 + 2\delta^2)}{(|z|^2 + \delta^2)^2} \tag{2.11}$$

In [36] Beale and Majda extend the super-gaussian core function to explicitly generate arbitrary m^{th} order core functions of the type

$$K_{BM} = \frac{z}{2\pi|z|^2}(1 - Q^{(m)}(z/\delta)e^{-z^2/\delta^2}) \tag{2.12}$$

where $Q^{(m)}(r)$ are Laguerre polynomials of r^2 normalized to set the first term to 1.

The final set of kernels used are the spectrally convergent set derived from Bessel functions of the first kind. The simplest appears in [35] as

$$K_{spectral_0} = \frac{z}{2\pi|z|^2}(1 - J_0(\frac{z}{\delta})) \tag{2.13}$$

with a more involved alternative from [33]

$$K_{spectral_2} = \frac{z}{2\pi|z|^2}\left[1 - \frac{8}{45(z/\delta)^2}(4J_2(\frac{4z}{\delta}) - 5J_2(\frac{2z}{\delta}) + J_2(\frac{z}{\delta}))\right] \tag{2.14}$$

these have the advantage that *all* moments are conserved, but they do not have compact support. They are also quite expensive to evaluate due to the Bessel evaluations. For a Lagrangian method this is a serious disadvantage (especially if one wishes to employ a FMM); however for an Eulerian method the vorticity solution points are fixed, so the kernel values need only be calculated once and re-used.

The high-order Eulerian approach taken here means that the Biot-Savart inte-

gral 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 can be constructed *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.

2.4 Discontinuous Galerkin

In order to solve Eqn. (2.15) we adopt a method-of-lines approach [32]. 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 Eqn. (2.15) 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 Eqn. (2.15), that we shall denote as $\widetilde{\omega}(x,t)$. Some residual will remain for the approximate solution of the PDE at any time t

$$\frac{\partial \widetilde{\omega}}{\partial t} + \frac{\partial f}{\partial x_i} = R(x) \tag{2.15}$$

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:

$$\int_\Omega R(x)\phi_j \, dx = 0 \quad \text{for all } j \tag{2.16}$$

Substituting the residual for our conservation PDE yields:

$$\int_\Omega \frac{\partial \widetilde{\omega}}{\partial t} \, \phi_j \, dx + \int_\Omega \frac{\partial f(\widetilde{\omega})}{\partial x} \, \phi_j \, dx = 0 \quad \text{for all } j \tag{2.17}$$

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

$$\omega(x,t) \approx \widetilde{\omega}(x,t) = \sum_{i=0}^{M} a_i(t)\psi_i(x) \tag{2.18}$$

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

$$\sum_{i=0}^{M} \left[\frac{da_i(t)}{dt} \int_{x_L}^{x_R} \psi_i(x) \, \phi_j(x) \, dx \right] + \int_{x_L}^{x_R} \frac{\partial f(\widetilde{\omega})}{\partial x} \, \phi_j \, dx = 0 \tag{2.19}$$

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

$$\sum_{i=0}^{M} \left[\frac{da_i(t)}{dt} \int_{x_L}^{x_R} \psi_i(x) \, \phi_j(x) \, dx \right] + f\phi_j(x)\Big|_{x_L}^{x_R} - \int_{x_L}^{x_R} f(\widetilde{\omega}) \, \frac{d\phi_j(x)}{dx} \, dx = 0 \tag{2.20}$$

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 $[[\omega]] = \omega^+ - \omega^-$ [31], yields

$$\hat{f}_{upwind}(x^+, x^-) = u\{\{\widetilde{\omega}\}\} + \frac{|u|}{2}[[\widetilde{\omega}]] \tag{2.21}$$

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

$$x = g(X) = \frac{X+1}{2}\Delta x + x_L \quad , \quad X = G(x) = \frac{2(x - x_L)}{\Delta x} + 1 \tag{2.22}$$

Applying a change of variables using the mapping to Eqn. (2.23), 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:

$$\frac{\Delta x}{2}\sum_{i=0}^{M}\left[\frac{da_i}{dt}\int_{-1}^{1}\psi_i\,\phi_j\,dX\right] + \hat{f}\phi_j\Big|_{x_L}^{x_R} - \int_{-1}^{1} f(\widetilde{\omega})\,\frac{d\phi_j}{dX}\,dX = 0 \tag{2.23}$$

(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 §3.2. 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

Solver specific implementation details are deferred until the next chapter while we develop method specific choices and numerical machinery. However it is instructive to briefly outline the programmatic flow of the solver.

3.1 Preliminary Solver Overview

```
Define problem parameters
Define solver parameters
Calculate derived solver parameters
Setup intitial conditions
Initialize solver
%Time stepping
for t=0 to end
    if datalog?=yes
       save system state to file and plot
    end
    %Loop through RK stages
    for s=1 to last_stage
       %For elements above threshold
       for each vorticity source
          calculate velocity contributions
       end
       %Calculate semi-discrete system terms
       interpolate boundary_vorticity
       calculate numerical_fluxes
       calculate total_surface_flux
       calculate internal_stiffness_flux

       vorticity_rate_of_change=...
          internal_stiffness_flux - total_surface_flux

       RK_stage= (RK_coeff_a*RK_stage) +...
          (time_step * vorticity_rate_of_change)
       vorticity= vorticity + RK_coeff_b * RK_stage
    end
```

end

3.2 Method Specific DG Choices

We now seek to take the general form of Eqn. (2.23) 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 the basis is constructed for 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 then each can be treated separately. 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:

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

3.3 Interpolation Node Choice

It is well known that for moderate orders equispaced interpolation nodes are a poor choice that will suffer from Runge's Phenomenon. To avoid this, nodes that are the roots of an orthogonal polynomial family are commonly adopted; Chebyshev,

Legendre, and Legendre-Lobatto are all common choices [58].

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 [58]. 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 permissible, 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

$$\begin{aligned}
&\int \ell_i(x)\ell_j(x)dx = \int \frac{1}{c_1}\prod_{p\neq i}^{M}(x-x_p)\frac{1}{c_2}\prod_{q\neq j}^{M}(x-x_q)\ dx \\
&= \frac{1}{c_3}\int \Big[(x-x_0)...(x-x_{i-1})(x-x_{i+1})...(x-x_M)\Big]\prod_{p\neq j}(x-x_p)\ dx \\
&= \frac{1}{c_3}\int \Big[(x-x_0)...(x-x_{i-1})\boldsymbol{(x-x_i)}(x-x_{i+1})...(x-x_M)\Big]\frac{1}{\boldsymbol{(x-x_i)}}\prod_{p\neq j}(x-x_p)\ dx \\
&= \frac{1}{c_3}\int P_M(x)\pi_{M-2}(x)\ dx = 0
\end{aligned} \tag{3.3}$$

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 of the Legendre terms

that appear in this sum are lower order than P_M (and therefore orthogonal to P_M), 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

$$\int \ell_i(x)\ell_j(x)dx = \delta_{ij}w_j \tag{3.4}$$

3.4 Line-DG

An important consequence of the tensor basis means that the two basis directions are not coupled except at the interpolation nodes at the intersection of each direction's basis. This is the central idea in the line-DG [30] approach, a 2D problem is transformed into two 1D problems of the form of Eqn. (2.23). 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 use the developed 1D basis in a tensor product to form our 2D basis

$$f(x,y) \approx \left[\sum_{j=0}^{L} z_j\ell_j(y)\right] \times \left[\sum_{i=0}^{M} z_i\ell_i(x)\right] = \sum_{j=0}^{M}\sum_{i=0}^{M} z_{ij}\ell_j\ell_i = \sum_{j=0}^{M} z_{ij}\ell_j \sum_{i=0}^{M}\ell_i \tag{3.5}$$

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

We now substitute our basis into Eqn. (2.23) to get a particular direction's PDE

$$\frac{\Delta x}{2}\sum_{i=0}^{M}\left[\frac{dz_{ij}}{dt}\int_{-1}^{1}\ell_i\,\ell_j\,dX\right] + \hat{f}\ell_j\Big|_{x_L}^{x_R} - \int_{-1}^{1} f(\widetilde{\omega})\,\ell_j'\,dX = 0 \tag{3.6}$$

We solve the PDE "line-wise"; we form the tensor product of the chosen 1D interpolation nodes, as shown in 3.1. In this example we would solve 10 $\times$ 1D problems: 5 along the x-direction and 5 along y-direction. The solution of Eqn. (3.6) for any one of these 1D problems yields the rate of change of vorticity for all nodes along

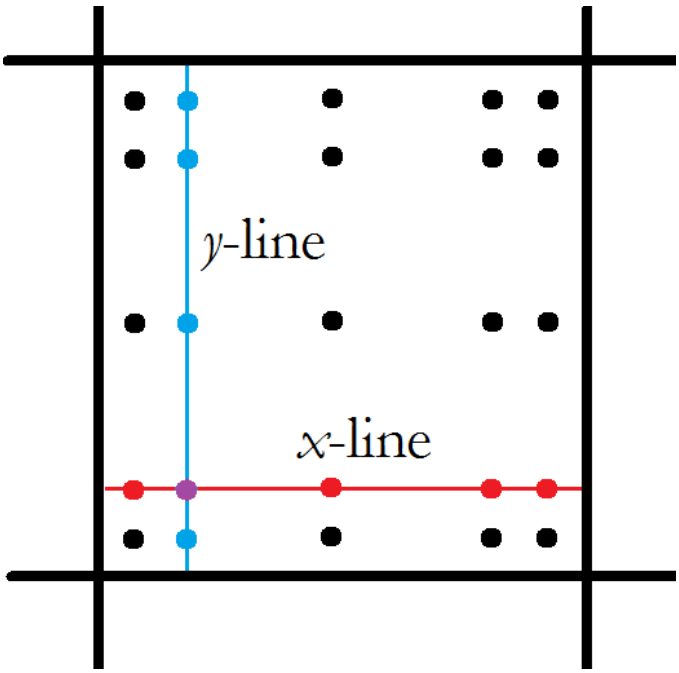

Figure 3.1: The splitting of nodal contributions along "line-wise" directions on a tensor grid.

that line. This only accounts for advection along one-direction; the complimentary perpendicular line intersecting at a node on the first line also contributes. In example figure 3.1 the rate of change of the vorticity at the node at the intersection of the two lines is the sum of rates of changes from each line. In general the rate of change at a particular node is:

$$\frac{\partial \omega_{ij}}{\partial t} = (\frac{\partial \omega_{ij}}{\partial t})_{x-line} + (\frac{\partial \omega_{ij}}{\partial t})_{y-line} \tag{3.7}$$

3.5 Flux Interpolation

The flux interpolant can be thought of as either the interpolation of the product of the velocity and vorticity, or the product of the interpolations of velocity and vorticity; the result of these two interpolants are identical at the nodes

$$I_N f(x_i) = I_N(u(x_i)\omega(x_i)) = I_N(u(x_i))I_N(\omega(x_i)) \tag{3.8}$$

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

$$\int f\ dx \approx \sum_{i}^{M} u_i \omega_i w_i = \sum_{i}^{M} u_i \sum_{j}^{M} \omega_j \int \ell_i \ell_j \tag{3.9}$$

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 may appear to be inconsequential. Consider however the case where we wish to integrate the product of the interpolated flux and the derivative of the weighting function. The interpolation of the product is order M and the derivative of the weighting basis M-1, the collocated Gauss-Legendre quadrature is exact

$$\int I_N(f)\, \ell_j'(x)\ dx = \sum_{i}^{M} u_i \omega_i \ell_j'(x_i) \tag{3.10}$$

There is an important decision that has been implicitly made however. By choosing the quadrature of this form we are choosing 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

$$\begin{aligned} \int u(x)\omega(x)\ell_j'(x)\ dx &\approx \int \sum_{i}^{M} u_i \ell_i(x) \sum_{k}^{M} \omega_k \ell_k(x)\ \ell_j'(x)\ dx \\ &= \sum_{i}^{M} u_i \sum_{k}^{M} \omega_k \int \ell_i(x)\ell_k(x)\ell_j'(x)\ dx \end{aligned} \tag{3.11}$$

We will do this in the next section.

3.6 Modified Quadrature

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

$$\sum_{i}^{M} u_i \sum_{k}^{M} \omega_k \; w_{unknown} = \sum_{i}^{M} u_i \sum_{k}^{M} \omega_k \int \ell_i(x)\ell_k(x)\ell'_j(x) \; dx \tag{3.12}$$

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

$$\sum_{i}^{M} u_i \sum_{k}^{M} \omega_k \sum_{q}^{M} \ell'_j(x_q) \; w_{unknown2} \tag{3.13}$$

However a better approach is to keep the derivative weighting basis inside the integral. The derivative of the weighting basis can be derived analytically by use of logarithmic differentiation

$$f' = f\,[ln(f)]' \rightarrow \ell'_j(x) = \ell_j(x) \sum_{\substack{p=0 \\ p \neq j}}^{M} \frac{1}{(x - x_p)} \tag{3.14}$$

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)

$$\int \ell_i(x)\ell_k(x)\ell'_j(x) \; dx = \mathbf{W}_{ikj} \tag{3.15}$$

The integral can be performed numerically using a 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

$$\int u(x)\omega(x)\ell'_j(x)\,dx \approx \mathbf{u}_i^T \mathbf{W}_{ikj}\,\boldsymbol{\omega}_k \tag{3.16}$$

for each jth weighting basis function.

This may seem to incur a high cost, until one considers several facts:

- Normal Gauss-Legendre quadrature requires the stiffness integral be calculated for each weighting basis already, so the j dimension of the quadrature matrix is no worse than before.

- 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.

- 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.

- 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 significant advantages for the overall method, both in terms of minimizing both 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 contributes

additional interpolation error, as well as potential aliasing problems. This additional interpolation step would also require 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 results in 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.

The maximum order polynomial that can be integrated exactly is also similar to some composite quadrature rules (such as Gauss-Kronrod [58]): 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 Appendix A validates the convergence claims and order of accuracy of the modified quadrature method.

Table 3.1 reports an example convergence study for a mixed order/type run with an 7th order Gauss-Legendre and 9th order Gauss-Lobatto quadrature rule showing

Table 3.1: Convergence of Modified Quadrature for a 7/9th order Method

Order M+N	IoP	PoI	Quad. Error	Discretiz. Error	Order M
5	-0.0543	0.234	2.22e-16	0.234	3
7	0.0162	0.0182	-5.55e-16	0.0182	4
9	-0.0143	0.0493	-1.67e-16	0.0493	5
11	0.00342	0.00133	2.22e-16	0.00133	6
13	-0.00182	0.00375	5.00e-16	0.00375	7
15	0.000445	9.38e-05	5.55e-16	9.38e-05	8
17	-0.000161	0.000139	8.33e-16	0.000139	9
19	4.20e-05	6.15e-06	0.000326	0.000332	10
21	-1.11e-05	2.90e-06	0.000523	0.000526	11
23	3.15e-06	3.37e-07	-9.75e-05	-9.72e-05	12

quadrature error to within machine precision up to order $M+N+1$. It also compares the error in the interpolation of product (IoP) and product of interpolants (PoI), with the latter being an order of magnitude more accurate for this set of example functions.

4 Implementation

To provide a clear picture of the overarching structure of the solver we restate the overview of §3.1, but now add in specific solver subroutines and more specific details on the required steps.

4.1 Overall Solver Structure

```
Define problem parameters [domain size, initial condition type]
Define solver parameters [order velocity/vorticity, element size,
   time stepping choice, time step, cutoff radius, minimum
   vorticity threshold, kernel type]
Calculate derived solver parameters
   -interpolation nodes/weights
   -modified quadrature matrix
   -node numbering/associativity
   -node coordinates
Setup intitial conditions
Initialize solver
   -vorticity interpolation vectors
   -calculate generalized boundary kernel
   -calculate nearby elemental kernel values
   -calculate quadrature weight outer product for later
      use in pre-multiplication for vorticity integration

for t=0 to end
   if Log?=yes
      save system state to file and plot
   end

   for s=1 to last_stage (NRK14C method)
      Calculate elemental vorticity sums and create mask of
         those greater than threshold
      Pre-multiply vorticity with quad. weight outer product

      Calculate un-assigned nearby elemental velocities for
         vorticity source>threshold
```

```
        for each vorticity source>threshold
           -Assemble global kernel of boundary values from
              generalized kernel
           -Calculate global boundary velocities (for surface
              flux use)
           -Correct global boundary values to generate element
              boundary values from far field sources (for
              far field stiffness use)
           -Assign nearby elemental velocities to specific
              element, merge with corrected element boundary
              values for near field sources (for near field
              stiffness use)
        end

        interpolate boundary_vorticity
        calculate numerical_fluxes
        calculate total_surface_flux
        calculate internal_stiffness_flux

        vorticity_rate_of_change=...
           internal_stiffness_flux - total_surface_flux

        RK_stage= (RK_coeff_a*RK_stage) +...
           (time_step * vorticity_rate_of_change)
        vorticity= vorticity + RK_coeff_b * RK_stage
    end
end
```

We briefly elaborate on the main parts of the implementation, the sections following give full detail for each major part.

We will use terminology commonly used in Finite Element Methods to refer to specific pieces of the 1D PDE solved along each line from Eqn. (3.6); the integral involved in the time derivative generates the *mass* matrix.

$$\boldsymbol{M} = \frac{\Delta x}{2} \sum_{i=0}^{M} \left[\frac{dz_{ij}}{dt} \int_{-1}^{1} \ell_i \, \ell_j \, dX \right] \tag{4.1}$$

The integral involved in the spatial derivative of the weighting function generates the *stiffness* matrix.

$$\boldsymbol{K} = \int_{-1}^{1} f(\widetilde{\omega}) \, \ell_j' \, dX \tag{4.2}$$

The total surface flux is the net flow across element bounds into a particular line. It is calculated from the numerical flux and the weighting basis evaluated at the line boundaries.

$$\boldsymbol{F} = \hat{f}\ell_j\Big|_{x_L}^{x_R} \tag{4.3}$$

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 as was described in §3.6.

The mass matrix will be shown to be diagonalizable in §4.4 allowing it to be decoupled from the time derivative. This allows the quadrature matrix to be multiplied 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.

The next part of the solver creates element-line-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 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. This is more full described in §4.2.

Next, 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 mass entry like in the quadrature weight matrix to avoid element-wise multiplications later. Finally the "generalized" and "nearby" kernel evaluation matrices are constructed for later re-use as described in §4.6.

At this point the remainder of the code occurs inside the main time marching loop. A fourth-order, 14 stage, 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 as described in §4.8. The sum of the vorticity in each element is computed and those above a threshold are added to element-wise and line-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 (§4.5). 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 diagnostics (L^2 norm, vorticity conservation etc.).

4.2 Structured Tensor Mesh

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

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 "line-

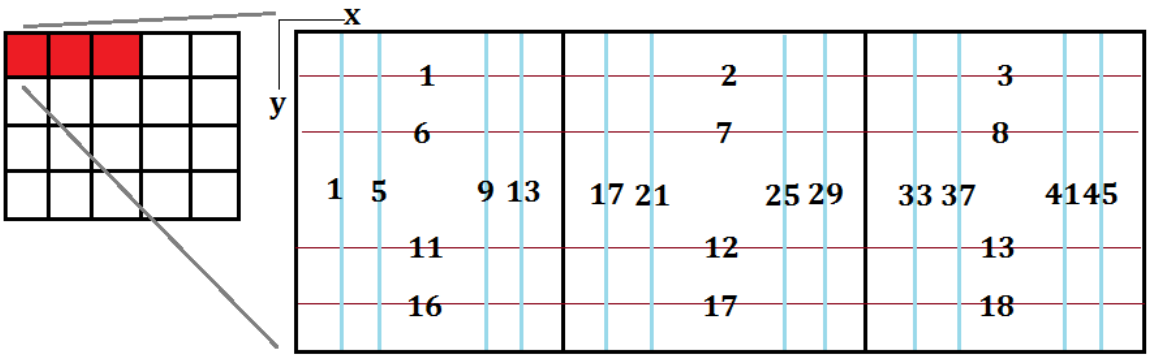

Figure 4.1: "Line-wise" numbering for elements 1, 5, and 9 in a 5x4 element global domain.

wise" format. The line-wise format groups nodes along the coordinate directions. Adjacent lines are parallel 1D basis functions that share a boundary node. Lines are numbered left to right, bottom to top for x-lines and bottom to top, left to right for y-streams. Both direction's lines are numbered separately. For an example layout of x and y lines see Figure 4.1. The inset only shows the upper left chunk of a 5x4 global domain. The lowest two y-lines shown aren't sequential because one needs to continue counting in the y direction from 1, there are 3 more elements with the left most y-line in each numbered '2,3,4' sequentially. Once we reach the top of the global domain the numbering wraps back around to the bottom, second leftmost y-stream: '5'.

4.3 Boundary Conditions

Boundary conditions are typically handled automatically. The global domain of the model is sized so that no vorticity is advected past the boundary. Should vorticity still manage to pass the boundary it will be lost thanks to a 'NoInflow' condition; this is imposed by enforcing all surface flux along the domain boundary to be 0. Any vorticity that does reach the boundary is destroyed.

A set of periodic boundary conditions were also created that permit study of toroidal or infinite cylinder domains that may be useful in the study of semi-infinite

phenomena such as the Kelvin-Helmholtz instability on a infinite line of vorticity.

4.4 Mass Matrix

As discussed in §3.3, the set of Legendre roots for the vorticity interpolation points results in

$$\int \ell_i(x)\ell_j(x)dx = \delta_{ij}w_j \tag{4.4}$$

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. Eqn. (3.6) is then simplified to

$$\frac{d\widetilde{\omega}_j}{dt}\frac{w_j\Delta x}{2} + \hat{f}\ell_j\Big|_{x_L}^{x_R} - \int_{-1}^{1} f(\widetilde{\omega})\,\ell_j'\,dX = 0 \tag{4.5}$$

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.

4.5 Stiffness Matrix

We can apply the results for the modified quadrature rule from Eqn. (3.16) 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-line" interpolation the velocity nodes should be co-linear with the vorticity lines. The result for a 5th order velocity and vorticity field is shown in Figure 4.2.

It would appear we actually need double the velocity evaluation points. The non-

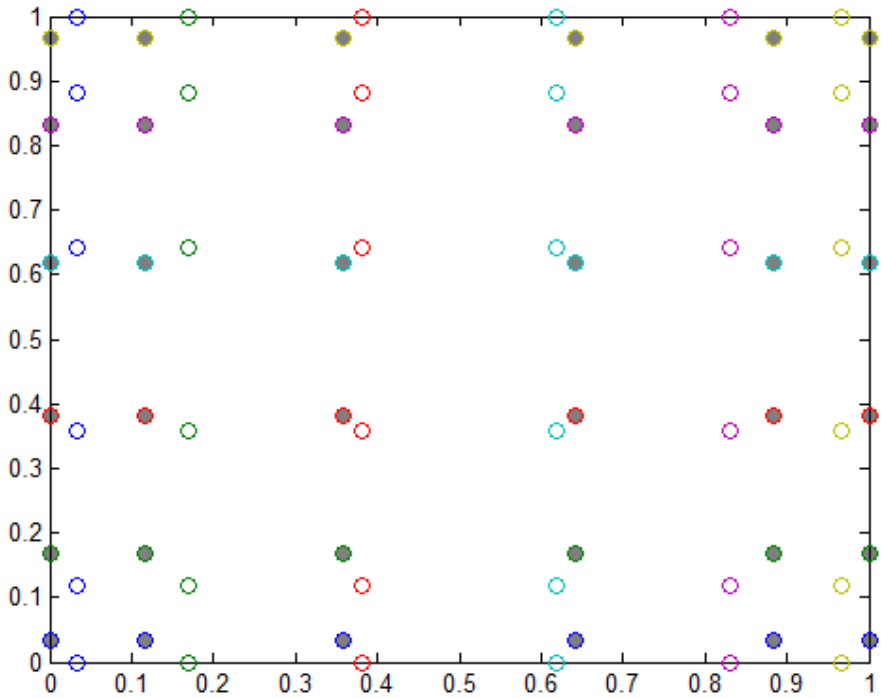

Figure 4.2: 5^{th} order Lobatto velocity nodes along x-lines (dark circles) and y-lines (empty circles).

square tensor product of the Lobatto nodes with the Legendre nodes means that the x-line velocity nodes will never coincide with the y-line velocity nodes. The solution to this is that the velocity is evaluated in a component-wise manner. Along the x-line we only require u_x and along the y-line we only require u_y. The different direction's line's nodes may not coincide with each other, however this doesn't pose any problems as we do not need need the other velocity component at any particular velocity node. A small caveat is that if we would like to calculate the full velocity field this must be added in as an additional post-processing step to calculate the skipped velocity components.

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

$$\frac{d\widetilde{\omega}_j}{dt}\frac{w_j\Delta x}{2} + \hat{f}\ell_j\Big|_{x_L}^{x_R} - \mathbf{u}^T\mathbf{W}_{--j}\,\boldsymbol{\omega} = 0 \tag{4.6}$$

rearranging to obtain the rate of change of the nodal vorticity

$$\frac{d\widetilde{\omega}_j}{dt} = \frac{2}{w_j\Delta x}\left[\mathbf{u}^T\mathbf{W}_{--j}\,\boldsymbol{\omega} - \hat{f}\ell_j\Big|_{x_L}^{x_R}\right] \tag{4.7}$$

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

4.6 Velocity Evaluation

It is important to note that direct pair-pair velocity evaluation has $\mathcal{O}(N^2)$ complexity; it is well known that methods exist that can reduce this to $\mathcal{O}(N\log N)$ or even $\mathcal{O}(N)$ [48, 38, 49, 50]. However for ease of implementation and to eliminate the effects of as much approximation error as possible on validation, the velocity is evaluated directly at each desired velocity point by means of the desingularized Biot-Savart integral (Eqn. (2.9)).

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 Biot-Savart integral is done via a 2D tensor product Gauss-Legendre quadrature of order equal to the vorticity field with quadrature points collocated at the vorticity interpolation points.

$$u(x^*) = \sum_{E=1}^{N_{mask}} [\omega_{pre}]^T K_\delta (x^* - [x_E]) \tag{4.8}$$

$$\omega_{pre} = [\omega(x_E)]. * [w_i \otimes w_j] \tag{4.9}$$

Items in brackets are column vectors and $.*$ indicates element-wise multiplication. ω_{pre} is the vorticity for each element in the mask pre-multiplied by the outer product of the two tensor direction's 1D quadrature weights. As a given mask element appears as a source for all velocity evaluation points, performing the multiplication by the quadrature weights and caching realizes significant savings. This technique combined with the summation of all quadrature points in a particular source element by a vector-vector product realizes a significant speed-up: over 2000% over the more natural but slower vector-(matrix-matrix)-vector multiplication, $[w_i]^T(\omega. * K_\delta)[w_j]$.

4.6.1 Biot-Savart Kernel Calculations

The majority of the cost of the present solver results from the assembly of the boundary kernel values from the generalized template. 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 degrees of freedom) would require 1.66GB of memory to store:

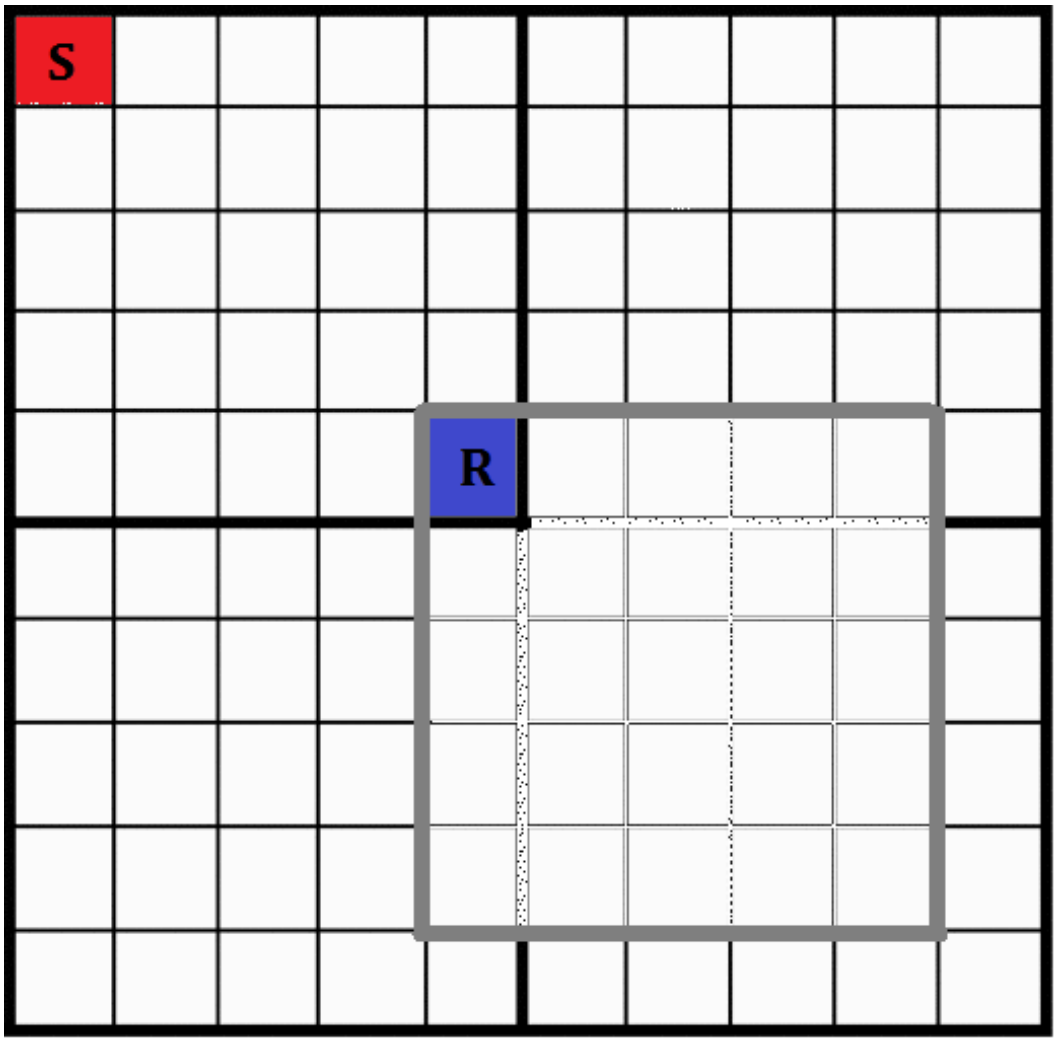

Figure 4.3: The "global" kernel matrix showing (R)eference source and actual (S)ource, the upper left quadrant is the actual full domain, the selected region with respect to (R) selects the correct values of the kernel for (S).

kernel values per target node x number of targets = $[30^2(6^2)][30^2(6^2)]\times$8bytes/double float .

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 4.3. This is implemented in the solver as

```
kernel_xB= ...
    gkernel_xB(:, [1:Np*K(2)] +Np*(K(2)-Enumy(Src)),...
    [1:K(1)+1] +(K(1)-Enumx(Src)) );
kernel_yB =...
     gkernel_yB(:, [1:Np*K(1)] +Np*(K(1)-Enumx(Src)),...
    [1:K(2)+1] +(K(2)-Enumy(Src)) );
```

where K(x,y) is the number of elements, Np is the number of interpolation points along one direction, Enumx/Enumy return the x and y directional numbering of Src (the global element number), and gkernel_xB/gkernel_yB is the generalized kernel values that contain x and y reflections of the domain.

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 accounts for about 60% of CPU time according to the profiler. 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. 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.

Table 4.1: Relative error in computed velocities for the self-influence of an element; 5th order quadrature method for a Gaussian vorticity distribution, K_{WL}, $\delta = 0.35h$

	$LGL(x_1)$	$LGL(x_2)$	$LGL(x_3)$	$LGL(x_4)$	$LGL(x_5)$	$LGL(x_6)$
$GL(x_1)$	0.0014	0.0028	0.0047	0.0011	0.00011	0.00012
$GL(x_2)$	0.0012	0.00096	0.0011	0.0022	9.4e-06	0.00016
$GL(x_3)$	0.0014	0.0051	0.0024	0.0060	0.0037	0.00030
$GL(x_4)$	0.0010	0.0023	0.0018	0.0047	0.0062	0.00037
$GL(x_5)$	0.0021	0.0021	0.0042	0.0060	0.0081	0.00018
$GL(x_6)$	0.0016	0.0042	0.00024	0.0067	0.0035	0.00028

Table 4.2: Relative error in computed velocities for an element two elements away from the source; 5th order quadrature method for a Gaussian vorticity distribution, K_{WL}, $\delta = 0.35h$

	$LGL(x_1)$	$LGL(x_2)$	$LGL(x_3)$	$LGL(x_4)$	$LGL(x_5)$	$LGL(x_6)$
$GL(x_1)$	7.67E-09	6.79E-09	5.88E-09	1.85E-09	2.49E-09	4.38E-09
$GL(x_2)$	9.87E-09	9.28E-09	8.13E-09	2.56E-09	3.39E-09	5.91E-09
$GL(x_3)$	1.49E-08	1.51E-08	1.35E-08	4.00E-09	5.95E-09	9.95E-09
$GL(x_4)$	2.42E-08	2.79E-08	2.42E-08	6.16E-09	1.22E-08	1.91E-08
$GL(x_5)$	3.79E-08	4.58E-08	4.16E-08	8.24E-09	2.45E-08	3.55E-08
$GL(x_6)$	5.08E-08	6.38E-08	5.97E-08	9.02E-09	3.92E-08	5.40E-08

4.6.2 Convergence

For the far field canonical Gauss-Legendre quadrature 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 [44]. As an illustrating point consider the data in Tables 4.1 and 4.2. Compare the relative error for the computed velocities due to self-influence and a source element just two elements away. The difference in the relative error is approximately 5 orders of magnitude. Furthermore the observed error in the two element away case is mostly discretization error: the same integration parameters performed with a dummy kernel of $K = 1$ had a relative error of 5E-10, within an order of magnitude of the actual test case.

The weighting function for the Legendre polynomials is simply $w_{Leg}(x) = 1$. As such, the product of the K_δ and the vorticity near the source element is poorly

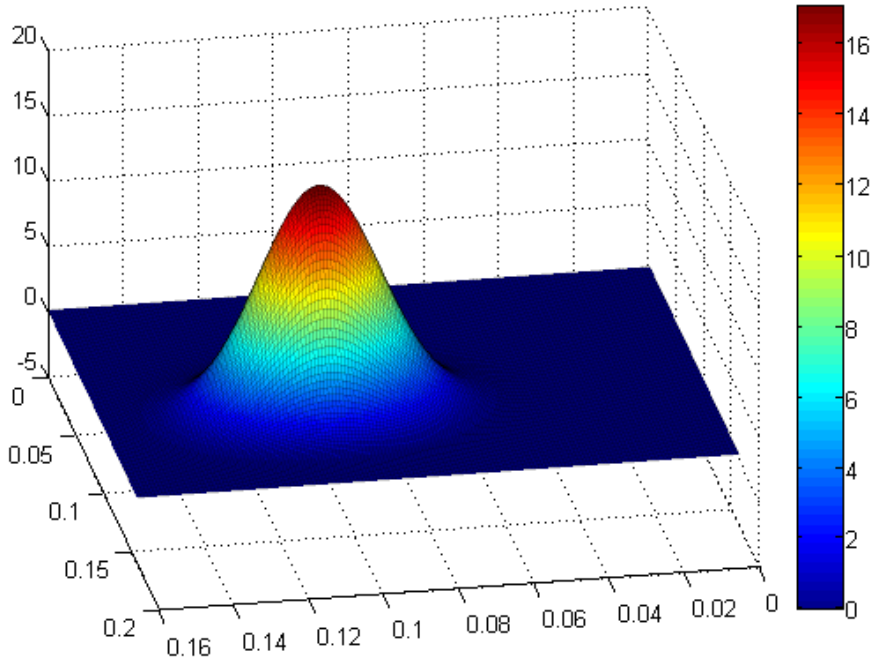

Figure 4.4: True product of example ω and K_δ.

interpolated by a polynomial. To illustrate consider the same setup as the one used in the tabulated velocity errors comparison: plots of the true product of the vorticity and K_{WL}, it's matching interpolation, and the error between them are plotted in Figures 4.4, 4.5, 4.6 respectively.

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 [51, 52, 53].

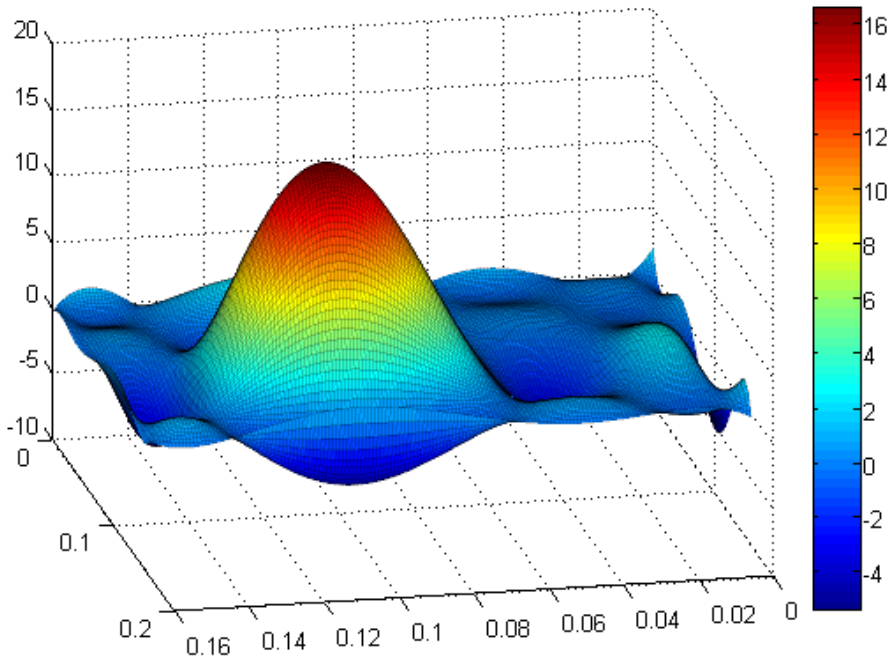

Figure 4.5: Interpolated product of example ω and K_δ.

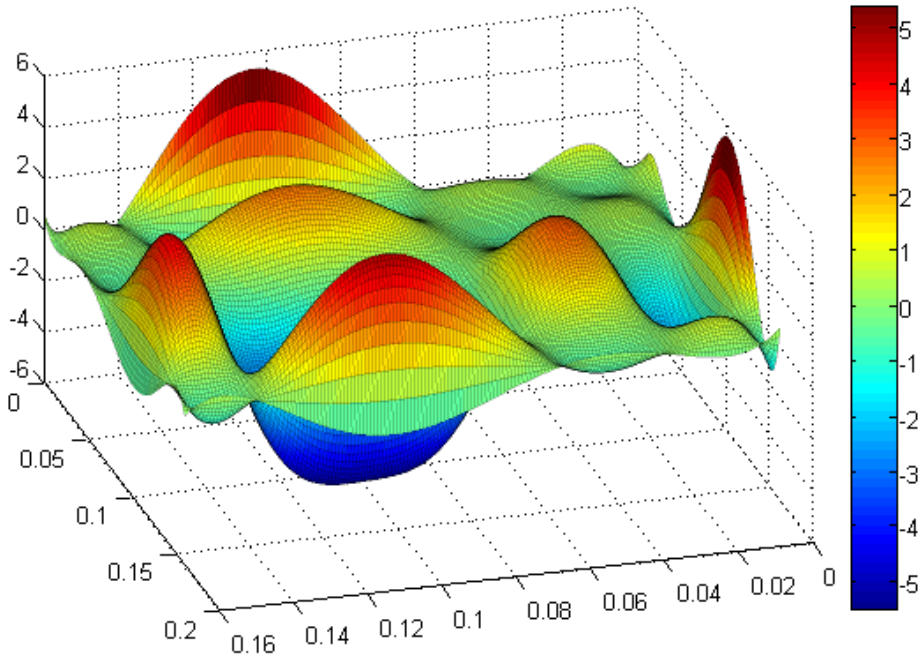

Figure 4.6: Error between true and interpolated products.

4.7 Vorticity Sparseness

One of the advantages of DG as an advection scheme is the compact local support. 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 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 omitted from the 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 no longer be ignored, lest some instability result from improperly handling non-zero elements.

From a vorticity-velocity perspective elements with zero vorticity or boundary flux driven "noise" 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 solution time. The unmasked elements have only a marginal impact on the cost of the velocity evaluations.

4.8 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 diagonalized

mass matrix created in §4.4 permits the compact rearrangement of Eqn. (4.5) to solve explicitly for the nodal rate of change of vorticity due to each of the lines that intersect the node. The contribution from each line is added and the total rate of change is used to advance stagewise within the RK routine:

```
for t=0:delt:EndTime %Step
    for i=1:nS %Stage
        (Velocity calculations)
        (semi-discrete calculations for advection)
        wx_dt= permute(Stiff_x-SurfFlux_x,[4 1 3 2]);
        wy_dt= reshape(reshape(...
        Stiff_y-SurfFlux_y,K(2),[])',Np,1,[]);

        k2= RKa(i)*k2 + delt*(wx_dt+wy_dt);
        wx= wx+RKb(i)*k2;
        wy= reshape(reshape(wx,K(1)*Np,[])',Np,1,[]);
    end
end
```

where delt is the timestep, nS is the number of stage RK method chosen, RKa and RKb are RK specific coefficients, Stiff_x/Stiff_y are the nodal stiffness values, SurfFlux_x/SurffFlux_y are the nodal surface flux values, k2 is the low-storage RK state variable, and wx/wy are line-wise vorticity state variables

The two chief categories are multi-step and multi-stage methods, typically either Adams-Bashforth or Runge-Kutta respectively [61]. It is known that Forward Euler and AB2 are unstable for DG with an upwind flux due to it's dissipative nature [54]. The choices permitted then are Runge-Kutta or a higher-order Adams-Bashforth.

The recent thesis work of Atcheson [54] 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 [55] one of the competitive options was the "NRK14C" scheme developed by Niegemann et al. [56].

This is an optimized 4th order, 14 stage low storage scheme of similar structure to other 2N style schemes [57].

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.

5 Results

5.1 Validating Test Cases

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. The following test cases are evaluated and examined for a range of items of interest: behavior of Biot-Savart kernel choice and cutoff radius, stage vs step-wise velocity evaluation, effect of velocity order, convergence rate, L^2 error, and appropriate conservation of vorticity.

5.1.1 Perlmann: Stationary Vortex

Perlmann's 7th order polynomial was the first case studied [43]. The intial condition is a stationary vortex that permits an analytical solution to compare against. The initial vorticity distribution is of the form:

$$\omega(z) = (1 - |z|^2)^7, \ |z| \leq 1 \quad \omega(z) = 0, \ |z| > 1 \quad z^2 = x^2 + y^2 \tag{5.1}$$

and is invariant with time. This test case is particularly useful as the availability of an analytical solution allows an accurate assessment of the L^2 error of the computed solution.

5.1.2 Strain: Interacting Vortex Patches

Strain's system of interacting random vortex patches [44] forms the next validation case. Specific initial conditions are specified for each of the vortex patches in Table

Table 5.1: Interacting Vortex Patch Parameters

j	x_j	y_j	ρ_j	Ω_j
1	-0.6988	-1.7756	0.6768	-0.4515
2	1.4363	-1.4566	0.3294	0.4968
3	-0.1722	0.4175	0.5807	-0.9643
4	-1.5009	-0.0937	0.2504	0.3418

5.1 with the overall system consisting of the linear superposition of each patch

$$\omega(x,y,0) = \sum_{j=1}^{m} \Omega_j exp(-((x-x_j)^2 + (y-y_j)^2)/\rho_j^2) \tag{5.2}$$

An explicit data set for the results is not reported, but a set of contour plots illustrates the time evolution. The same parameters are used in the present solver and similar contour plots are compared against the validating set.

This test case has several features that make it a useful diagnostic test. The relative motion of each vortex is important in this case to compute an accurate solution, both short and long-range interactions need to be accurately computed. Additionally the onset of a vortex collision occurs, creating strong vorticity gradients which are more difficult to discretize.

5.1.3 Koumoutsakos: Elliptical Vortex

The final validation case is the evolution of an elliptical vortex patch [45]. The specific initial configuration is specified as

$$\omega^{II}(r) = 20(1-(r/0.8)^4),\ r \leq 0.8 \quad \omega^{II}(r) = 0,\ r > 0.8 \tag{5.3}$$

however the cases tested were actually elliptical, to accommodate this the initial distribution was modified to

$$\omega^{II}(x,y,0)_{mod} = 20(1-((x/a)^2 + (y/b)^2)^2/0.8^4) \quad a=1,\ b=2 \tag{5.4}$$

The contour levels are specified in the resultant figure as 0.25, 0.50, 1, 2, ..., 20; these levels are used in the current code's resultant plot for direct comparison.

This test case also has several relevant diagnostic features. The elliptical distribution of the vortex rotates creates weak filamentary structures that are important to properly resolve and accurately advect as they later collide with the main vortex body. Additionally, the initial profile of the vortex has no continuous derivatives at the transition from the vortex core to the surrounding fluid which provides a challenging test for the polynomial basis functions.

5.2 Perlmann: Stationary Vortex

5.2.1 Comparison of Biot-Savart Kernel Effects

5.2.2 Cutoff Radius Effects

.

5.2.3 Stage vs. Step-wise Velocity Evaluation

.

5.2.4 Reduced Order Velocity

5.2.5 Convergence Rates of Various Orders

.

5.2.6 Comparison of L^2 Error

.

5.2.7 Conservation of Vorticity

5.3 Strain: Interacting Vortex Patches

5.3.1 Comparison with Originally Published Results

5.3.2 Stage vs. Step-wise Velocity Evaluation

5.3.3 Approximated Convergence Rates of Various Orders

.

5.3.4 Comparison of L^2 Error

5.3.5 Conservation of Vorticity

5.4 Koumoutsakos: Elliptical Vortex

5.4.1 Comparison with Originally Published Results

.

5.4.2 Stage vs. Step-wise Velocity Evaluation

5.4.3 Approximated Convergence Rates of Various Orders

.

5.4.4 Comparison of L^2 Error

5.4.5 Conservation of Vorticity

5.5 Runtime Speed and Memory Footprint of Solver

6 Discussion

6.1 Validation

6.1.1 Analytical

6.1.2 Qualitative

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

.

6.2 Convergence Rate

.

6.2.1 Decay of Convergence Rate

6.2.2 Effect of Flow Conditions

6.2.3 Generation of Non-physical Artifacts

7 Conclusion

7.1 Summary

7.2 Recommendations

7.2.1 Extension to 3D

7.2.2 Flux Limiters

7.2.3 Impinging Geometry and Boundary Conditions

7.2.4 Viscosity

7.2.5 Implicit and Local Time Stepping

7.2.6 Parallelizability

7.2.7 FMM Improvements

7.2.8 2D Non-classical Quadrature for Nearly Singular Integrands

8 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] Hesthaven, Jan S., and Tim Warburton. Nodal discontinuous Galerkin methods: algorithms, analysis, and applications. Vol. 54. Springer Science & Business Media, 2007.

[32] Cockburn, Bernardo, and Chi-Wang Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. II. General framework. Mathematics of Computation 52.186 (1989): 411-435.

[33] Hald, Ole H. Convergence of vortex methods. Vortex methods and vortex motion. Vol. 1. 1991.

[34] Hald, Ole H. Convergence of vortex methods for Euler's equations, III. SIAM journal on numerical analysis 24.3 (1987): 538-582.

[35] Winckelmans, G. S., and A. Leonard. Contributions to vortex particle methods for the computation of three-dimensional incompressible unsteady flows. Journal of Computational Physics 109.2 (1993): 247-273.

[36] Beale, J. Thomas, and Andrew Majda. High order accurate vortex methods with explicit velocity kernels. Journal of Computational Physics 58.2 (1985): 188-208.

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

[38] 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.

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

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

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

[42] 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.

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

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

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

[46] 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.

[47] 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.

[48] 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.

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

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

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

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

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

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

[55] 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.

[56] 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.

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

[58] Quarteroni, Alfio, and Alberto Valli. Numerical approximation of partial differential equations. Vol. 23. Springer Science & Business Media, 2008.

[59] Allen, Myron B., and Eli L. Isaacson. Numerical analysis for applied science. Vol. 35. John Wiley & sons, 2011.

[60] Gustafsson, Bertil. Fundamentals of Scientific Computing. Vol. 8. Springer Science & Business Media, 2011.

[61] Butcher, John C. Numerical methods for ordinary differential equations in the 20th century. Journal of Computational and Applied Mathematics 125.1 (2000): 1-29.

[62] Reprinted from Journal of Computational Physics, 138, P. Koumoutsakos, Inviscid axisymmetrization of an elliptical vortex, 821–857, Copyright (1997), with permission from Elsevier.

[63] Reprinted from Journal of Computational Physics, 124.1, J. Strain, 2D vortex methods and singular quadrature rules, 131-145, Copyright (1996), with permission from Elsevier.

A Notable Code and Algorithms

A.1 Modified Two-part quadrature

It is applied to the integration of the product of M and $N = M - 1$ order functions, with M varying over a particular range. The interpolation of the product (IoP) and product of the interpolants (PoI) were numerically integrated by Matlab's built-in adaptive quadrature routine until machine precision was

Vectorized Lagrange evaluation, Lagrange derivatives, etc

A.2 Binary Singleton Expansion Matrix-Vector Products

A.3 Lagrange Bases and Constructed Quadrature

A.4 Velocity Evaluation

A.5 Semi-discrete System Construction

A.6 Low-storage Stability Optimized Runge-Kutta