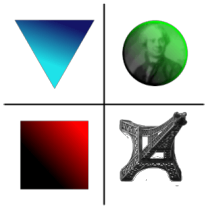
Higher-order adaptive methods for fluid dynamics

Rajarshi Roy Chowdhury



Sorbonne Université

École Doctorale Sciences Méchaniques, Acoustique, Électronique & Robotique

Institut Jean Le Rond d'Alembert (∂'Alembert UMR7190)

&

Laboratoire d'Informatique de Paris 6 (LIP6 UMR CNRS 7606)

Higher-order adaptive methods for fluid dynamics

By M. Rajarshi Roy Chowdhury

PhD thesis on Fluid Mechanics

Co-directed by M. Stéphane Popinet & M. Stef Graillat

Thesis Defense on 30 November, 2018

Before a jury composed of

Mme. Donna Calhoun	Boise State University, USA	Rapporteuse
M. Frédéric Golay	Université de Toulon, France	Rapporteur
M. Stéphane Vincent	Université Paris-Est-Marne-La-Vallée, France	Examinateur
M. Ivan Delbende	Sorbonne Université, France	Examinateur
M. Stef Graillat	Sorbonne Université, France	Co-directeur de These
M. Stéphane Popinet	Sorbonne Université, France	Co-directeur de These

Abstract

Higher-order adaptive methods for fluid dynamics

Fluids (gases and liquids) exist everywhere around us. Water covers 70% of the Earth's crust and gases like nitrogen and oxygen surround the planet. The field of fluid dynamics involves the study of liquids or gases in motion. The equations which govern the motion of fluids viz. the Navier–Stokes equations, are complex non-linear partial differential equations which do not have closed-form analytical solutions for most problems of practical interest. However, using numerical schemes, these partial differential equations of continuous variables can be transformed into huge algebraic systems of discrete variables and solved using high-performance computers.

A numerical method solved on a computing device will introduce errors in the final solution, will require a given amount of computational resource like memory and processor, and will take a finite amount of time to reach a solution. Thus the development of more accurate and faster algorithms to numerically model the equations of fluid dynamics is a constantly evolving research field. The present document is dedicated to both the study of existing lower-order numerical algorithms as well as either the implementation of existing or development and implementation of new higher-order algorithms, relevant for solving the incompressible Navier–Stokes equations.

The entire work has been carried out on the adaptive Cartesian solver for fluid equations Basilisk. We specifically research solvers for convection–diffusion, Poisson–Helmholtz equations, time-marching schemes, and for the shallow-water equations. We look at adaptive mesh methods for solving these equations and taking the Basilisk implementation of the adaptive wavelet algorithm on a quad-octrees as our starting point, we build a novel higher-order adaptive scheme. A recurring theme throughout this thesis is the comparison in accuracy and computing performance of different higher-order schemes when compared to their lower-order counterparts.

Méthodes adaptatives d'ordre élevé pour la dynamique des fluides

Les fluides (gaz et liquides) existent partout autour de nous. Alors que l'eau recouvre 70% de la croûte terrestre, des couches de gaz comme l'azote et l'oxygène entoure notre planète. Le domaine de la dynamique des fluides comprend l'étude des liquides ou des gaz en mouvement. Les équations qui régissent le mouvement des fluides à savoir les équations de Navier - Stokes sont des équations aux dérivées partielles non-linéaires complexes qui n'ont pas de solutions analytiques pour la plupart des problèmes d'intérêt pratique. Cependant, en utilisant des schémas numériques, ces équations aux dérivées partielles de variables continues peuvent être transformées en d'énormes systèmes algébriques de variables discrètes et résolues à l'aide d'ordinateurs à haute-performance.

Une méthode numérique résolue sur un dispositif informatique introduira des erreurs dans la solution finale, nécessitera une quantité donnée de ressources de calcul comme la mémoire et le processeur, et prendra une quantité finie de temps pour parvenir à une solution. Ainsi,

le développement d'algorithmes plus précis et plus rapides pour résoudre numériquement les équations d'un système de dynamique des fluides est un domaine de recherche en évolution constante. Le présent document est dédié à la fois à l'étude des algorithmes numériques d'ordre peu élevé, ainsi qu'à la mise en œuvre de méthodes existantes ou le développement et la mise en œuvre de nouvelles méthodes d'ordre supérieur, pertinentes pour la résolution des équations de Navier–Stokes incompressibles.

L'ensemble du travail a été effectué sur le solveur adaptatif Cartésien d'équations fluides Basilisk. Nous recherchons en particulier des solveurs pour la convection–diffusion, les équations de Poisson–Helmholtz, les schémas temporels et les équations de Saint-Venant. Nous examinons des méthodes de maillage adaptatif pour résoudre ces équations et prenons l'implémentation de Basilisk de l'algorithme adaptatif en ondelettes sur quad-octree comme point de départ pour construire un nouveau schéma adaptatif d'ordre supérieur. Un thème récurrent tout au long de cette thèse est la comparaison de la précision et des performances informatiques de différents schémas d'ordre supérieur par rapport à leurs homologues d'ordre inférieur.

Acknowledgements

This work was made possible with the financial support of Institut des Sciences du Calcul et des Données, Sorbonne Université. I am especially indebted to Prof. Pascal Frey, director ISCD, for supporting my career goals.

I would like to express my sincere gratitude to my thesis advisors Dr. Stéphane Popinet & Prof. Stef Graillat for their continuous support during my PhD. Dr. Popinet has been a source of immense encouragement and his insightful comments and difficult questions have always incentivized me to push the boundaries of my research. Without his continuous & diligent support, this thesis would not have taken the shape that it has.

Besides my advisors, I would like to thank Prof. Stéphane Zaleski for motivating me towards taking up this PhD. His weekly lab group-meetings were a source of immense learning experience for me. I thank my colleagues and labmates for the numerous stimulating research discussions we had.

I would like to extend my sincere gratitude to Prof. Tapan Sengupta (IIT-Kanpur), for motivating me to build a career in the field of fluid dynamics.

Last but not the least, I would like to thank my family: my parents, my sister and my fiancée for supporting me throughout this thesis and being there for me.

Contents

Chapter 1

Introduction

Contents

In this chapter, we introduce the general context of this PhD thesis, and focus primarily on fluid dynamics numerical solvers and their computing performance. We also briefly introduce the fluid dynamics solver Basilisk.

1.1 Background

The field of fluid dynamics involves the study of flows of liquids and gases. A typical solution to a fluid dynamics problem comprises of calculating the various properties of the flow eg. the flow velocity, the fluid pressure, density and temperature, as functions of space and time. Initial research in fluid dynamics began with designing simple experiments to take measurements of various flow properties with probes. Subsequently those measurements were used to formulate empirical or semi-empirical laws governing fluid flow which could be used for solving more complex practical problems.

The foundation of modern fluid dynamics is built over conservation laws, specifically conservation of mass and conservation of momentum. The addition of a continuum assumption and of a thermodynamic equation of state for the fluid, leads to the derivation of the Navier–Stokes equations which describe a large class of fluid dynamics problems. The governing equations of fluid dynamics are expressed mathematically through eq. 1.1 & 1.2.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{1.1}$$

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = -\nabla p + \nabla \cdot \sigma + \rho \mathbf{g} \tag{1.2}$$

where ρ is the fluid density, σ is the deviatoric stress tensor, which can be expressed using eq. 1.3, and where λ is the coefficient of bulk viscosity and μ is the coefficient of dynamic viscosity.

$$\sigma = \lambda (\nabla \cdot \mathbf{u}) \mathbf{I} + \mu (\nabla \mathbf{u} + \nabla \mathbf{u}^{\mathbf{T}}) \tag{1.3}$$

These equations are complex non-linear partial differential equations, which, barring very few notable exceptions, do not have a mathematical closed form solution even for the simplest of physical flows. In the pre-computation era, scientists and engineers unable to solve the full NS equations, were therefore reliant on applying empirical/semi-empirical laws derived from experimental measurements. However, with the advent of computation, and the ever-increasing performance of modern day supercomputers, the research field of computational fluid dynamics (viz. application of computer algorithms to numerically solve NS equations) has seen a massive growth.

(a) **(b)**

(c) **(d)**

Figure 1.1: *Applications of computational fluid dynamics:*
(a) An F/A 18C Hornet flying over the Pacific ocean, by Jonathan Chandler,
(b) A gas pipe in the dry region of Antofagasta, Chile. by Diego Delso,
(c) Surf on a rocky topography. Porto Covo, Portugal, by Joaquim Alves Gaspar &
(d) Most detailed true-color image of the entire Earth to date, using a collection of satellite-based observations, by Reto Stöckli.

A solver for the Navier–Stokes equations has a wide range of applications. For instance, it may be used to calculate the forces and moments acting on a fighter-aircraft (fig 1.1a), or to calculate the volume flow rate of gas through a gas-pipeline (fig 1.1b), or to study the dynamics of ocean waves as they break over a topographical land feature (fig. 1.1c) - which can have applications in disaster management studies (e.g. to predict coastal inland damage that can be caused by a tsunami), or to undertake meterological studies to predict future weather conditions using current atmospheric conditions, measured through satellites (fig 1.1d). There are numerous other applications which could range from studying interstellar phenomenon like nebulae to designing computer game algorithms to simulate maybe an explosion or the flow of smoke from a burning house etc.

1.2 Computational fluid dynamics

Computational fluid dynamics is a branch of fluid dynamics that uses numerical analysis and algorithms to solve and analyze problems that involve fluid flows. Computer algorithms are used to perform the calculations required to simulate the interaction of gases and liquids with surfaces defined by boundary conditions. With high-speed supercomputers, better numerical solutions

can be achieved, which improve on both the accuracy as well as the computing speed of fluid simulations.

Any kind of computational method involves the following basic operations:

- The geometry (physical bounds) of the problem is defined.
- The volume occupied by the fluids is divided into discrete finite volumes (the mesh). The mesh may be uniform or non-uniform or adaptive (changes as the solution progresses).
- The physical modeling is defined – for example, the equations of motion, the enthalpy, thermodynamic equation of state, species conservation etc.
- Numerical algorithms are chosen to solve the equations defining the fluid system. The boundary conditions are defined. This involves specifying the fluid behavior and properties at the domain boundaries. For transient problems, the initial conditions are also defined.
- The simulation is started and the equations are solved iteratively till a steady state is reached or solved with time marching to compute a transient unsteady solution.
- Finally a post-processor is used for the analysis and visualization of the resulting solution. The numerical results are matched with existing experimental measurements to verify the accuracy of the numerical algorithm.

The development of more accurate and faster algorithms to numerically model the equations of a fluid dynamic system is a constantly evolving research field. Different researchers, in the past, have developed and worked on different numerical schemes for converting these non-linear partial differential equations of fluid motion into computationally solvable simultaneous algebraic equations. These methods include finite volume methods, finite element methods, finite difference methods, spectral element methods, boundary element methods and high-resolution compact discretization schemes. The methodology of finite volume discretization schemes, has been chosen for this work. The reasons for this choice will be explained in Chapter 2. In the following section the well-known Chorin's algorithm will be used as a benchmark to demonstrate the process of converting a partial differential equation to an algebraic set of computationally solvable equations, which is the basic start of any computational method.

1.3 Chorin's projection method

In this section, we will look at the work of Chorin (1967) (similar to the work of Temam (1969)) in more detail to illustrate the development of a numerical scheme. Following the equations of continuity and momentum conservation (given by eq 1.1, 1.2 & 1.3), the Navier–Stokes equations for a single phase incompressible flow can be written down in the form of the eq. 1.4 & 1.5

$$\nabla \cdot \mathbf{u} = 0 \tag{1.4}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \vartheta \nabla^2 \mathbf{u} \tag{1.5}$$

From the eq. 1.5, it is evident that the velocity field and the pressure field are numerically coupled. Alexander Chorin in his 1967 paper, introduced the projection algorithm, which has since become an effective means for solving the incompressible Navier–Stokes equations. The key advantage of this method is that the computations of the velocity and pressure are de-coupled by using the Helmholtz decomposition which is used to split any vector field into a solenoidal component and an irrotational component.

The algorithm consists of two stages. In the first stage, an intermediate velocity that does not satisfy the incompressibility constraint is computed at each time step. In the second step, the pressure field is used to project the intermediate velocity onto a space of divergence-free velocity field to get the next update of velocity and pressure. The procedure is illustrated below

mathematically. We start with the momentum equation and break it into two parts to compute an intermediate velocity.

$$\frac{\mathbf{u}^* - \mathbf{u}^\mathbf{n}}{\Delta t} = -(\mathbf{u}^\mathbf{n} \cdot \nabla)\mathbf{u}^\mathbf{n} + \vartheta \nabla^2 \mathbf{u}^\mathbf{n} \tag{1.6}$$

The pressure gradient term has been dropped in the computation of the intermediate velocity u^* (eq. 1.6). All other velocities used are from the previous timestep n. The next step is to compute the final velocity at u^{n+1} using the pressure gradient term, using eq. 1.7.

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \frac{\Delta t}{\rho} \nabla p^{n+1} \tag{1.7}$$

$$\nabla^2 p^{n+1} = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{u}^* \tag{1.8}$$

$$\nabla \cdot \mathbf{u}^{n+1} = 0 \tag{1.9}$$

However the pressure gradient term in eq. 1.7 is at $n+1$ timestep. This can be solved for by taking the divergence of the eq. 1.7 and imposing the continuity condition on the velocity $\mathbf{u}^{n+1}$ (eq. 1.9). This gives a Poisson equation in pressure (eq 1.8). Thus, the pressure field p^{n+1} can be computed by solving the poisson equation and the computed pressure can be back-substituted in eq. 1.7 to get the new velocity field $\mathbf{u}^{n+1}$.

1.4 Error of a numerical scheme

Since we are working with numerical schemes, which interpret and subsequently approximate partial differential expressions, into algebraic expressions, it is obvious that the final solution that we can obtain by solving the numerical system will have certain deviations from the actual physical solution. So, for instance if we are looking for a solution of a velocity field in a fluid dynamics system, then while the actual solution can be given by $\mathbf{u}_\mathbf{x}^\mathbf{physical}$, the numerical solution will be given by $\mathbf{u}_\mathbf{x}^\mathbf{numeric}$. Thus we are left with different forms of error norms defined by eqs. 1.10.

$$\begin{aligned}
Error|_{L_\infty} &= MAX\left(\left|\mathbf{u}(x_i)^{physical} - \mathbf{u}(x_i)^{numeric}\right|\right) \\
Error|_{L_1} &= \frac{1}{N}\sum_{i=1}^{N}\left|\mathbf{u}(x_i)^{physical} - \mathbf{u}(x_i)^{numeric}\right| \\
Error|_{L_2} &= \frac{1}{N^2}\sum_{i=1}^{N}\left(\left|\mathbf{u}(x_i)^{physical} - \mathbf{u}(x_i)^{numeric}\right|\right)^2
\end{aligned} \tag{1.10}$$

The error (e) on the solution is a function of two parameters, namely the grid spacing (h) and the order of the scheme (k). What the eventual order of a numeric scheme will be, depends on the way the scheme has been derived. The relationship between the two is given by eq 1.11.

$$Error \propto h^k \tag{1.11}$$

The goal of this thesis is to (explore existing / develop new) higher-order ($\mathcal{O} > 2$) finite-volume numerical schemes, which will then be used to solve a range of fluid dynamic equations, and the solutions so obtained will be analyzed by comparing their accuracy and computational cost to the solutions obtained by solving the same system using an existing, widely used lower-order ($\mathcal{O}(2)$) numerical scheme. Both higher- and lower-order numerical schemes have their own set of advantages and disadvantages.

Advantages of a lower-order scheme

1. A lower-order scheme has compact stencils and is easier to code. This will be explained in the following sections.
2. A lower-order scheme is more robust compared to its higher-order counterpart.
3. A lower-order scheme is inexpensive to develop, hence has wider industrial usage compared to its higher-order counterpart.

Advantages of a higher-order scheme

1. **Higher accuracy for problems with a smooth solution**
   - Vortex dominated flows (We face higher numerical dissipation problems with lower order methods).
   - Boundary layer flows.
2. **Computation cost**
   - For reaching the same error levels, a lower grid spacing is required for higher-order methods.
   - However, as the complexity of the algorithm increases, higher CPU time is now required for computing each step viz-a-viz the lower-order scheme.
   - Therefore, we have a tradeoff. To determine with precision as to which of the two phenomena dominates, thus making the higher-order scheme cheaper or expensive, computationally speaking, is something that needs to be analyzed numerically. This will be a recurring theme in the thesis as we develop newer higher-order schemes.

Having given a brief theoretical introduction on the nature of the research that was undertaken for this PhD, I will now introduce the Cartesian PDE solver called Basilisk, and talk about its components while provide further context to my thesis.

1.5 Basilisk

Basilisk is the name of an open source program used for solving partial differential equations on adaptive Cartesian meshes. It is destined to be the successor of Gerris and is developed by the same authors working mostly at Institut Jean le Rond d'Alembert. All research undertaken through the course of this PhD has been carried out using Basilisk.

1.6 Grids

Any computational scheme starts with converting continuous fluid fields such as (pressure, velocity etc.) into sets of discrete variables, which give the values of the given fields at discrete locations in the fluid system. This is done to convert a partial differential equation into a solvable set of simultaneous algebraic equations. This step is accomplished by partition the fluid domain into distinct discrete finite volume cells. Such a partitioned fluid domain is referred to as grid/mesh in CFD terminology. There are different methods to partition the domain, with each method giving rise to different grid structures. Since we are working in Basilisk we will be restricting our discussion to Cartesian grids. The different kinds of grids that are available in Basilisk are as follows -

1. **Uniform Grid** - The grid spacing is uniform throughout the domain or the grid level is constant throughout. (Schematic fig. : 1.2a)
2. **Non-Uniform Grid** - The grid spacing is not the same throughout the domain, which means there can be patches of higher or lower grid refinement. (Schematic fig. : 1.2b)

3. **Adaptive Grid** - A large number of practical fluid flows exhibit a separation of spatial scales of behavior, meaning, there could be regions of the flow, where the flow fields may vary hugely either spatially or temporally or both, and there may be other regions of the same flow which may be relatively smooth or un-interesting. Such behaviors can be observed in compressible flows with shocks, flows with vorticity generating boundary layers, intermittent turbulent flows etc. This has led researchers to implement adaptive grids which follow the evolution of flow structures. In this context, two main approaches have been developed : (a) The AMR approach by Berger and Colella (1989) & (b) The Quad/Octree approach by Coirier (1994), Howell and Bell (1997) & Khokhlov (1998). While the AMR framework uses standard discretization algorithms on a hierarchy of regular Cartesian grids which interact with each other through boundary conditions, the Quad/Octree schemes, adapt the discretization operators in a way that they may be usable at coarse/fine cell boundaries.

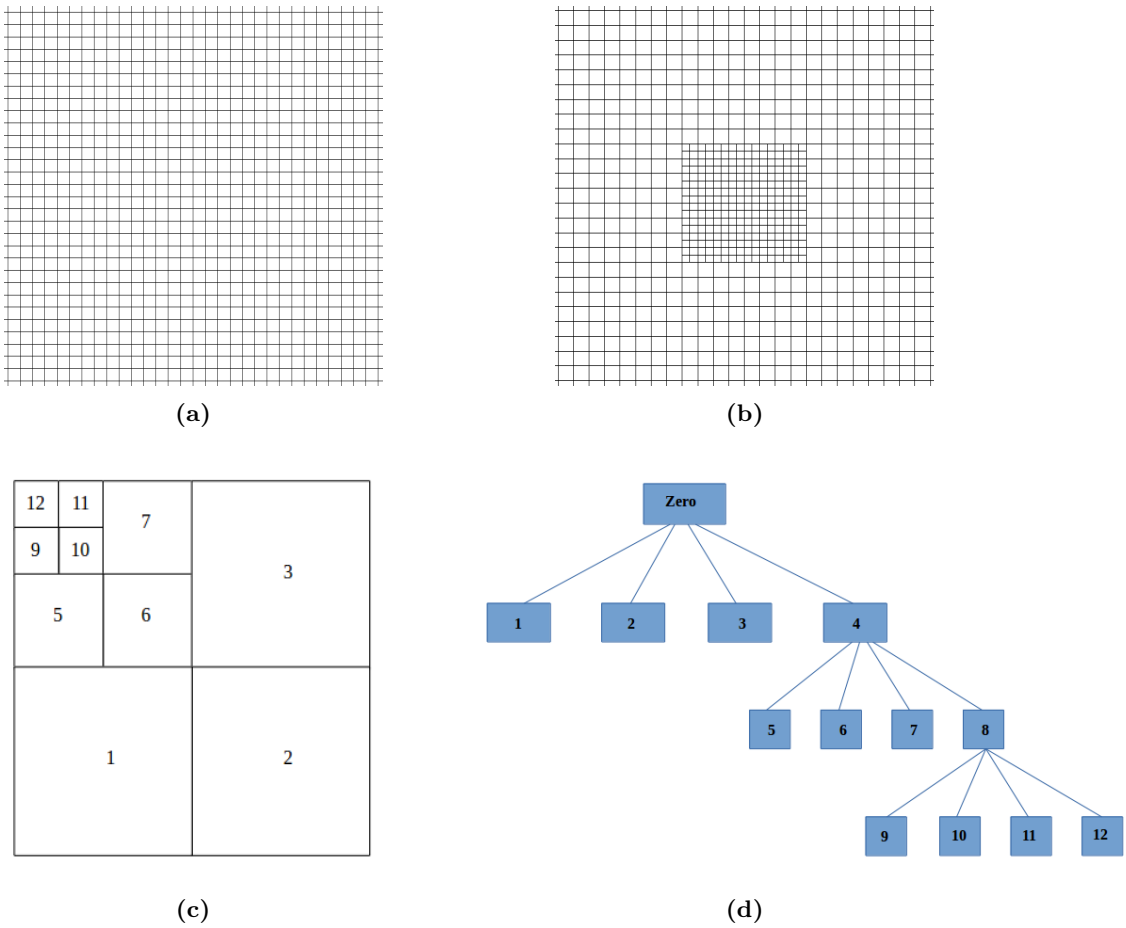


Figure 1.2: *(a) A sample uniform grid, (b) A sample non-uniform grid, (c) A sample non-uniform grid with numbered finite volumes, (d) Equivalent Quadtree representation of the non-uniform grid shown in 1.2(c).*

Basilisk identifies each finite-volume cell with a variable called the grid-level, which is related to the size of the cell Δ, by the formulation : $(\Delta = 1/2^l)$. Basilisk has a multigrid implementation option for uniform grids which is used, when solving problems which have multiple scales of behavior. A multigrid implementation involves a hierarchial arrangement of successively refined grids, starting from the coarsest level (level = 0, just 1 cell) up to the fine cell resolution N (which has 2^N cells along each direction). For the case of non-uniform or adaptive grids, Basilisk uses a tree implementation. viz. - bitrees for 1D, quadtrees for 2D and octrees for 3D implementation. In fact, Gerris, the predecessor of Basilisk, pioneered the usage of quadtree-adaptive implementation for solving incompressible fluid dynamic problems (Popinet (2003)). Figure 1.2c shows a schematic of a demo 2D non-uniform grid and fig. 1.2d shows how the same grid is stored in a quadtree representation, with each parent cell having four children. The terminal cells with no further refinement are called leaf cells.

1.6.1 Basilisk adaptivity

While solving a fluid dynamics problem on an adaptive mesh, it is usually a common strategy to first identify the prominent physical behavior of the fluid system, and then adjust our mesh adaptation implementing parameters to track that particular behavior, and adapt the mesh accordingly. However, given that Basilisk is built with a goal of being a generic PDE solver, the implementation of a problem-specific strategy to implement adaptivity was not pursued.

Instead a generic numerical wavelet-based adaptive algorithm was used to come up with a suitable generic adaptation criterion. What this algorithm essentially does is that it uses wavelet transform algorithms to make an estimation of spatial discretization errors on flow field, and the subsequent refinement or coarsening action is based solely on these error estimates. Basilik works with tree-based grids which are convenient for refinement and coarsening actions. A more detailed section on wavelets, multi-resolution analysis and adaptive mesh refinement (detailing the mathematics of restriction and prolongation functions) will be presented in chapter 2.

1.7 Fields

In Basilisk, the field constructs are used to store variables discretised spatially. They can be seen as a generalisation of C arrays. There are three types of fields in Basilisk, which are known as scalars, vectors and tensors. A scalar field, as the name suggests is used to store scalars. Vector fields are a collection of D scalar fields (where D is the dimension of the spatial discretisation) and tensor fields are a collection of D vector fields. The Basilisk declaration informations can be found through the following links - *scalar fields*, *vector & tensor fields*.

1.8 Boundary Conditions

Boundary conditions are an integral part of any CFD system, which are basically a set of additional constraints, to be implemented at the fluid boundaries. There are two kinds of boundaries viz. Domain boundaries and Refinement boundaries. Domain boundary conditions can be classified into Dirichlet, Neumann, symmetry or periodic boundary conditions. Basilisk implements these boundary conditions by using ghost cells. Ghost cells are one/two additional layers of cells which lie immediately outside the domain boundaries, and whose values are filled in such a way so that the exact boundary constraint can be applied at the domain boundary line. In the fig. 1.3 the cells marked in red are the ghost cells. Here we have two layers of ghost cells, which is required for higher-order schemes. However for $\mathcal{O}(2)$ schemes we only require one layer of ghost cells.

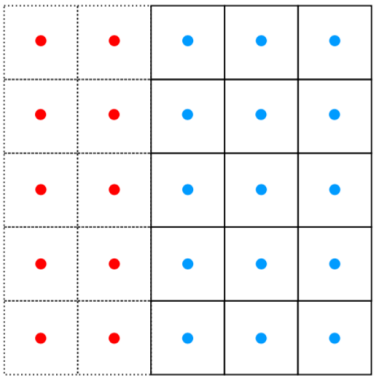


Figure 1.3: *Ghost cells outside domain boundaries.*

The refinement boundaries are the regions across which the grid changes its level of refinement. The implementation details of a few different kinds of boundary conditions are presented next.

Periodic Boundary Conditions: As the name suggests, periodic boundary conditions need to be implemented in such a fashion that the domain periodicity is retained. The fig. 1.4 elaborates this idea of filling ghost cell values.

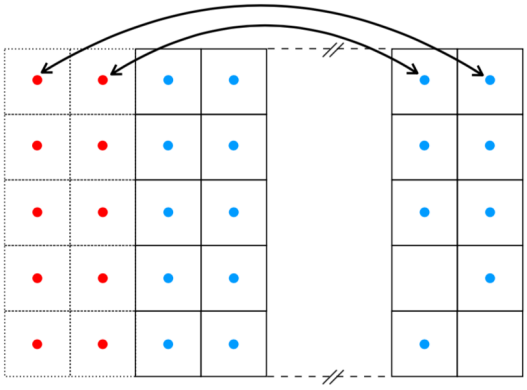


Figure 1.4: *Filling ghost cells to impose periodic boundary conditions.*

Symmetry Boundary Conditions: The symmetry boundary conditions need to ensure a zero derivative of the field at the boundary line. And hence, the ghost cell values are filled in while maintaining symmetry across the boundary line as shown in fig. 1.5.

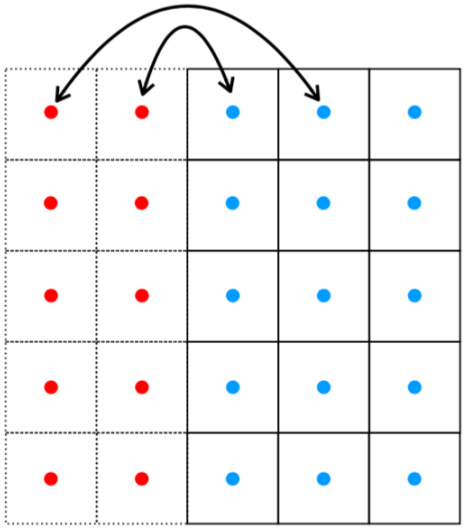


Figure 1.5: *Filling ghost cells to impose symmetry boundary conditions.*

Refinement Boundary Conditions: Boundary conditions need to be implemented in regions where the grid spacing changes. Take a look at fig. 1.6. The refinement boundary marks the region where the coarse and the fine cells come in contact. These grids are at different levels. So, for the simulation the appropriate boundary values need to be put in place for both the fine grids and the coarse grids. The implementation of boundary conditions on the fine level mesh is shown in fig. 1.7. The cell center values are simply obtained using the prolongation operator (to be described in chapter 2). Similarly, while computing the ghost values on the coarser level, the restriction operator is used.

While computing the appropriate boundary fluxes at refinement boundaries, there can be a possible inconsistency. In fig. 1.6, all the face-fluxes which are marked with black arrows can be computed using the normal flux formulations. However the flux value, that is marked in purple, has to be consistent with the sum of the flux values marked in red on the finer cell implementation, in fig. 1.7. This consistency is important to satisfy the conservation principle. Basilisk, maintains this consistency by computing the flux-values on the fine-cell using the formulations of the

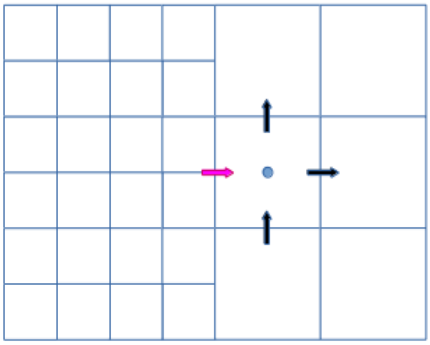


Figure 1.6: *The grid close to the refinement boundary.*

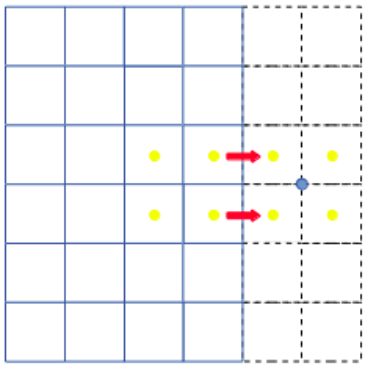


Figure 1.7: *The fine level mesh with refinement boundary ghost cells.*

numerical scheme being used, and these fine-cell fluxes are then added up and allocated for the flux at the coarser level implementation.

1.9 State of the art for Basilisk at the start of my PhD

Although Basilisk has a wide collection of partial differential equation solvers which are widely used in the study of fluid dynamics, my thesis will be focused on developing and analyzing solvers, which are used for computing single-phase Navier–Stokes equations and the Saint-Venant equations. The PDE solvers which are specifically used while solving these two system of equations and which were available in Basilisk at the start of my PhD were:

1. *Poisson–Helmholtz equation solver - $\mathcal{O}(2)$*
2. *Advection scheme - $\mathcal{O}(2)$*
3. *Viscosity solver (Implicit and Explicit) - $\mathcal{O}(2)$*
4. *Incompressible Navier–Stokes solver - centered scheme - $\mathcal{O}(2)$*
5. *Explicit solver for Saint-Venant equations - $\mathcal{O}(2)$*
6. *Predictor Corrector time marching scheme - $\mathcal{O}(2)$*
7. *Mesh adaptivity using $\mathcal{O}(2)$ wavelets*

1.10 Outline of this PhD Thesis

The research undertaken throughout the course of my PhD was aimed at building higher-order numerical schemes for the above mentioned solvers as well as demonstrating the comparative performance of the newly developed higher-order solvers with respect to the existing solvers. To this end, new solvers of either $\mathcal{O}(4)$ or $\mathcal{O}(5)$ have been developed for these systems of equations, and their performance study has been accomplished.

The higher-order implementations of the advection solver and a novel adaptive wavelet algorithm are the subject matter of Chapter 2 of this thesis, a new $\mathcal{O}(4)$ Poisson–Helmholtz solver is discussed in Chapter 3, an $\mathcal{O}(4)$ semi-implicit viscosity solver & the new higher-order centered incompressible Navier–Stokes solver are detailed out in Chapter 4 and finally the explicit Saint-Venant solvers, along with the predictor-corrector algorithm for time marching form the contents of Chapter 5 of this thesis.

Chapter 2

Advection Solver

Contents

In this chapter, we will focus on the solution of hyperbolic advection problems in fluid dynamics. Specifically, we will be talking about the existing Basilisk implementation of the $\mathcal{O}(2)$ Bell, Collella & Glaz advection scheme, the theory and proposed implementations of a $\mathcal{O}(5)$ WENO advection scheme along with Runge–Kutta time marching schemes. We will also explain the mathematics behind Basilisk implementation of adaptivity, and introduce a novel algorithm for implementing a higher-order adaptive method. A recurring theme throughout this chapter will be to compare and contrast the convergence and computing performance of the existing Basilisk $\mathcal{O}(2)$ schemes, with the proposed higher-order schemes.

2.1 Context

The idea behind this chapter is to efficiently implement higher-order schemes for solving hyperbolic advection equations of the form given by equation 2.1, where $\mathbf{u}$ is the vector of the conserved variable and F($\mathbf{u}$) is the vector of the fluxes. We have the initial condition on u at $t = 0$ and boundary condition (U_{LB} and U_{RB}) at all times.

$$\begin{aligned} &\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot F(\mathbf{u}) = 0 \\ &\mathbf{u}(x,0) = u_0(x) \\ &\mathbf{u}(0,t) = U_{LB}(t) \\ &\mathbf{u}(1,t) = U_{RB}(t) \end{aligned} \tag{2.1}$$

It should be pointed out that the equations have been written in their conservative forms, and it is particularly important that they be solved using conservative numerical methods, especially when dealing with problems which have discontinuities in their solutions. In such cases, the usage of a non-conservative method may give a solution which may seem to be reasonable at first but will end up being inaccurate. A well-known illustration is the solution of the momentum equation of an isothermal gas, by ignoring the pressure gradient term viz. Burger's equation, whose non-conservative numerical formulation, in the presence of an initial discontinuity fails in estimating the shock speed correctly LeVeque (1992). Later Hou and LeFloch (1994) conclusively proved that non-conservative schemes do not converge to the correct solution in the presence of a shock wave. On the other hand Lax and Wendroff (1960) proved that conservative numerical schemes do converge to a weak solution of the problem. It should be pointed out that for a problem which has only smooth solutions, both conservative and non-conservative methods are equivalent. Therefore, to broaden the scope of our solver we will be chosing a conservative numerical scheme. This is the rationale for applying finite volume methods to the advection problem, as their very formulation as presented in the next section, ensures conservation at the discrete level.

2.2 Godunov's scheme and the Riemann problem

The integral form of equation 2.1 can be written by first integrating it over space, and applying the divergence theorem to the flux term, and subsequently integrating it over time between t^n and t^{n+1} to obtain equation 2.2, which can be simplified into equation 2.3

$$\int_{x_{i-1/2}}^{x_{i+1/2}} \mathbf{u}(x,t^{n+1}) = \int_{x_{i-1/2}}^{x_{i+1/2}} \mathbf{u}(x,t^n) + \int_{t^n}^{t^{n+1}} F(\mathbf{u}(x_{i-1/2},t))dt - \int_{t^n}^{t^{n+1}} F(\mathbf{u}(x_{i+1/2},t))dt \tag{2.2}$$

$$\begin{aligned} &\rightarrow U_i^{n+1} = U_i^n + \frac{\Delta t}{\Delta x}\left(\mathcal{F}_{i-1/2} - \mathcal{F}_{i+1/2}\right) \\ &\text{where,} \quad U_i^n = \frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} U(x,t^n)dt \quad \text{and} \quad \mathcal{F}_{i-1/2} = \frac{1}{\Delta t}F(u(x_{i-1/2},t))dt \end{aligned} \tag{2.3}$$

Equation 2.3 is converted into a numerical scheme, by introducing approximations to the terms U_i^n and $\mathcal{F}_{i-1/2}$, based on appropriate interpretations of those terms. The first-order method introduced by Godunov (1959), assumes a piecewise constant distribution of the data over the spatial grid, i.e. by assuming the U_i's are constant in each cell, Godunov's method replaces the time integral of each flux with a forward Euler method which yields a fully discrete update formula as given by eq. 2.4. Since the quantity U_i jumps across adjacent numerical cells, we are left with a sequence of local Riemann problems.

$$\int_{t^n}^{t^{n+1}} F(u(x_{i-1/2},t)dt \approx \Delta t \mathcal{F}_{Riemann}(U_{i-1}^n, U_i^n) \tag{2.4}$$

Riemann problem: Godunov's method requires that we solve a Riemann problem at every cell boundary and at each timestep. The solution to a Riemann problem cannot be given in a closed form analytical solution, and hence approximate Riemann solvers are constructed to find the answers numerically. The approximate Riemann solvers can be divided into two categories, namely

- **Approximate State Riemann Solvers**: Here an approximation is calculated for the state variable $U(x_{i\pm1/2}, t)$, which is then used to calculate the flux.
- **Approximate Flux Riemann Solvers**: Here an approximation of the flux is calculated directly eg: HLLC, HLLE, Roe, Kurganov. See Godlewski and Raviart (2013) for a detailed description of different Riemann solvers.

While deciding on the timestep, the scheme must abide by the CFL condition, introduced in the work of Courant et al. (1967), and represented by eq. 2.5 where v^n_{max} denotes the maximum wave velocity present throughout the domain at time t^n.

$$\Delta t \leq \frac{\Delta x}{|v^n_{max}|} \tag{2.5}$$

The spatial accuracy of the Godunov method can be improved by adopting some kind of reconstruction procedure, whereby instead of using the left and right cell values by assuming piecewise constant formulation, we instead reconstruct the left and right face values on the faces where we are computing the fluxes and then plug these reconstructed values in the Riemann solver viz. $\mathcal{F}_{Riemann}(U^+_{i-1/2}, U^-_{i-1/2})$. The temporal accuracy of the solver can be improved by combining the above method with a Runge–Kutta scheme.

2.3 Temporal Scheme - Runge–Kutta schemes

While solving time marching problems in fluid mechanics, it is a common practice to first discretize the spatial variables to obtain a semi-discrete Method of lines scheme. The scheme can then be represented using eq. 2.6, which is an ODE in the time variable, and can subsequently be discretized using a known ODE solver.

$$\frac{du}{dt} = L(u) \tag{2.6}$$

In the past, explicit Runge–Kutta methods have been used by numerous researchers for solving similar time evolution equations to obtain higher-order temporal accuracy. Using classical numerical formulations, I implemented the *Runge–Kutta solvers (RK-2 and RK4)* in Basilisk. The classical formulations for the RK2 and the RK4 schemes are given by eqs 2.7 & 2.8 respectively.

$$\underline{\text{Runge–Kutta-2}}: \quad u_{n+1} = u_n + \Delta t \times L\Big(t_n + \frac{\Delta t}{2}, u_n + \frac{\Delta t}{2} L(t_n, u_n)\Big) \tag{2.7}$$

$$\begin{aligned}
\underline{\text{Runge–Kutta-4}}: \quad u_{n+1} &= u_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\
k_1 &= L(t_n, u_n) \\
k_2 &= L\Big(t_n + \frac{\Delta t}{2}, u_n + \frac{k_1 \Delta t}{2}\Big) \\
k_3 &= L\Big(t_n + \frac{\Delta t}{2}, u_n + \frac{k_2 \Delta t}{2}\Big) \\
k_4 &= L\Big(t_n + \Delta t, u_n + k_3 \Delta t\Big)
\end{aligned} \tag{2.8}$$

However, while adressing pertinent questions on the stability of the scheme, it should be noted that usually a linear stability analysis is adequate for problems which have smooth solutions, while for problems which may have sharp discontinuities, such as solutions to hyperbolic problems, a stronger measure of stability is usually required. This is particularly true for ENO/WENO based schemes for spatial discretization which use moving stencils as will be discussed in subsequent sections. On the other hand, linear stability analysis is based on the premise that the stencils are fixed and errors accumulate in a predictable fashion. Hence, what is required is a stronger stability preserving condition.

For this reason I reviewed the available literature to understand the formulations of higher-order strong-stability-preserving time discretization schemes. Such time discretization schemes were first introduced as Total Variation Diminishing (TVD) schemes in the works of Shu (1988) & Shu and Osher (1989) and further developed in the work of Gottlieb and Shu (1998).

Given a method of lines problem as shown in eq. 2.6, it is assumed that the spatial discretization has a unique property that when it is combined with a first-order forward Euler time discretization scheme, given by eq. 2.9, while maintaining the CFL condition 2.10 on the timestep, the total variation of the solution u^n must follow the TVD property as shown in eq. 2.11.

$$u^{n+1} = u^n + \Delta t L(u^n) \tag{2.9}$$

$$\Delta t < \Delta t_{FE} \tag{2.10}$$

$$\begin{aligned} TV(u^{n+1}) < TV(u^n), \quad \text{where} \\ TV(u^n) = \sum_j |u^n_{j+1} - u^n_j| \end{aligned} \tag{2.11}$$

The aim of the higher-order multistep SSP Runge–Kutta scheme is to obtain higher-order time accurate solutions, while still retaining this strong stability property of Euler discretizations, maybe with a modified CFL condition, given by eq. 2.12, where c is the CFL coefficient.

$$\Delta t \leq c \Delta t_{FE} \tag{2.12}$$

I will be working with the SSP-RK3 scheme developed by Shu and Osher (1989), and which was used by Liu et al. (1994) in their pioneering work on WENO schemes. The formulation of the SSP–RK3 scheme is given in eq. 2.13.

$$\begin{aligned} \underline{\text{SSP-RK3}}: \quad & y^{(1)} = y^n + \Delta t L(u^n) \\ & y^{(2)} = \frac{3}{4} y^n + \frac{1}{4} y^{(1)} + \frac{1}{4} \Delta t L(u^{(1)}) \\ & y^{n+1} = \frac{1}{3} y^n + \frac{2}{3} y^{(2)} + \frac{2}{3} \Delta t L(u^{(2)}) \\ & \Delta t \leq c \Delta t_{FE} \quad \text{where,} \quad c = 1 \end{aligned} \tag{2.13}$$

2.4 Basilisk - State of the art: October 2015

When I started my PhD on 01/10/2015, the Basilisk tracer advection solver was based on the work of Bell, Colella and Glaz (BCG) (Bell et al. (1989)) and it still remains the work-horse solver for Basilisk. The BCG solver is a full Navier–Stokes solver, with formulations for the convective terms, diffusive terms and distinct projection methods with time marching schemes. Given the context of this chapter I will be focussing on the convective term discretization with temporal scheme of the BCG solver in the next section, while the rest of the solver will be discussed in detail in chapter 4 on Navier–Stokes equations.

Basilisk is a powerful adaptive cartesian grid solver for PDEs, and the way adaptivity is implemented in Basilisk is through the use of wavelet function. This method will be discussed in detail in section 2.10 in this chapter. The error estimations and the general order of the numerical method, which was considered state of the art for Basilisk at the time of the start of my PhD was a $\mathcal{O}(2)$ method based on bilinear prolongation functions.

We intend to build higher-order methods for advection on Basilisk. For this we will be looking at WENO schemes for advection, implement the higher-order temporal schemes as introduced in section 2.3 and build a higher-order method to implement adaptivity.

2.5 Advection scheme by Bell, Colella and Glaz

Bell et al. (1989) introduced a second-order projection method for Navier Stokes equations in their 1988 paper, which improved upon the original projection method of Chorin (1967). The BCG algorithm, similar to Chorin's method, first solves the diffusion-convection equation to predict an intermediate velocity field, which is subsequently projected on a space of divergence-free vector fields. Where the two methods differ, is that in the BCG method the diffusion-convection step and the projection step are coupled in a unique way to achieve a second-order temporal discretization which is not the case for Chorin's original algorithm, which has a first-order temporal accuracy.

2.5.1 Convection term

The convection term is mathematically expressed as $\left[(\mathbf{U}\cdot\nabla)\mathbf{U}\right]^{n+1/2}$ and its formulation is based on the work of Colella (1990) and Vanleer (1983). The method involved here is different from the conventional upwind differencing methods, in that, these unsplit second-order Godunov methods couple the spatial and the temporal discretization by making sure that the information is propagated only along the characteristics, leading to robust schemes with higher-order discretizations and improved phase error. There are four steps in the process, namely Reconstruction, characteristic extrapolation, Riemann problem and flux computation.

- **Reconstruction**: This step involves the computation of limited slope profiles in each cell. The gradient is computed using a slope limiter method as described by equations 2.14 & 2.15. The limiting algorithm is constructed to prevent the centered slopes from introducing new maxima and minima in the velocity field. A new parameter, named θ determines the exact limiting scheme. If the θ value is equal to 1 we get the minmod limiter, which is the most dissipative scheme, while with a θ value of 2 we get the superbee limiter which is the least dissipative.

$$\begin{aligned} d_1 &= \theta \times (u_i - u_{i-1}) \\ d_2 &= \frac{u_{i+1} - u_{i-1}}{2} \\ d_3 &= \theta \times (u_{i+1} - u_i) \end{aligned} \tag{2.14}$$

$$(\Delta_x u) = \begin{cases} \begin{cases} min(d_1, d_3) & \text{if } d_2 \leq d_1 \\ 0 & \text{otherwise} \end{cases} & \text{if } u_{i-1} \leq u_i \leq u_{i+1} \\ \begin{cases} max(d_1, d_3) & \text{if } d_2 \geq d_1 \\ 0 & \text{otherwise} \end{cases} & \text{if } u_{i-1} \geq u_i \geq u_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

- **Characteristic extrapolation**: This step involves the characteristic extrapolation of cell-centered velocity field U^n to predict the value of U on the cell faces at half time steps $t^{n+1/2}$. Its a simple Taylor series expansion to second-order as shown in eq. 2.16.

$$
\begin{aligned}
U^{n+1/2,L}_{i+1/2,j} &= U^{n}_{i,j} + \frac{\Delta x}{2}(U_x)_{i,j} + \frac{\Delta t}{2}(U_t)_{i,j} \\
U^{n+1/2,T}_{i,j+1/2} &= U^{n}_{i,j} + \frac{\Delta y}{2}(U_y)_{i,j} + \frac{\Delta t}{2}(U_t)_{i,j}
\end{aligned}
\tag{2.16}
$$

In eq. 2.16, the expressions for $(U_x)_{i,j}$ and $(U_y)_{i,j}$ are computed using the limited slope formulations using equation 2.15 whereas the expression for U_t uses the differential equation to express the time derivatives in terms of spatial derivatives using eq. 2.17 . While evaluating derivatives in eq. 2.17, the derivatives normal to the cell edge are computed using limited values, while the transverse derivatives are computed using upwind formulations as in eq. 2.18

$$
U_t = \mathbf{P}\begin{pmatrix} \epsilon\nabla^2 u - uu_x - vu_y \\ \epsilon\nabla^2 v - uv_x - vv_y \end{pmatrix} \approx \begin{pmatrix} \epsilon\nabla^2 u - uu_x - vu_y \\ \epsilon\nabla^2 v - uv_x - vv_y \end{pmatrix} - \begin{pmatrix} p_x^{n-1/2} \\ p_y^{n-1/2} \end{pmatrix}
\tag{2.17}
$$

$$
(U_y)_{i,j} = \begin{cases} \frac{U_{i,j}-U_{i,j-1}}{\Delta y} + \frac{1}{2}\left(1 - \frac{\Delta t v_{i,j}}{\Delta y}\right) \times \frac{(\Delta_y U)_{i,j}-(\Delta_y U)_{i,j-1}}{\Delta y} & \text{if } v_{i,j} > 0 \\ \frac{U_{i,j+1}-U_{i,j}}{\Delta y} - \frac{1}{2}\left(1 + \frac{\Delta t v_{i,j}}{\Delta y}\right) \times \frac{(\Delta_y U)_{i,j+1}-(\Delta_y U)_{i,j}}{\Delta y} & \text{if } v_{i,j} < 0 \end{cases}
\tag{2.18}
$$

- **Riemann problem**: There are ambiguities in edge values as each face has two expansions, each derived from the characteristic expansion from each side of the interface. To resolve this ambiguity, a Riemann problem is solved by first dropping the diffusion, pressure gradients and the transverse flux terms. We are left with the equation system 2.19.

$$
\begin{aligned}
u_t + uu_x &= 0 \\
v_t + uv_x &= 0
\end{aligned}
\tag{2.19}
$$

On the $i + 1/2$ face, while working with the pair $\left(U^{n+1/2,L}_{i+1/2,j}, U^{n+1/2,R}_{i+1/2,j}\right)$, we notice that u satisfies the quasilinear form of Burger's equation, while v is passively advected by u, thus giving us the motivation of upwinding u based on eq. 2.20.

$$
u_{i+1/2,j} = \begin{cases} u^L & \text{, if } u^L \geq 0,\ u^L + u^R \geq 0 \\ 0 & \text{, if } u^L < 0,\ u^R > 0 \\ u^R & \text{, otherwise} \end{cases}
\tag{2.20}
$$

The upwind determination for v is given by eq. 2.21.

$$
v_{i+1/2,j} = \begin{cases} v^L & \text{, if } u_{i+1/2,j} > 0 \\ v^R & \text{, if } u_{i+1/2,j} < 0 \\ \frac{v^L+v^R}{2} & \text{, if } u_{i+1/2,j} = 0 \end{cases}
\tag{2.21}
$$

- **Flux Computation**: Analytically, for divergence-free vector fields, the relationship : $(U \cdot \nabla)U = \nabla \cdot (U \otimes U)$, holds true, however this relationship is not applicable at $t_{n+1/2}$ because of the effects of limiting and the approximate projection operator defined by using old values of pressure. Hence, the operator is built by listing all the terms in the conservative form and then subtracting the required terms to modify the expression to its convective form. The final formulation is given by eq. 2.22.

$$
\begin{aligned}
[(U \cdot \nabla)U]^{n+1/2} = uU_x + vU_y \approx \frac{1}{2}(u_{i+1/2,j} + u_{i-1/2,j})\frac{U_{i+1/2,j} - U_{i-1/2,j}}{\Delta x} \\
+ \frac{1}{2}(v_{i,j+1/2} + v_{i,j-1/2})\frac{U_{i,j+1/2} - U_{i,j-1/2}}{\Delta y}
\end{aligned}
\tag{2.22}
$$

Timestep Restriction: The Godunov method is an explicit time marching method and hence it requires to adhere to the CFL condition given by eq. 2.23.

$$max_{i,j}\left(\frac{u_{i,j}\Delta t}{\Delta x}, \frac{v_{i,j}\Delta t}{\Delta y}\right) \leq 1 \tag{2.23}$$

2.6 Weighted Essentially Non Oscillatory (WENO) scheme

WENO schemes are based on ENO (Essentially Non Oscillatory) schemes, which were first introduced by Harten et al. (1987). The key idea behind the ENO scheme is to use a higher-order flux reconstruction using a broad stencil in regions of smooth solutions while breaking down the higher-order stencil into multiple lower-order stencils, and then to use only the smoothest stencil among several candidate stencils to approximate the flux in regions of solution discontinuities. ENO schemes are uniformly higher-order accurate right up to the regions of solution discontinuities (shocks / hydraulic jumps) and are very robust to use. However there are some drawbacks as well. Firstly, wherever the solution or its derivative approaches a zero value, even round-off perturbations near that region can cause completely different choices of stencils. Secondly, ENO schemes require heavy usage of logical statements while choosing the appropriate stencil, which results in poor performance while implementation on vector supercomputers.

The WENO schemes of Liu et al. (1994) overcomes these drawbacks while maintaining the robustness of the ENO schemes. The methodology behind WENO differs from the methodology behind the ENO scheme in the way it approximates the numerical flux. While the ENO scheme uses only one of the candidate stencils in regions of solution discontinuities, the WENO scheme uses a convex combination of all the candidate stencil contributions. Each of the candidate stencil is assigned a weight based on a stencil smoothness indicator and this determines the effective contribution of the candidate stencil to the final flux approximation. The weights are assigned in such a way that in regions of smooth solutions the effective reconstruction is a higher-order one, while in regions near the discontinuities the stencils containing the discontinuity are assigned a nearly-zero weight. WENO schemes remove the dependence on the logical stencil choosing statements making them run twice as fast compared to ENO codes on vector supercomputers. Also, the WENO schemes are also insensitive to round-off errors, unlike the ENO schemes. The method of Liu et al. (1994) was a third-order accurate finite volume method. Later Jiang and Shu (1996) derived the framework for implementing an arbitrary order WENO scheme, whose fifth-order implementation remains the most popularly applied version of the WENO scheme. Next, based on the work of Jiang and Shu (1996), a step by step guide to implement the classical fifth-order WENO scheme is presented (which we will be using for our application). For simplicity, we will start with a one-dimensional case and subsequently extend the application to higher-dimension cases. We look at the initial value problem given by eq. 2.1, the semi-discrete form of which can be expressed using eq. 2.24.

$$\begin{aligned} U_i^{'}(t) &= -\frac{1}{\Delta x}\Big\{f(u(x_{i+1/2},t)) - f(u(x_{i-1/2},t))\Big\} \\ \text{Where, } U_i(t) &= \int_{x_{i-1/2}}^{x_{i+1/2}} u(x,t)dx \end{aligned} \tag{2.24}$$

We need to compute the numerical fluxes $F_{i\pm1/2}$, which are an approximation to the exact fluxes $f(u(x_{i\pm1/2},t))$ within an order of accuracy of $\mathcal{O}(\Delta x^{2r-1})$, where $r = 3$ for the fifth-order WENO scheme.

Using the three different stencils S1, S2 and S3, as shown in fig. 2.1 three different $\mathcal{O}(\Delta x^r)$ polynomial reconstructions for the fluxes at locations $x_{i\pm1/2}$ can be made, which can be represented by $P^r_{i,k}(x_{i\pm1/2})$. The idea of the WENO scheme is now to combine all the three-lower order reconstructions through weights w_k, which satisfy $\sum_{k=1}^{3} w_k = 1$, such that the conserved quantity u at at locations $x_{i\pm1/2}$ can be written using eq. 2.25

$$u_{i\pm1/2} = \sum_{k=1}^{3} w_k P^r_{i,k}(x_{i\pm1/2}) + \mathcal{O}(\Delta x^{2r-1}) \tag{2.25}$$

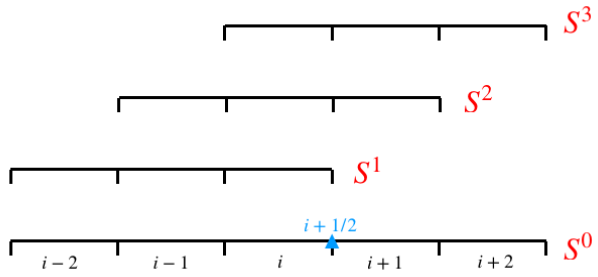


Figure 2.1: Illustrations of the stencils for $r = 3$.

If the solution is smooth on the large stencil, then a polynomial reconstruction of $\mathcal{O}(\Delta x^{2r-1})$ for flux at $x_{i-1/2}$ can be computed using all five stencil points, and similar polynomial reconstructions of $\mathcal{O}(\Delta x^r)$ can be computed for all three individual stencils. By matching the coefficients using eq. 2.26 one gets the values of weights for smooth solutions which are referred to as γ_k. To compute the respective WENO weights w_k from γ_k we use eq. 2.27.

$$\sum_{k=1}^{r} \gamma_k P^r_{i,k}(x_{i-1/2}) = P_i^{2r-1}(x_{i-1/2}) \tag{2.26}$$

$$\begin{aligned} w_k &= \frac{\tilde{w_k}}{\sum_{k=1}^{r} \tilde{w_k}} \\ \tilde{w_k} &= \frac{\gamma_k}{\{\epsilon + IS_k\}^p} \\ IS_k &= \sum_{l=1}^{r-1} \Delta x^{2l-1} \int \left\{ \frac{d^l}{dx^l} P^r_{i,k}(x) \right\}^2 \\ \epsilon &= 10^{-6} \end{aligned} \tag{2.27}$$

The functions IS_k is called the smoothness indicator, and is a measure of the smoothness of the interpolation polynomial P in that particular stencil, which is also a measure of the total variation of the interpolating polynomial. The construction of weights is done in a way such that the total variation for the approximation can be minimized, hence giving us a convex combination of individual stencil contributions. The detailed derivations can be looked up in the classical paper on WENO schemes by Liu et al. (1994).

2.6.1 Left and right side reconstruction

The stencil presented in fig 2.1 will be used to calculate the right side flux of u i.e. $f(u(x_{i-1/2}, t))^+$ at the location $x_{i-1/2}$ as there are three stencil values to the right and two to the left of the face $x_{i-1/2}$, thus the computation will introduce a rightward bias. To compute a left side flux of u i.e. $f(u(x_{i-1/2}, t))^-$ at the same location $x_{i-1/2}$, the stencil chosen should include the cells $i-3, i-2, i-1, i, i+1$ i.e. three to the left and two to the right of the face. Subsequently, the actual flux through the face can be computed using a numerical flux function such as the local Lax–Friedrichs flux, or solving a Riemann problem at each face. In simple incompressible flows the direction of upwinding will simply depend on the direction of the local velocity.

2.6.2 Fifth-order WENO – Formulations

The fifth-order WENO reconstruction for the left and the right side in a one-dimensional domain are expressed using the classical formulations given by eq. 2.28 & 2.29 respectively.

Left side reconstruction

$$
\begin{aligned}
u_{i-1/2}^{S1-} &= \frac{1}{3}U_{i-3} - \frac{7}{6}U_{i-2} + \frac{11}{6}U_{i-1} \\
IS_1 &= \frac{13}{12}\{U_{i-3} - 2U_{i-2} + U_{i-1}\}^2 + \frac{1}{4}\{U_{i-3} - 4U_{i-2} + 3U_{i-1}\}^2 \\
u_{i-1/2}^{S2-} &= -\frac{1}{6}U_{i-2} + \frac{5}{6}U_{i-1} + \frac{1}{3}U_i \\
IS_2 &= \frac{13}{12}\{U_{i-2} - 2U_{i-1} + U_i\}^2 + \frac{1}{4}\{U_{i-2} - U_i\}^2 \\
u_{i-1/2}^{S3-} &= \frac{1}{3}U_{i-1} + \frac{5}{6}U_i - \frac{1}{6}U_{i+1} \\
IS_3 &= \frac{13}{12}\{U_{i-1} - 2U_i + U_{i+1}\}^2 + \frac{1}{4}\{3U_{i-1} - 4U_i + U_{i+1}\}^2 \\
\{\gamma_1, \gamma_2, \gamma_3\} &= \left\{\frac{1}{10}, \frac{3}{5}, \frac{3}{10}\right\}
\end{aligned}
\tag{2.28}
$$

Right side reconstruction

$$
\begin{aligned}
u_{i-1/2}^{S1+} &= -\frac{1}{6}U_{i-2} + \frac{5}{6}U_{i-1} + \frac{1}{3}U_i \\
IS_1 &= \frac{13}{12}\{U_{i-2} - 2U_{i-1} + U_i\}^2 + \frac{1}{4}\{U_{i-2} - 4U_{i-1} + 3U_i\}^2 \\
u_{i-1/2}^{S2+} &= \frac{1}{3}U_{i-1} + \frac{5}{6}U_i - \frac{1}{6}U_{i+1} \\
IS_2 &= \frac{13}{12}\{U_{i-1} - 2U_i + U_{i+1}\}^2 + \frac{1}{4}\{U_{i-1} - U_{i+1}\}^2 \\
u_{i-1/2}^{S3+} &= \frac{11}{6}U_i - \frac{7}{6}U_{i+1} + \frac{1}{3}U_{i+2} \\
IS_3 &= \frac{13}{12}\{U_i - 2U_{i+1} + U_{i+2}\}^2 + \frac{1}{4}\{3U_i - 4U_{i+1} + U_{i+2}\}^2 \\
\{\gamma_1, \gamma_2, \gamma_3\} &= \left\{\frac{3}{10}, \frac{3}{5}, \frac{1}{10}\right\}
\end{aligned}
\tag{2.29}
$$

2.6.3 Implementing WENO-5 stencils on Basilisk

The complete stencil for a 1D - WENO-5 scheme includes data from seven points. As can be seen from the left and the right side formulations, the interpolation on the $x_{i-1/2}$ face requires volume averages from - $\{U_{i-3}, U_{i-2}, U_{i-1}, U_i, U_{1+1}, U_{1+2}\}$ cells, similarly the interpolation on the $x_{i+1/2}$ face requires volume averages from - $\{U_{i-2}, U_{i-1}, U_i, U_{i+1}, U_{1+2}, U_{1+3}\}$ cells. Hence, to discretize the convection term, we need access to seven volume averages or for three neighbors on each side for each cell. However, the way basilisk implements its stencils in the adaptive quadtree, a programmer gets access to only two neighbors on each side of a cell, or five cells in total. (More details here - Basilisk-Stencils).

To have a work-around for this problem, we use an indirect approach of computing a $\mathcal{O}(2)$ cell-centered gradient field, with the formulation given in eq. 2.30. After applying boundary-conditions on this gradient field we now have access to a 7-point stencil, where the third neighbor on each side can be referenced by using the eq. 2.31.

$$\nabla_x U_i = \frac{U_{i+1} - U_{i-1}}{2\Delta} \tag{2.30}$$

$$\begin{aligned} U_{i+3} &= U_{i+1} + 2\nabla_x U_{i+2} \\ U_{i-3} &= U_{i-1} - 2\nabla_x U_{i-2} \end{aligned} \tag{2.31}$$

2.6.4 Literature review

In this section, we provide further background to convection-dominated problems, and also talk about applications and further developments to the WENO schemes. Traditionally, WENO schemes have been applied to the study of convection dominated problems, which have solution discontinuities (shocks or contact discontinuities for high speed gas dynamics), or for even complex solutions like vortices or acoustic waves. Traditionally such problems were solved using the first-order scheme by Godunov (1959) or the scheme by Roe (1981), which resolve discontinuities monotonically and do not introduce spurious oscillations, however they are largely dissipative (even for the smooth parts of the solution) and hence complicated smooth structures like vortices require many grid points when using these schemes. This was followed by the development of schemes like Van Leer (1979), Harten (1983) & Colella and Woodward (1984), which were second-order in smooth regions and resolved discontinuities monotonically (with sharper transitions than first-order schemes). Then came the development of ENO schemes (Harten et al. (1987), Shu and Osher (1988) & Shu and Osher (1989)), for resolving problems containing both shocks as well as complicated smooth structures (eg: shock-vortex interactions). At this point, to improve upon the ENO schemes, the WENO scheme of Liu et al. (1994) & Jiang and Shu (1996) were introduced. WENO schemes have been proved to be convergent to an high-order of accuracy, for smooth solutions, however no such general proof of convergence/stability exists for discontinuous flows, though the scheme demonstrates these properties in applications.

Multi-dimensional implementation of WENO schemes for Cartesian grids is quite straightforward. This algorithm was derived in the work of Shi et al. (2002). The section 2.8 illustrates the scheme. The finite-volume schemes can also be discretized in time, giving rise to a class of Central-WENO schemes (Nessyahu and Tadmor (1990) & Levy et al. (1999)), which have the advantage that the numerical flux is computed on the smooth part of the reconstructed function, hence there is no need for the Riemann solver. These schemes use component wise WENO reconstruction to build third-order or fourth-order schemes, however for high-order central WENO schemes, a characteristic reconstruction is still necessary to obtain stable results (Qiu and Shu (2002)).

Competing with the WENO schemes, are the Discontinuous-Galerkin methods, introduced in the works of Cockburn and Shu (1998) & Cockburn and Shu (2001), as finite element methods but they can also be used as a generalized finite-volume method for finding solutions to conservation laws. While the WENO schemes, evolve only one piece of information in a cell (viz. the cell average) and all the interface values are re-constructed, the Discontinuous-Galerkin method evolves an entire polynomial (if the scheme order is K, then there are K+1 variables which are being fed into the time-marching scheme), thus increasing the storage and computational cost of the evolution. On the other hand, they cut down on the cost of reconstruction. We decided to use WENO schemes rather than Discontinous-Galerkin methods for this PhD.

WENO schemes also find application in designing hyperbolic conservation schemes with source terms (also referred to as balance laws - see Chapter 5). An added advantage of using a high-order schemes for such problems, is that they can resolve small scale perturbations of an equilibrium solution very accurately, without using an excessively refined grid. Such schemes for still water solutions of Saint–Venant equations can be found in (Vukovic and Sopta (2002), Xing and Shu (2005), Xing and Shu (2006) & Noelle et al. (2007)).

2.7 Test case – Advection in a 1D Domain

In this section, we take two different one-dimensional test cases to observe the performance of the developed WENO schemes in Basilisk. Both cases involve the passive advection of a tracer field under the influence of a given velocity field. The results and performance of the WENO methodolgy are compared and contrasted with the available second-order, Bell, Collella and Glaz scheme.

2.7.1 Passive advection of a 1D smooth tracer field

Our first test case involves the passive transport of a one dimensional sinusoidal field given by the initial condition $Tracer(x, 0) = sin(2\pi x)$ over a periodic domain $x \in [-0.5, 0.5]$, under the influence of a uniform velocity field given by $u_x = 0.1$. The time marching is carried out over one time period viz. $t \to 0$ to 10, and the advected solution is compared with the initial condition. The maximum errors are noted for a range of simulations carried out over various grid resolutions using both the BCG scheme and the new WENO scheme. The runs are repeated over two CFL numbers.

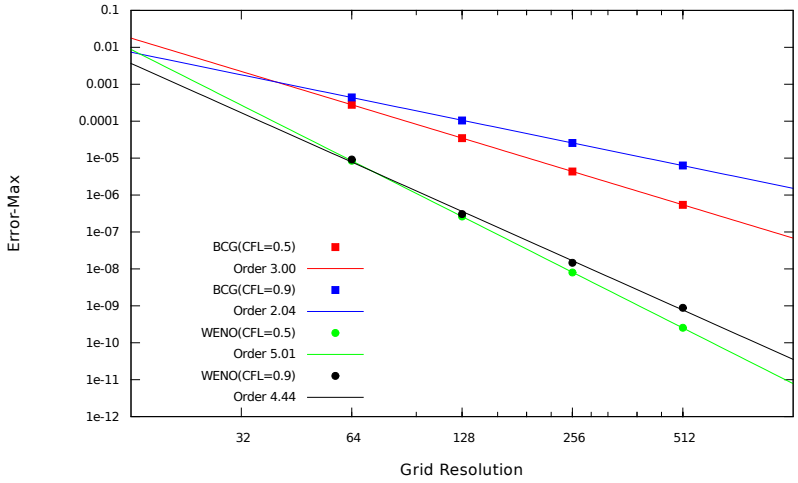
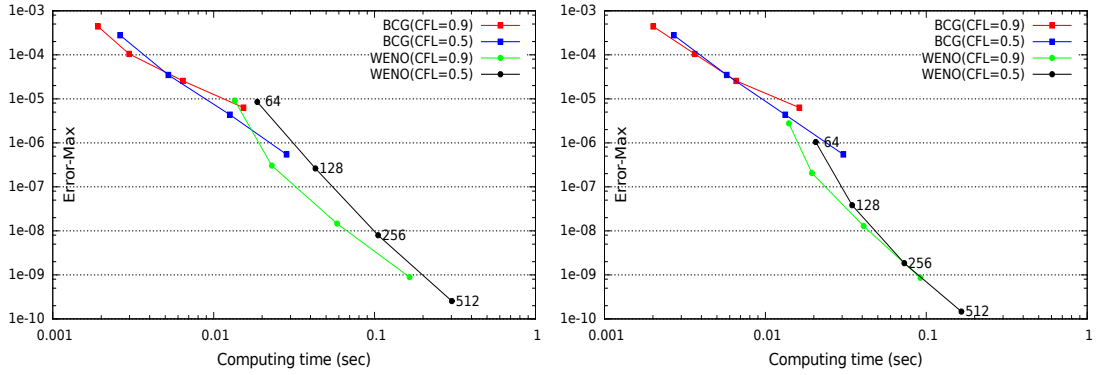


Figure 2.2: Convergence of maximum error for BCG and WENO schemes.



(a) WENO scheme with limiting option. **(b)** WENO scheme with no limiting option.

Figure 2.3: Computing performance - BCG vs WENO schemes.

The figure 2.2 shows the convergence of the max errors. The results show that while the BCG scheme converges with an order of $\mathcal{O}(\Delta x^3)$ for a CFL number of 0.5 (where the error is dominated by the convection term), while it saturates at $\mathcal{O}(2.11)$ for a higher CFL number of 0.9. The WENO implementation with an RK4 time marching shows a convergence of $\mathcal{O}(\Delta x^{5.01})$ at CFL 0.5, while the scheme convergence falls down to $\mathcal{O}(4.57)$ for a CFL of 0.9. In both cases, the conclusion is that the errors accumulated due to time marching, starts dominating as we increase the CFL number, which is expected.

The computation time vs max error plot (fig. 2.3a) shows some interesting result. The cost of computation in terms of time is almost the same for the WENO-RK4 scheme as compared to the BCG scheme. Take a look at CFL=0.9 and compare the time the BCG and the WENO solver takes to get to a solution with an error of 10^{-5}. The BCG solver requires 512 grid points, while the WENO scheme gives the same error at 64 grid points. Both have identical computation times. However, for errors lower than 10^{-5}, the WENO scheme will have a faster run time. This system will have a smooth and continuous solution at all times. Hence, we can further optimize the problem, by running a WENO scheme, which has no limiting option. We do not calculate the smoothness factor and the weights, but use the γ values to sum up individual stencil contributions. This gives a full stencil formulation as given by eq. 2.32. We now achieve significant speed up, as evident from figure. 2.3b. This provides a justification to work with higher-order schemes in Basilisk, since there is a significant gain in both error convergence of the solution as well as computation cost.

$$\begin{aligned} u^{S0-}_{i-1/2} &= \frac{1}{30}\overline{u_{i-3}} - \frac{13}{60}\overline{u_{i-2}} + \frac{47}{60}\overline{u_{i-1}} + \frac{9}{20}\overline{u_i} - \frac{1}{20}\overline{u_{i+1}} \\ u^{S0+}_{i-1/2} &= -\frac{1}{20}\overline{u_{i-2}} - \frac{27}{60}\overline{u_{i-1}} + \frac{47}{60}\overline{u_i} + \frac{13}{60}\overline{u_{i+1}} - \frac{1}{60}\overline{u_{i+2}} \end{aligned} \tag{2.32}$$

Next, we compare the performance of the different temporal schemes. We run simulations for a range of CFL numbers ranging from 0.05 upto 0.82. The results are plotted in fig. 2.4. It is of course evident that for a given grid resolution the RK4 scheme will be the costliest while the RK2 scheme will be the cheapest. However, once we start looking at the error norms plotted on the y-axis it is deducable why the RK-4 scheme outperforms the other two schemes. At higher CFL numbers (near to 0.8), which is where we would like to solve such convection dominated problems, the RK-4 scheme gives errors which are lower than the errors of the convection weno scheme (hence, we see saturation of errors for the RK4 scheme even as the time step is further lowered), which is not the case for the RK-3 or RK-2 methods, where the accumulated errors due to the time marching scheme dominate the solution.

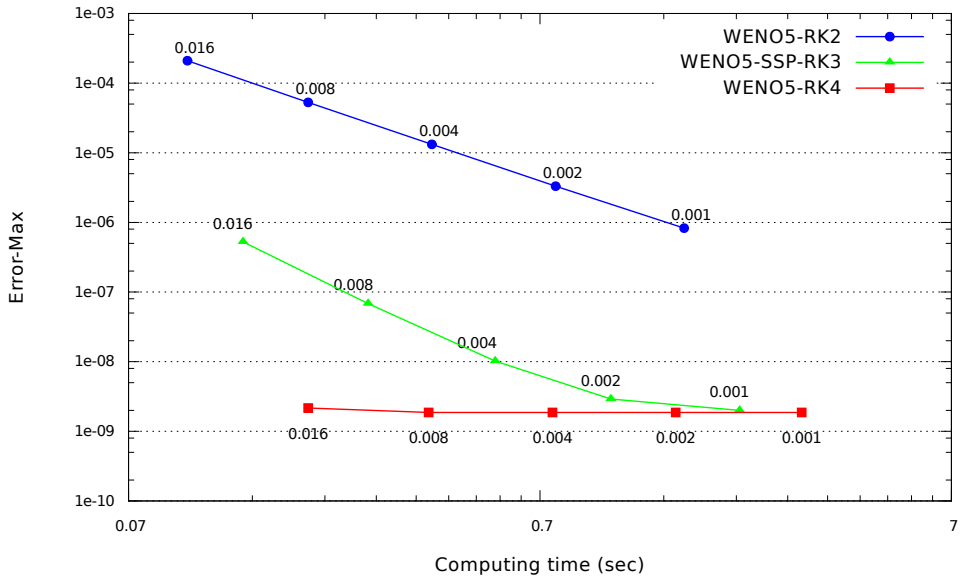


Figure 2.4: *Computation Time performance graph using RK2, SSP-RK3 and RK4 schemes, Grid Resolution = 512. Chart data labels represent simulation timesteps. Equivalent CFL number* $\subset [0.05 : 0.82]$.

In conclusion from this test case, we can say that the usage of a higher-order WENO scheme has pushed the performance both in context of error norms as well as in respect to computing time. It is important to note that, fluid dynamic systems which have a continuous solution should use the RK4 time-marching scheme in conjunction with the WENO scheme with no limiting option, whereas for systems which have discontinuous solutions, we should go for SSP-RK3 time marching scheme with a full WENO5 scheme with limiting.

2.7.2 Passive advection of a 1D discontinuous tracer field

Here, we solve another test case to demonstrate the limiting capability of the WENO scheme. Here we have tracer fields which are discontinuous, and are advected by a uniform flow, similar to the previous test case. The initial tracer field contains a smooth but narrow combination of Gaussians, a square wave, a sharp triangular wave and a half ellipse. The formulation of the tracer is given by eq. 2.33.

$$Tracer(x, t=0) = \begin{cases} \frac{1}{6}\big(G(x,\beta,z-\delta) + G(x,\beta,z) + G(x,\beta,z+\delta)\big), & \text{if } -0.8 \le x \le 0.6 \\ 1, & \text{if } -0.4 \le x \le -0.2 \\ 1 - |10(x-0.1)|, & \text{if } 0 \le x \le 0.2 \\ \frac{1}{6}\big(F(x,\alpha,a-\delta) + F(x,\alpha,a) + F(x,\alpha,a+\delta)\big), & \text{if } 0.4 \le x \le 0.6 \\ 0, & \text{otherwise} \end{cases} \tag{2.33}$$

where,

$$G(x,\beta,z) = e^{-\beta(x-z)^2}$$

$$F(x,\alpha,a) = \sqrt{max(1-\alpha^2(x-a)^2, 0)}$$

$$a = 0.5,\ z = -0.7,\ \delta = 0.005,\ \alpha = 10 \text{ and } \beta = \frac{log(2)}{36\delta^2}$$

A uniform velocity field is imposed and the tracer is advected with periodic boundary conditions over one full time period. First we look at the performance of the BCG scheme in figure 2.5. We compare the results using different limiter options in fig. 2.5. There is a case with no limiter used as well, and this case clearly shows numerical oscillations. The Minmod is certainly more dissipative than the Superbee limiter, which is what we expect.

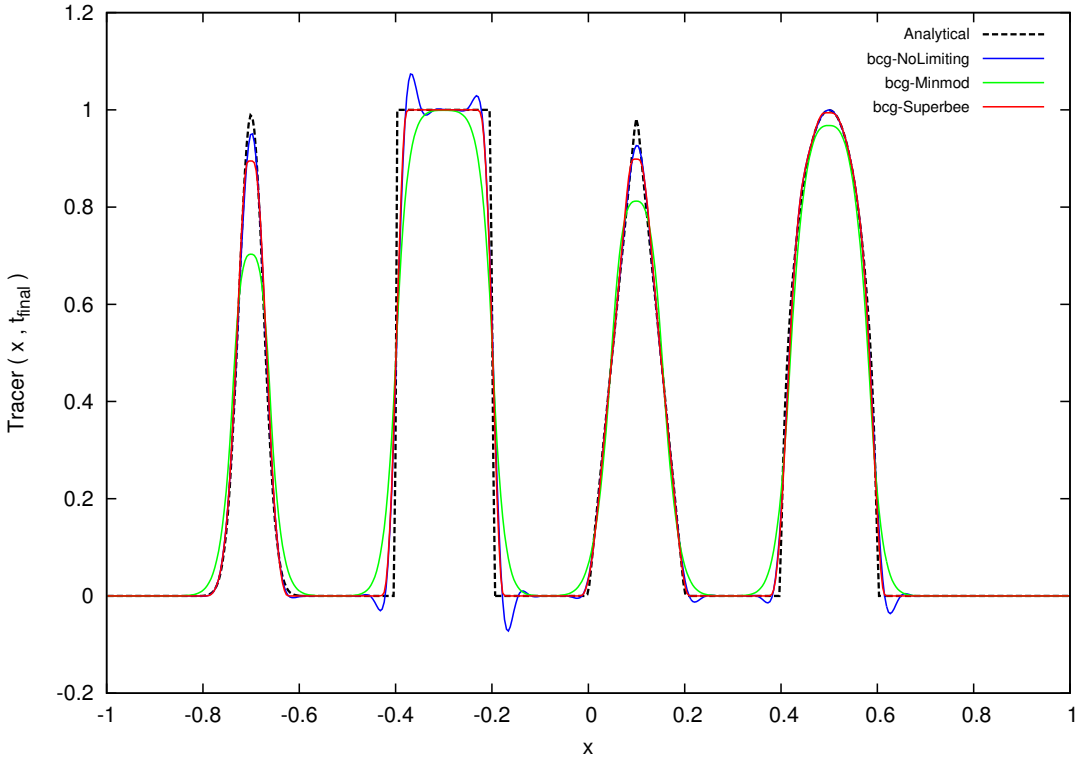


Figure 2.5: *BCG advection scheme results with different limiting options - Minmod, Superbee & NoLimiter. Grid Resolution = 512, CFL = 0.4.*

Next we compare the BCG-Superbee scheme to the WENO5-Limiting with SSP-RK3 time marching scheme (fig. 2.6), to understand the behaviour of these schemes near sharp field

discontinuities or regions of sharp discontinuities of field derivatives. The WENO scheme performs better compared to the Superbee for the narrow Gaussian wave and for the triangular wave while it has an almost equal performance for the square wave. However the WENO-5 SSP-RK3 is better than the minmod limiter in all four tracer features.

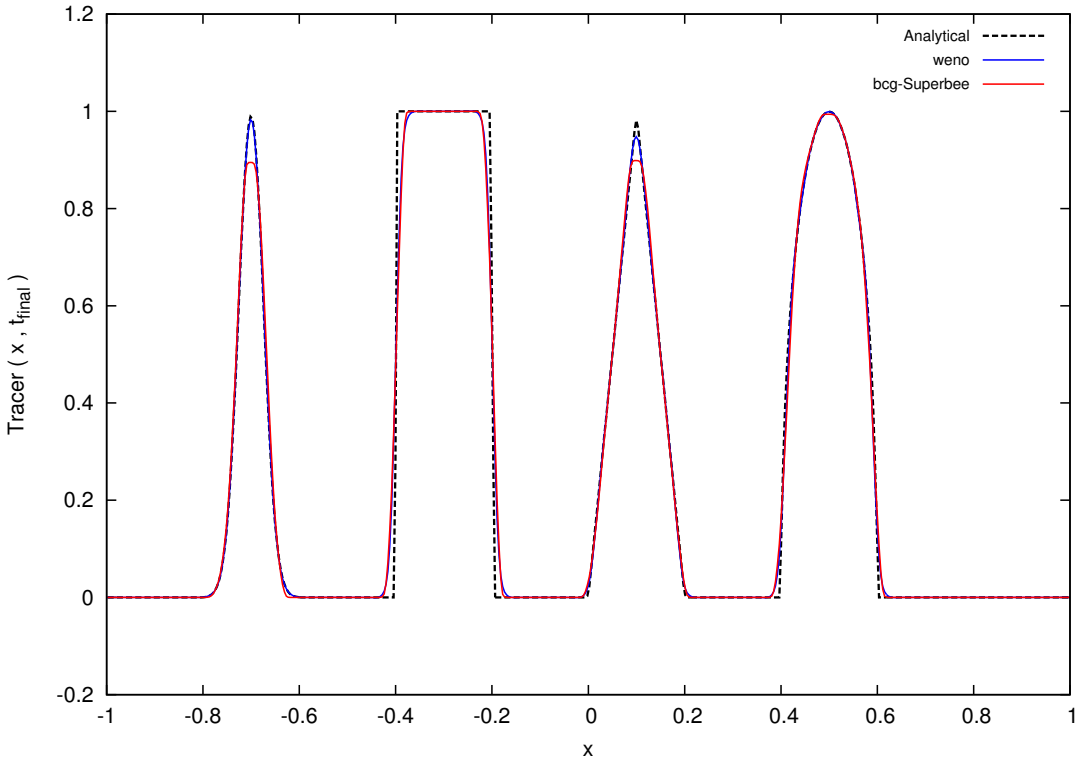


Figure 2.6: *Advection scheme results - BCG with Superbee limited scheme & WENO-5 limited scheme. Grid Resolution = 512, CFL = 0.4.*

In conclusion to the 1D discontinuous test case, it can be said that the WENO scheme, when compared to the superbee limited BCG scheme does show a better limiting performance, in flow features where the gradient of the field has a sharp discontinuity, but will have a marginally lower performance in regions which are very close to field discontinuities. With this we end the section on 1D test cases and move on to implement the generalized higher-dimension WENO scheme, and study its performance.

2.8 WENO in 2D and 3D cases

Rather than going for a multivariate polynomial interpolation function, a commonly used and simple formulation is the dimension by dimension approach. Let us look at the initial value problem of the form given by eq. 2.34.

$$\begin{gathered} \frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} + \frac{\partial g(u)}{\partial y} = 0 \\ u(x, y, t = 0) = 0 \end{gathered} \tag{2.34}$$

The semi-discrete form of the equation with mesh width Δx and Δy is expressed as eq. 2.35 & 2.36

$$U'_{i,j}(t) = -\frac{1}{\Delta x}\Big\{\mathcal{F}_{i+1/2,j}(t) - \mathcal{F}_{i-1/2,j}(t)\Big\} - \frac{1}{\Delta y}\Big\{\mathcal{G}_{i,j+1/2}(t) - \mathcal{G}_{i,j-1/2}(t)\Big\} \tag{2.35}$$

$$\begin{aligned}
\text{Where, } U_{i,j}(t) &= \frac{1}{\Delta x \Delta y}\int_{x_{i-1/2}}^{x_{i+1/2}}\int_{y_{j-1/2}}^{y_{j+1/2}} u(x,y,t)dydx \\
\mathcal{F}_{i\pm1/2,j}(t) &= \frac{1}{\Delta y}\int_{y_{j-1/2}}^{y_{j+1/2}} f(u(x_{i\pm1/2},y,t))dy \\
\mathcal{G}_{i,j\pm1/2}(t) &= \frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} g(u(x,y_{j\pm1/2},t))dx
\end{aligned} \tag{2.36}$$

The idea behind the WENO method is to numerically approximate the exact fluxes $\mathcal{F}$ and $\mathcal{G}$ with the approximate WENO fluxes F and G such that eq. 2.37 is satisfied

$$\begin{aligned}
F_{i\pm1/2,j} &= \mathcal{F}_{i\pm1/2,j} + \mathcal{O}(\Delta x^5 + \Delta y^5) \\
G_{i,j\pm1/2} &= \mathcal{G}_{i,j\pm1/2} + \mathcal{O}(\Delta x^5 + \Delta y^5)
\end{aligned} \tag{2.37}$$

Applying one-dimensional WENO reconstruction from the previous section, we compute averaged values of the conserved quantities at all grid cell interfaces, which are denoted by $U^{\pm}_{i+1/2,j}$ and $U^{\pm}_{i,j+1/2}$, expressed in eq. 2.38.

$$\begin{aligned}
U^{\pm}_{i+1/2,j} &= \frac{1}{\Delta y}\int_{y_{j-1/2}}^{y_{j+1/2}} u(x_{i+1/2},y)dy + \mathcal{O}(\Delta x^5) \\
U^{\pm}_{i,j+1/2} &= \frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} u(x,y_{j+1/2})dx + \mathcal{O}(\Delta y^5)
\end{aligned} \tag{2.38}$$

Subsequently, the flux can be computed using the formulation $F_{i-1/2,j} = f_{riemann}(U^{+}_{i-1/2,j}, U^{-}_{i-1/2,j})$ However this computation will reduce the order of the approximation down to two when the flux is not a linear function of u, because a point value at the center of the line is only a 2^{nd}-order accurate approximation to the line averaged value, we can observe this in eq. 2.39.

$$\begin{aligned}
f(U_{i+1/2,j}) &= f(\frac{1}{\Delta y}\int_{y_{j-1/2}}^{y_{j+1/2}} u(x_{i+1/2},y)dy + \mathcal{O}(\Delta x^5)) \\
&= f(u(x_{i+1/2},y_j)) + \mathcal{O}(\Delta x^5 + \Delta y^2) \\
&= \mathcal{F}_{i+1/2,j} + \mathcal{O}(\Delta x^5 + \Delta y^2)
\end{aligned} \tag{2.39}$$

Hence, to mitigate the reduction of order a new method is required which captures not just the averaged value of u over the faces but rather the distribution of u, which would require an interpolation sweep in the transverse directions.

2.8.1 Transverse sweeps - Gaussian quadratures

In numerical analysis, a Gaussian quadrature is used to construct an approximation to the definite integral of a function, usually carried out through a weighted sum of function values at specified point locations within the integration domain. An n-point Gaussian quadrature (given by eq. 2.40) rule can evaluate exact results for polynomial functions up to an order $2n-1$. We will take a look at the 2-point and the 3-point Gaussian quadrature formulations and evaluate their convergence performance on a test problem.

$$\begin{aligned}
\int_{-1}^{1} f(x)dx &= \sum_{i=1}^{n} w_i f(x_i) \\
\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right) &= \frac{b-a}{2}\sum_{i=1}^{n} w_i f\left(\frac{b-a}{2}x_i + \frac{a+b}{2}\right)
\end{aligned} \tag{2.40}$$

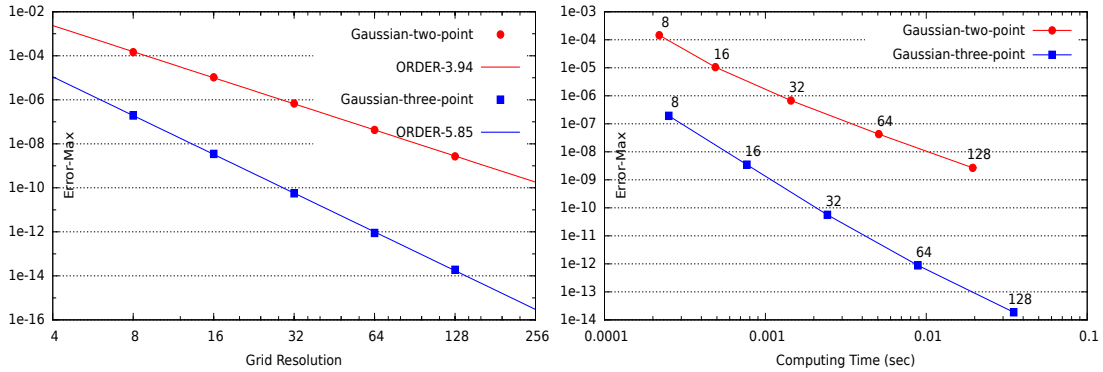
- **Two point formulation**: The points are located at $x_1 = -\frac{1}{\sqrt{3}}$ and $x_2 = \frac{1}{\sqrt{3}}$ The associated weights are $w_1 = 1$ and $w_2 = 1$.

- **Three point formulation**: The points are located at $x_1 = -\sqrt{\frac{3}{5}}$, $x_2 = 0$ and $x_3 = \sqrt{\frac{3}{5}}$ The associated weights are $w_1 = \frac{5}{9}$, $w_2 = \frac{8}{9}$ and $w_3 = \frac{5}{9}$.

To study the convergence of the two methods, we look at a domain $(x, y) \subset [-0.5 : 0.5] \times [-0.5 : 0.5]$, and chose a function $f(x, y) = sin(2\pi x)sin(2\pi y)$. For a domain uniformly divided into grids of spacing Δ, we know the analytical cell volume average is given by the eq. 2.41.

$$\frac{1}{\Delta^2}\int_{x_i-\frac{\Delta}{2}}^{x_j+\frac{\Delta}{2}}\int_{y_j-\frac{\Delta}{2}}^{y_j+\frac{\Delta}{2}} f(x,y)dydx = \left\{\frac{cos(2\pi(x_i-\frac{\Delta}{2})) - cos(2\pi(x_i+\frac{\Delta}{2}))}{2\pi\Delta}\right\}\times \left\{\frac{cos(2\pi(y_j-\frac{\Delta}{2})) - cos(2\pi(y_j+\frac{\Delta}{2}))}{2\pi\Delta}\right\} \tag{2.41}$$

We then use the Gaussian quadrature methods to numerically compute the cell volume averages and compare it with the analytical results to compute the error norms. We repeat the test for a number of different grid spacings from grids of level 3 to level 7. The error norm is plotted on a log-log scale against the grid points to give the slope of the two methods as illustrated in fig. 2.7a, while the computing time performance is plotted in the fig. 2.7b.



(a) Error convergence. **(b)** Computation cost.

Figure 2.7: Two-point and three-point Gaussian Quadrature methods.

The three-point quadtrature formulation has an error convergence of 6, while the two-point formulation has an error convergence of order 4. We however are working with WENO formulations which are of order 5, and will also be developing order 5 prolongation functions (used for multigrid iterations and adaptive grids), both of which use Gaussian quadrature formulations in their derivations, and hence we would not want the order of the method to be restricted by the choice of the quadrature method. Therefore we made a choice to use the $\mathcal{O}(6)$ three-point Gaussian quadrature method, and as can be seen from fig. 2.7b, the time penalty we pay is negligible, while the error reduction is substantial for the three-point method compared to the two-point method.

Picking up the analysis from eq. 2.38, the idea here is to use the averaged u values in the transverse direction to compute point values at Gaussian quadrature locations. These point values of u are used to compute the respective point values of the fluxes $f(u)$. Then a Gaussian integration is carried over the quadrature point fluxes to get the average face flux values which now has an approximation error of order 5 instead of 2. We use the three-point quadrature formulation as shown in eq. 2.42 & 2.43.

$$F_{i+1/2,j} = \frac{1}{\Delta y}\int_{y_{j-1/2}}^{y_{j+1/2}} f(u(x_{i+1/2}, y))dy \approx \frac{1}{\Delta y}\sum_{s=1}^{3} c_s f_{i+1/2,j_s} \tag{2.42}$$

$$\text{where,} \quad y_{j_1} = y_j - \frac{\Delta y}{2}\sqrt{\frac{3}{5}}, \quad y_{j_2} = y_j, \quad y_{j_3} = y_j + \frac{\Delta y}{2}\sqrt{\frac{3}{5}} \\ c_1 = \frac{5}{9}, \quad c_2 = \frac{8}{9}, \quad c_3 = \frac{5}{9} \tag{2.43}$$

Hence, first we need to compute the $\mathcal{O}(5)$ approximations of point values of the conserved quantity $\mathbf{u}$ at the Gaussian quadrature points, using eq. 2.44.

$$u^{\pm}_{i+1/2,j_s} = u(x_{i+1/2,y_{j_s}}) + \mathcal{O}(\Delta x^5 + \Delta y^5) \tag{2.44}$$

At any particular quadrature point $(x_{i+1/2}, y_{j_s})$, the $\mathcal{O}(5)$ weno interpolations can be carried out by using eq. 2.45 & 2.46, the coefficients which are listed in the table below.

$$u_{i+1/2,j_s} = \gamma^s_0 u^{(0)}_{i+1/2,j_s} + \gamma^s_1 u^{(1)}_{i+1/2,j_s} + \gamma^s_2 u^{(2)}_{i+1/2,j_s} \tag{2.45}$$

$$\text{with,} \quad u^{(0)}_{i+1/2,j_s} = a^s_{0,0} U_{i+1/2,j-2} + a^s_{0,1} U_{i+1/2,j-1} + a^s_{0,2} U_{i+1/2,j} \\ u^{(1)}_{i+1/2,j_s} = a^s_{1,0} U_{i+1/2,j-1} + a^s_{1,1} U_{i+1/2,j} + a^s_{1,2} U_{i+1/2,j+1} \\ u^{(2)}_{i+1/2,j_s} = a^s_{2,0} U_{i+1/2,j} + a^s_{2,1} U_{i+1/2,j+1} + a^s_{2,2} U_{i+1/2,j+2} \tag{2.46}$$

l	0	1	2
$a^1_{0,l}$	$\frac{2-3\sqrt{15}}{60}$	$\frac{-4+12\sqrt{15}}{60}$	$\frac{62-9\sqrt{15}}{60}$
$a^1_{1,l}$	$\frac{2+3\sqrt{15}}{60}$	$\frac{56}{60}$	$\frac{2-3\sqrt{15}}{60}$
$a^1_{2,l}$	$\frac{62+9\sqrt{15}}{60}$	$\frac{-4-12\sqrt{15}}{60}$	$\frac{2+3\sqrt{15}}{60}$
$a^1_{0,l}$	$-\frac{1}{24}$	$\frac{2}{24}$	$\frac{23}{24}$
$a^1_{1,l}$	$-\frac{1}{24}$	$\frac{26}{24}$	$-\frac{1}{24}$
$a^1_{2,l}$	$\frac{23}{24}$	$\frac{2}{24}$	$-\frac{1}{24}$
$a^1_{0,l}$	$\frac{2+3\sqrt{15}}{60}$	$\frac{-4-12\sqrt{15}}{60}$	$\frac{62+9\sqrt{15}}{60}$
$a^1_{1,l}$	$\frac{2-3\sqrt{15}}{60}$	$\frac{56}{60}$	$\frac{2+3\sqrt{15}}{60}$
$a^1_{2,l}$	$\frac{62-9\sqrt{15}}{60}$	$\frac{-4+12\sqrt{15}}{60}$	$\frac{2-3\sqrt{15}}{60}$

k	0	1	2
γ^1_k	$\frac{1008+71\sqrt{15}}{5240}$	$\frac{403}{655}$	$\frac{1008-71\sqrt{15}}{5240}$
γ^{2+}_k	$\frac{9}{80}$	$\frac{49}{20}$	$\frac{9}{80}$
γ^{2-}_k	$\frac{9}{40}$	$\frac{49}{40}$	$\frac{9}{40}$
γ^3_k	$\frac{1008-71\sqrt{15}}{5240}$	$\frac{403}{655}$	$\frac{1008+71\sqrt{15}}{5240}$

For, the quadrature point y_{j_2}, we follow the approach of Shi et al. (2002) and split γ^2_k, for $k = 0, 1, 2$ into a positive and a negative part. We compute the interpolation using eq. 2.47.

$$u_{i+1/2,j_{2\pm}} = \gamma^{2\pm}_0 u^{(0)}_{i+1/2,j_2} + \gamma^{2\pm}_1 u^{(1)}_{i+1/2,j_2} + \gamma^{2\pm}_2 u^{(2)}_{i+1/2,j_2} \\ u_{i+1/2,j_2} = \sigma^+ u_{i+1/2,j_{2+}} - \sigma^- u_{i+1/2,j_{2-}} \\ \sigma^+ = \sum_{k=0}^{2} \gamma^{2+}_k = \frac{214}{80} \quad \text{and} \quad \sigma^- = \sum_{k=0}^{2} \gamma^{2-}_k = \frac{67}{40} \tag{2.47}$$

2.9 Test case - Passive advection in a 2D domain

In this section we will take a look at two different two-dimensional test cases to observe the performance of the WENO schemes in Basilisk. Both cases involve the passive advection of a tracer field under the influence of a given velocity field. The results and performance of the WENO methodology are compared and contrasted with the available second-order Bell, Collella and Glaz scheme.

2.9.1 Periodic tracer in a uniform velocity field

The 2D domain chosen is given by : $(x, y) \in [-0.5 : 0.5, -0.5 : 0.5]$, where the periodic tracer function has the formulation given in eq. 2.48. The velocities are initialized with constant values as shown in eq. 2.49. The time marching is carried out over one entire time period and the error is estimated by comparing the initial and the final tracer data. We use the WENO non-limited scheme along with RK4 time marching. The simulations are run for two distinct CFL numbers.

$$Tr(x, y, t = 0) = sin(2\pi x)sin(2\pi y) \tag{2.48}$$

$$\mathbf{u} = 0.1\hat{x} + 0.1\hat{y} \tag{2.49}$$

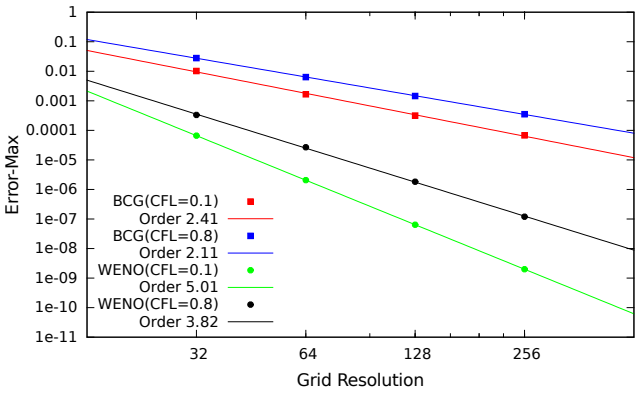
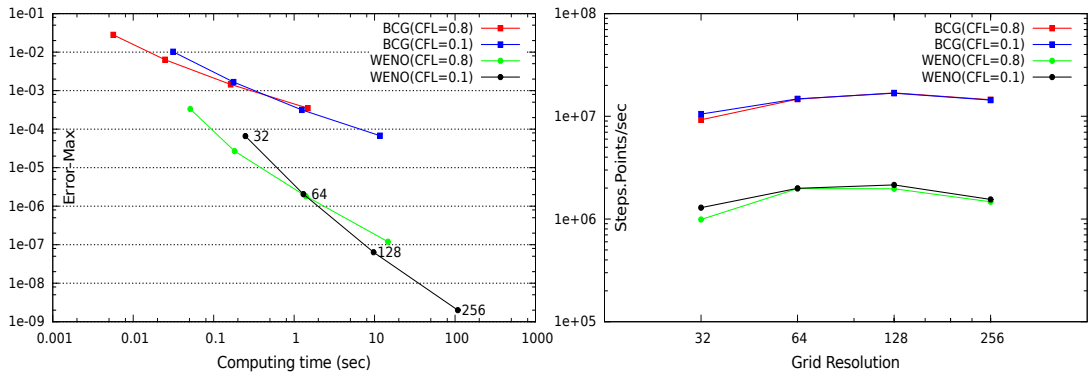


Figure 2.8: Error convergence of the 2D advection solver (BCG vs WENO).



(a) Error vs computing time. **(b)** Computing speed vs resolution.

Figure 2.9: Two-dimensional advection of a continuous periodic tracer - BCG vs WENO.

The error convergence ranges from $\mathcal{O}(3.8)$ to $\mathcal{O}(5.01)$ for the WENO scheme for a flow CFL of 0.8 to 0.1 respectively. Fig. 2.9a shows that the relative overall performance of the WENO scheme is better compared to the BCG scheme. For instance, to achieve an error level of 10^{-4}, the WENO scheme takes approximately one-fourth the time of the BCG scheme. The individual WENO scheme timesteps are roughly 10 times more expensive than the BCG scheme timesteps, as can be seen from Fig. 2.9b.

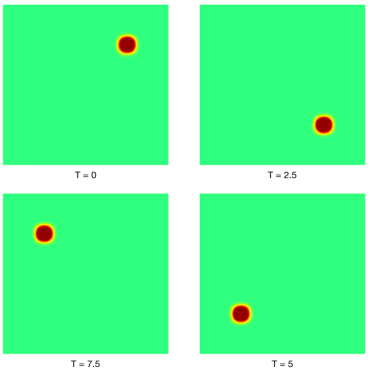
2.9.2 Compact tracer in a solid body rotation

In this test case, instead of a plain translation we introduce a solid body rotation advection current. This complicates the computation of the convection terms, since now along with the tracer field, the velocity field is also variable and a function of space, and this is where the importance of the quadrature calculations really come out. We start with a compact tracer function of the form given be eq. 2.50 and our velocity field can be expressed using the eq. 2.51.

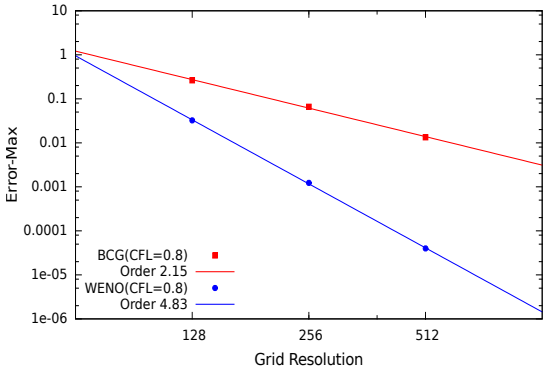
$$Tr(x,y) = \begin{cases} \left\{1 - \left(\frac{x-0.25}{0.1}\right)^2\right\}^7 \times \left\{1 - \left(\frac{y-0.25}{0.1}\right)^2\right\}^7, & \text{if} \quad (x-0.25)^2 \leq 0.01 \;\&\; \\ & \quad (y-0.25)^2 \leq 0.01 \\ 0, & \text{otherwise} \end{cases} \tag{2.50}$$

$$\mathbf{u} = -\frac{\pi y}{5}\hat{x} + \frac{\pi x}{5}\hat{y} \tag{2.51}$$

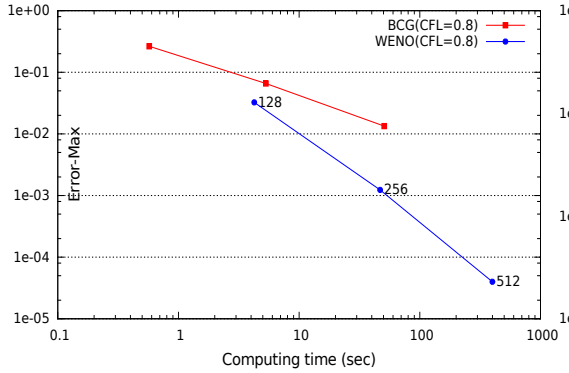
The simulation is run for one time period (10 sec.) i.e. till the tracer comes back to its original position after one full rotation. The error convergence study is carried out by matching the initial to the final solution. This simulation has been run at a relatively higher CFL number of 0.8. The results are presented in the fig. 2.10.



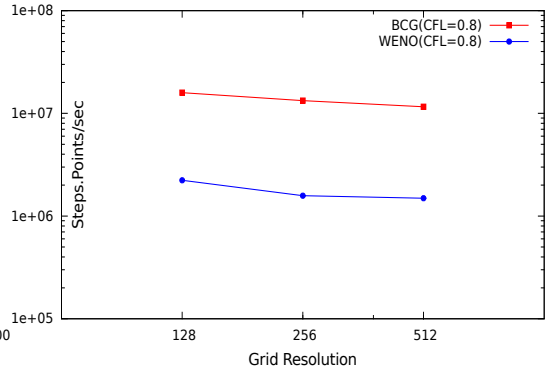
(a) Time series snapshots of the tracer field.



(b) Error convergence of the solver.



(c) Error vs computing time.



(d) Computing speed vs resolution.

Figure 2.10: Two dimensional solid body rotation of a continuous compact tracer - BCG vs WENO.

We observe a convergence of $\mathcal{O}(4.83)$ for the WENO scheme (fig. 2.10b), which validates the quadrature formulations used in the Basilisk code. The fig. 2.10c shows that the WENO scheme requires lesser time than the BCG scheme to attain similar error level. For eg. to reach a target error of 10^{-2}, the WENO scheme will be approximately five times faster than the BCG scheme. The graph fig. 2.10d shows that each time step of the BCG scheme is approximately 10 times faster compared to the WENO scheme.

2.10 Multi-resolution analysis

Wavelets are special mathematical functions which are used for representing data or other functions and satisfy certain mathematical conditions. The idea behind wavelets can be said to be borrowed from the work of Joseph Fourier, who introduced the Fourier transform, which represents functions using a superposition of sines and cosines. The distinguishing feature of a wavelet transform is that here the scale at which we look at the data plays a pivotal role. This means wavelet based algorithms process data at different scales of resolutions. To put it simply, if we were looking at a one dimensional field over a large x-window, we would observe the gross features of the field or the coarse resolution behavior. However, if we were to zoom in and observe the field in a narrow margin of x, we would observe the smaller scale crests and troughs of the signal, or its behavior at the fine scales as shown in fig. 2.11.

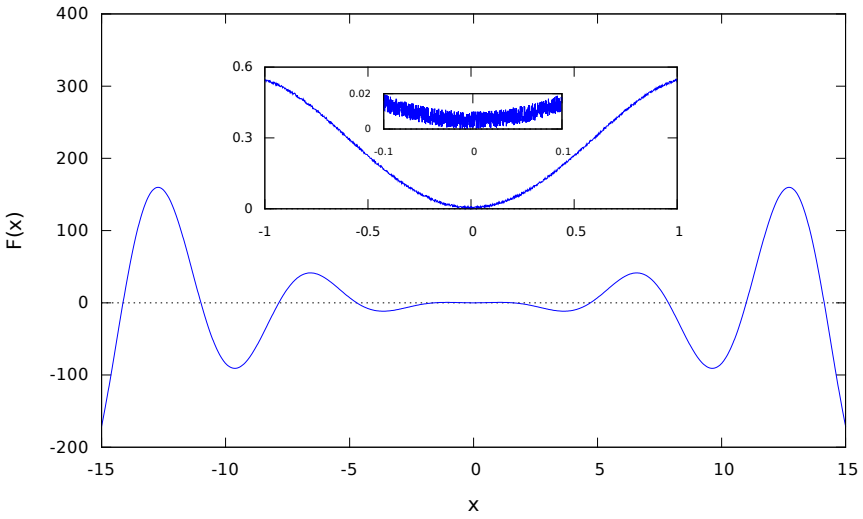
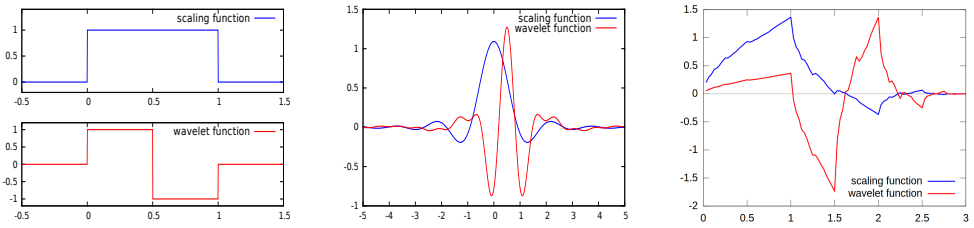


Figure 2.11: A 1D field $f(x, t = T_0)$ full view & zoomed in views.

The wavelet algorithm therefore is a powerful mathematical tool which can suitably represent a function / dataset at multiple levels of resolution. Unlike Fourier analysis, wavelet analysis expands functions in terms of wavelets which are generated in the form of dyadic translations and dilations of a fixed function called the mother wavelet. Fourier transforms have the disadvantage that the basis sine and cosine functions are non-local (and stretch out to infinity). They are therefore unsuitable for approximating sharp spikes. However with wavelets, the approximating functions are contained in finite domains, hence making them suitable for approximating discontinuities. This localization feature, makes many functions and operators using wavelets "sparse" when transformed into the wavelet domain. This sparseness, in turn, results in a number of useful applications such as data compression, removing noise from time series, or as in our case in refining adaptive grids for CFD computation.

The first wavelet was introduced by Haar in his 1909 PhD Thesis. One property of the Haar wavelet is that it has compact support, which implies it has zero value outside a finite interval. Unfortunately, Haar wavelets are not continuously differentiable which somewhat limits their applications. Later on, starting in 1985, Stephane Mallat pursued wavelets through his work in digital signal processing. He introduced the multiresolution framework and discovered some important relationships on orthonormal wavelet bases. His seminal works are published in Mallat (1989a), Mallat (1989b) & Mallat (1989c). Later, Y. Meyer pursued Mallat's work and constructed the first non-trivial wavelets (Meyer (1992)). Even though the Meyer wavelets are continuously differentiable, they do not have compact support. Ingrid Daubechies constructed a set of wavelet orthonormal basis functions that have today become the cornerstone of wavelet applications (Daubechies (1988) & Daubechies (1992)). Later, Wim Sweldens, in his paper Sweldens (1998), introduced the lifting scheme, which led to a simple construction of second generation wavelets, i.e. wavelets that are not necessarily translates and dilates of one fixed function. It is used for constructing wavelets as well as for performing the discrete wavelet transform.



(a) **(b)** **(c)**

Figure 2.12: *Different 1st Generation wavelets : (a) A Haar scaling function and wavelet, (b) A Meyer scaling function and wavelet & (c) A Daubechies D4 scaling function and wavelet - Image from Wikimedia.*

2.10.1 Wavelet Transform - Lifting Algorithm

Imagine a finely resolved signal, which has 2^j samples. Here j represents the level of refinement of the signal. The idea behind a wavelet transform is to subsequently split this finely resolved signal at level j, into a coarse signal at level $(j-1)$, and a corresponding detail signal at level j. This exercise can be expressed mathematically by the formulation: $s_j = s_{j-1} + d_j$. This process is carried out using the lifting algorithm by Sweldens. Following is a brief description of the three step wavelet transform algorithm.

1. **Split**: This stage essentially splits the signal at level j into two disjoint set of signals. One set consists the even indexed samples whereas the other set has the odd indexed samples. The operator can be expressed as:

$$Split(s_j) = (even_{j-1}, odd_{j-1})$$

2. **Predict**: Given the local correlation of the signal, it can be said that the interspersed even and odd signals will be highly correlated or in simple words, given one of the two sets it should be possible to predict the other set. This defines the Predictor operator and the detail signal is given by the formulation:

$$d_j = odd_{j-1} - P(even_{j-1})$$

3. **Update**: While carrying out the step to compute the coarser averaged signal s_{j-1}, it must be remembered that it has the same average value as the finer level signal. This process defines an Update operator U such that the identity of equal averages irrespective of level is maintained. The coarse level signal is then given by the following formulation:

$$s_{j-1} = even_{j-1} + U(d_j)$$

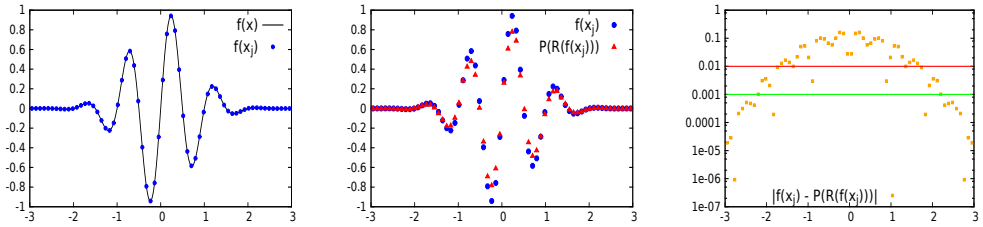
By following the above algorithm of split, predict, update on a signal with 2^n samples, and applying it repeatedly n times we can finally get a coarse base signal s_0 and n different detail signals d_l, where $1 \leq l \leq n$. In case of 1st generation wavelets, the detail signals at each step can subsequently be resolved into an orthonormal basis. These basis functions are called wavelet functions. They have certain inbuilt mathematical properties, for instance, they are smooth, compact and are dyadic translations and dilations of one fixed mother function called the mother wavelet $\psi(x)$. Using this mother wavelet, the detail signal at any resolution level can thus be expressed as a wavelet transform given by eq. 2.52

$$\begin{aligned} \psi_{j,l}(x) &= \psi(2^j x - l) \\ d_j &= \sum_{l=-\infty}^{\infty} A_{j,l} \psi_{\mathbf{j,l}}(\mathbf{x}) \end{aligned} \tag{2.52}$$

The coefficients $A_{j-1,l}$'s are called the wavelet coefficients, and they play a major role in determining which regions of the adaptive grids need to be refined or coarsened.

2.10.2 Wavelet transform and Basilisk adaptivity

In this section, I will explain the Basilisk implementation of adaptivity using a 1D example. Consider a 1D field : $f(x) = e^{-x^2} sin(2\pi x)$. Suppose the field is currently resolved at a uniform level j, giving the sampling values $f(x_j)$, as shown in fig. 2.13a. The idea is to carry out one level of wavelet transform to obtain the difference field representation at level j. The first step is to split the signal at level j and use the two distinct even and odd signals to construct an intermediate signal at level $j-1$. We call this process as restriction in Basilisk. The next step is to use the sample points from this newly constructed restriction signal to interpolate a new signal back to the level j. This process is called the prolongation step. Thus we arrive back at a new signal, which we can mathematically designate as $P(R(f(x_j)))$, where $R : x_j \rightarrow x_{j-1}$ denotes a restriction operator carried out on fine grid sample values, and $P : x_{j-1} \rightarrow x_j$ denotes a prolongation operator carried out on coarse grid sample values. Both the original signal $f(x_j)$ and the restricted-prolongated signal $P(R(f(x_j)))$ are plotted in fig. 2.13b.



(a) **(b)** **(c)**

Figure 2.13: *Basilisk adaptivity: (a) A continuous and discrete representation of a function f(x), (b) Orignal discrete representation and the restricted-prolongated representation of the function f(x), & (c) Error signal computed from the original discrete signal and restricted-prolongated signal.*

The next step is the calculation of the difference signal which is obtained by subtracting the restricted-prolongated signal from the original signal. We are not interested in the update step of the lifting algorithm while implementing adaptivity. So we now have a discrete difference signal. We can think of the individual values of this difference signal to correspond to wavelet coefficients. We plot the absolute values of the difference signal in fig. 2.13c. The *adapt_wavelet* function which implements adaptivity in Basilisk requires the user to set an upper tolerance condition on the field. In this eg. I assume the user set value is 10^{-2}, represented by the red line in fig. 2.13c. All grid points where the difference signal has values higher than the given tolerance limit are refined. Similarly, a lower tolerance condition is also present in the code (set in this eg as 10^{-3} and marked by the green line), and the grid cells where the difference signal is smaller than this limit are coarsened. Fig. 2.14 shows the drop in the diff signal values as the grid is adaptively refined twice by calling the adapt_wavelet function.

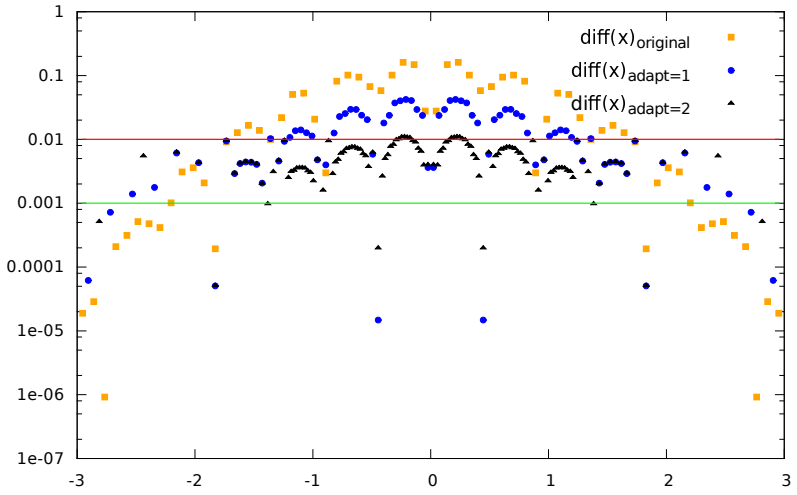


Figure 2.14: Difference signals after one and two runs of adapt_wavelet function.

2.10.3 Restriction operator

The restriction operator basically interpolates the volume averaged values from a fine cell to the next coarser level (fig. 2.15). Since the numerical quantities are all volume averages, a simple arithmetic average computes exactly the volume averaged value. Traditionally Basilisk implements this restriction operator using the formulation given by eq. 2.53.

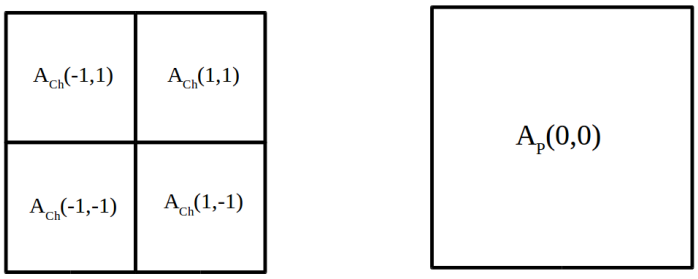


Figure 2.15: A schematic representing the Parent and Child cell layers.

$$A_P(0,0) = \frac{A_{Ch}(1,1) + A(-1,1) + A_{Ch}(1,-1) + A_{Ch}(-1,-1)}{4} \tag{2.53}$$

The formulation is exact and hence there is no order of approximation involved. So the restriction operator that works for a second-order adaptive method will hold equally well for the higher-order method developed as part of this work.

2.10.4 Prolongation operator

The basic idea behind a prolongation operator is to obtain fine cell values from the coarse cell values (fig. 2.16). In the traditional Basilisk method, the prolongation method is a second-order bilinear interpolation method. The method is illustrated below.

Bi-linear-prolongation : The child cell value can be interpolated to $\mathcal{O}(2)$ using eq. 2.54.

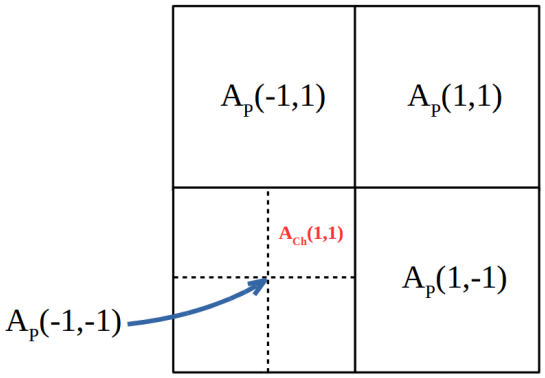


Figure 2.16: A schematic representing the Parent and Child cell layers.

$$A_{Ch}(1,1) = \frac{9A_P(-1,-1) + 3A_P(1,-1) + 3A_P(-1,1) + A_P(1,1)}{16} \tag{2.54}$$

2.10.5 Fifth-order prolongation

Before I began this PhD, Basilisk used the classical restriction operator and the bilinear prolongation operator to implement a second-order numerical method for mesh adaptivity. During the course of my work, I have derived and implemented a new formulation for a fifth-order adaptive meshing scheme which uses the same restriction operator but uses a higher-order prolongation operator. Before I begin describing the process, two important aspects need to be pointed out:

1. Since we are deriving higher-order operators a clear distinction needs to be made between the cell-centered values and cell-averaged values. When the order of the interpolation scheme is two, both compute to the same value, since a second-order interpolation assumes a linear variation of the function in the cell. However, while building a higher-order method, it should be kept in mind that the quantities that we come accross are all cell averages in 2D or volume averages in 3D and hence the interpolation scheme should be a compatible one, which means that it needs to interpolate cell average values given in a coarse cell to derive the cell average values over the fine cell.

2. We might encounter solution fields which may not be a continuous field, but may have sharp discontinuities, eg. shocks etc. We know that traditional polynomial interpolation methods introduce numerical oscillations in the solution near such sharp discontinuities. Hence the process of prolongation must introduce some form of a limiting operator to mitigate the issue of introducing spurious oscillations in the difference signal. This will be achieved by borrowing the limiting idea from the WENO scheme of Liu et al. (1994).

This process involves a combination of polynomial reconstructions and Gaussian quadtrature summations. I will first demonstrate the derivation for a 1D problem, and then move on to generalize the case in 2D and 3D.

Derivation of the prolongation operator in 1D

The basic idea is that in a 1D system, given the five line average coarse cell values marked by the five red dots in fig. 2.17, we should be able to derive a $\mathcal{O}(5)$ mathematical interpolation of the line average in the fine cell (which lies between $x \subset [0 : \frac{1}{2}]$). For this purpose, three Gaussian quadrature points are identified, viz. $x = \frac{1}{4} - \frac{1}{4}\sqrt{\frac{3}{5}}$, $\frac{1}{4}$ and $\frac{1}{4} + \frac{1}{4}\sqrt{\frac{3}{5}}$ respectively. These points are shown in fig. 2.17 by blue points.

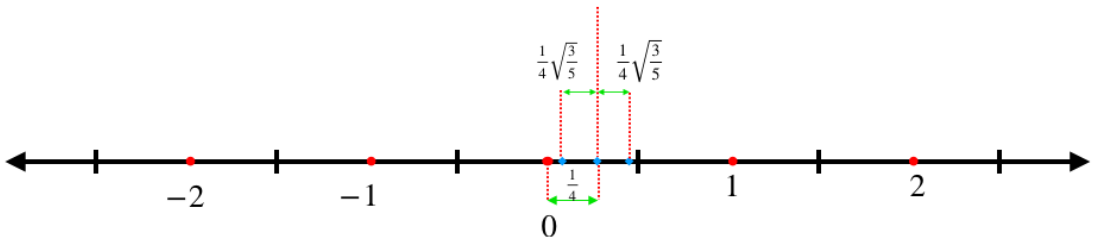


Figure 2.17: Schematic for 1D Gaussian Quadrature Interpolations.

A fifth-order numerical scheme is used to interpolate point values at the 3 quadrature points, from the coarse cell volume average data. However, a fifth-order interpolation without limiting would introduce numerical oscillations is the fine grid, if the interpolation is not limited for discontinuous fields. Hence, we split the stencil into three contributing stencils and borrow the limiting functions from the WENO scheme of Liu et al. (1994). For a smooth and continuously differentiable field, we can build a full five point interpolation scheme without building limiters, to save on computation resources. We start with the split stencil derivation for the generalized case.

1. **Left stencil interpolations**

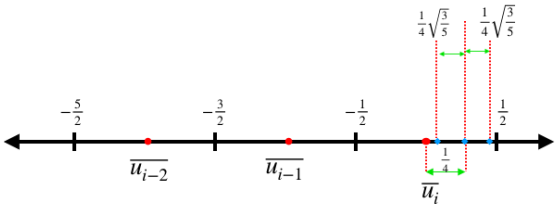


Figure 2.18: Left stencil for interpolation to quadrature point 1.

We define the primitive function $P(x)$ as a third-order polynomial, following a method similar to that used in WENO formulation derivations, as introduced in the paper Liu et al. (1994). The primitive function will by definition satisfy the cell face conditions as listed in eq. 2.55 which can be expressed in a linear system as shown in eq. 2.56. The system can be solved to find the appropriate stencil 1 interpolations on all three quadrature points as listed in 2.57, 2.58 & 2.59.

$$\begin{aligned} P(x) &= Ax^3 + Bx^2 + Cx + D \\ P\Big(-\frac{5}{2}\Big) &= 0 \\ P(-\frac{3}{2}) &= \overline{u_{i-2}} \\ P(-\frac{1}{2}) &= \overline{u_{i-2}} + \overline{u_{i-1}} \\ P(\frac{1}{2}) &= \overline{u_{i-2}} + \overline{u_{i-1}} + \overline{u_i} \end{aligned} \tag{2.55}$$

$$\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} -1/6 & 1/2 & -1/2 & 1/6 \\ -1/4 & 5/4 & -7/4 & 3/4 \\ 1/24 & -1/8 & -7/8 & 23/24 \\ 1/16 & -5/16 & 15/16 & 5/16 \end{bmatrix} \begin{bmatrix} 0 \\ \overline{u_{i-2}} \\ \overline{u_{i-2}} + \overline{u_{i-1}} \\ \overline{u_{i-2}} + \overline{u_{i-1}} + \overline{u_i} \end{bmatrix} \tag{2.56}$$

(a) **Interpolation at point $\mathbf{x_{q1}} = \frac{1}{4} - \frac{1}{4}\sqrt{\frac{3}{5}}$**

$$u(x_{q1})^{S1} = P^{'}(x_{q1}) = 3Ax_{q1}^2 + 2Bx_{q1} + C = \frac{3}{10}A + \frac{1}{2}B + C - \sqrt{\frac{3}{5}}\left\{\frac{3}{8}A + \frac{1}{2}B\right\}$$

$$u(x_{q1})^{S1} = \left\{\frac{2}{15}\overline{u_{i-2}} - \frac{31}{60}\overline{u_{i-1}} + \frac{83}{60}\overline{u_i}\right\} - \sqrt{\frac{3}{5}}\left\{\frac{3}{16}\overline{u_{i-2}} - \frac{5}{8}\overline{u_{i-1}} + \frac{7}{16}\overline{u_i}\right\} \tag{2.57}$$

(b) **Interpolation at point $\mathbf{x_{q2}} = \frac{1}{4}$**

$$u(x_{q2})^{S1} = P^{'}(x_{q2}) = 3Ax_{q2}^2 + 2Bx_{q2} + C = \frac{3}{16}A + \frac{1}{2}B + C$$

$$u(x_{q2})^{S1} = \frac{11}{96}\overline{u_{i-2}} - \frac{23}{48}\overline{u_{i-1}} + \frac{131}{96}\overline{u_i} \tag{2.58}$$

(c) **Interpolation at point $\mathbf{x_{q3} = \frac{1}{4} + \frac{1}{4}\sqrt{\frac{3}{5}}}$**

$$u(x_{q3})^{S1} = P^{'}(x_{q3}) = 3Ax_{q3}^2 + 2Bx_{q3} + C = \frac{3}{10}A + \frac{1}{2}B + C + \sqrt{\frac{3}{5}}\left\{\frac{3}{8}A + \frac{1}{2}B\right\}$$

$$u(x_{q3})^{S1} = \left\{\frac{2}{15}\overline{u_{i-2}} - \frac{31}{60}\overline{u_{i-1}} + \frac{83}{60}\overline{u_i}\right\} + \sqrt{\frac{3}{5}}\left\{\frac{3}{16}\overline{u_{i-2}} - \frac{5}{8}\overline{u_{i-1}} + \frac{7}{16}\overline{u_i}\right\} \tag{2.59}$$

(d) **Limiting coefficient equation**

$$IS_1 = \frac{13}{12}\left\{\overline{u_{i-2}} - 2\overline{u_{i-1}} + \overline{u_i}\right\}^2 + \frac{1}{4}\left\{\overline{u_{i-2}} - 4\overline{u_{i-1}} + 3\overline{u_i}\right\}^2 \tag{2.60}$$

The limiting coefficients, used to define the limiting weights are exactly similar in form to the one used in classical fifth-order WENO schemes, (refer : eq. 2.28) and represented for this stencil by eq. 2.60

2. **Center stencil interpolations**

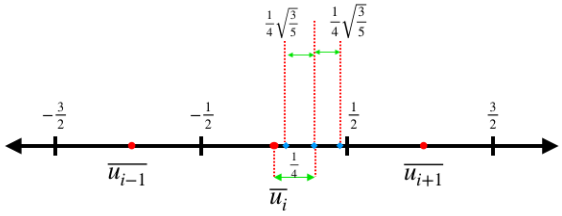


Figure 2.19: Center stencil for interpolation to quadrature point 1.

A similar primitive function $P(x)$ is defined for the center stencil and all the steps of the left stencil are followed

$$\begin{aligned} P(x) &= Ax^3 + Bx^2 + Cx + D \\ P\Big(-\frac{3}{2}\Big) &= 0 \\ P(-\frac{1}{2}) &= \overline{u_{i-1}} \\ P(\frac{1}{2}) &= \overline{u_{i-1}} + \overline{u_i} \\ P(\frac{3}{2}) &= \overline{u_{i-1}} + \overline{u_i} + \overline{u_{i+1}} \end{aligned} \tag{2.61}$$

$$\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} -1/6 & 1/2 & -1/2 & 1/6 \\ 1/4 & -1/4 & -1/4 & 1/4 \\ 1/24 & -9/8 & 9/8 & -1/24 \\ -1/16 & 9/16 & 9/16 & -1/16 \end{bmatrix} \begin{bmatrix} 0 \\ \overline{u_{i-1}} \\ \overline{u_{i-1}} + \overline{u_i} \\ \overline{u_{i-1}} + \overline{u_i} + \overline{u_{i+1}} \end{bmatrix} \tag{2.62}$$

(a) **Interpolation at point $\mathbf{x_{q1} = \frac{1}{4} - \frac{1}{4}\sqrt{\frac{3}{5}}}$**

$$u(x_{q1})^{S2} = P^{'}(x_{q1}) = 3Ax_{q1}^2 + 2Bx_{q1} + C = \frac{3}{10}A + \frac{1}{2}B + C - \sqrt{\frac{3}{5}}\left\{\frac{3}{8}A + \frac{1}{2}B\right\}$$

$$u(x_{q1})^{S2} = \left\{-\frac{7}{60}\overline{u_{i-1}} + \frac{59}{60}\overline{u_i} + \frac{2}{15}\overline{u_{i+1}}\right\} - \sqrt{\frac{3}{5}}\left\{-\frac{1}{16}\overline{u_{i-1}} - \frac{1}{8}\overline{u_i} + \frac{3}{16}\overline{u_{i+1}}\right\} \quad (2.63)$$

(b) **Interpolation at point $\mathbf{x_{q2} = \frac{1}{4}}$**

$$u(x_{q2})^{S2} = P^{'}(x_{q2}) = 3Ax_{q2}^2 + 2Bx_{q2} + C = \frac{3}{16}A + \frac{1}{2}B + C$$

$$u(x_{q1})^{S2} = -\frac{13}{96}\overline{u_{i-1}} + \frac{49}{48}\overline{u_i} + \frac{11}{96}\overline{u_{i+1}} \quad (2.64)$$

(c) **Interpolation at point $\mathbf{x_{q3} = \frac{1}{4} + \frac{1}{4}\sqrt{\frac{3}{5}}}$**

$$u(x_{q3})^{S2} = P^{'}(x_{q3}) = 3Ax_{q3}^2 + 2Bx_{q3} + C = \frac{3}{10}A + \frac{1}{2}B + C + \sqrt{\frac{3}{5}}\left\{\frac{3}{8}A + \frac{1}{2}B\right\}$$

$$u(x_{q3})^{S2} = \left\{-\frac{7}{60}\overline{u_{i-1}} + \frac{59}{60}\overline{u_i} + \frac{2}{15}\overline{u_{i+1}}\right\} + \sqrt{\frac{3}{5}}\left\{-\frac{1}{16}\overline{u_{i-1}} - \frac{1}{8}\overline{u_i} + \frac{3}{16}\overline{u_{i+1}}\right\} \quad (2.65)$$

(d) **Limiting coefficient equation**

$$IS_2 = \frac{13}{12}\left\{\overline{u_{i-1}} - 2\overline{u_i} + \overline{u_{i+1}}\right\}^2 + \frac{1}{4}\left\{\overline{u_{i-1}} - \overline{u_{i+1}}\right\}^2 \quad (2.66)$$

3. **Right stencil interpolations**

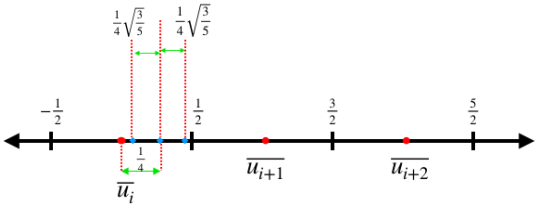


Figure 2.20: Right stencil for interpolation to quadrature point 1.

A similar primitive function $P(x)$ is defined for the right stencil and all the steps of the left stencil are followed

$$
\begin{aligned}
P(x) &= Ax^3 + Bx^2 + Cx + D \\
P\Big(-\frac{1}{2}\Big) &= 0 \\
P(\frac{1}{2}) &= \overline{u_i} \\
P(\frac{3}{2}) &= \overline{u_i} + \overline{u_{i+1}} \\
P(\frac{5}{2}) &= \overline{u_i} + \overline{u_{i+1}} + \overline{u_{i+2}}
\end{aligned}
\tag{2.67}
$$

$$
\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} -1/6 & 1/2 & -1/2 & 1/6 \\ 3/4 & -7/4 & 5/4 & -1/4 \\ -23/24 & 7/8 & 1/8 & -1/24 \\ 5/16 & 15/16 & -5/16 & 1/16 \end{bmatrix} \begin{bmatrix} 0 \\ \overline{u_i} \\ \overline{u_i} + \overline{u_{i+1}} \\ \overline{u_i} + \overline{u_{i+1}} + \overline{u_{i+2}} \end{bmatrix} \tag{2.68}
$$

(a) **Interpolation at point $\mathbf{x_{q1}} = \frac{1}{4} - \frac{1}{4}\sqrt{\frac{3}{5}}$**

$$
u(x_{q1})^{S3} = P^{'}(x_{q1}) = 3Ax_{q1}^2 + 2Bx_{q1} + C = \frac{3}{10}A + \frac{1}{2}B + C - \sqrt{\frac{3}{5}}\left\{\frac{3}{8}A + \frac{1}{2}B\right\}
$$

$$
u(x_{q1})^{S3} = \left\{\frac{19}{30}\overline{u_i} + \frac{29}{60}\overline{u_{i+1}} - \frac{7}{60}\overline{u_{i+2}}\right\} - \sqrt{\frac{3}{5}}\left\{-\frac{5}{16}\overline{u_i} + \frac{3}{8}\overline{u_{i+1}} - \frac{1}{16}\overline{u_{i+2}}\right\} \tag{2.69}
$$

(b) **Interpolation at point $\mathbf{x_{q2}} = \frac{1}{4}$**

$$
u(x_{q2})^{S3} = P^{'}(x_{q2}) = 3Ax_{q2}^2 + 2Bx_{q2} + C = \frac{3}{16}A + \frac{1}{2}B + C
$$

$$
u(x_{q2})^{S3} = \frac{59}{96}\overline{u_i} + \frac{25}{48}\overline{u_{i+1}} - \frac{13}{96}\overline{u_{i+2}} \tag{2.70}
$$

(c) **Interpolation at point $\mathbf{x_{q3}} = \frac{1}{4} + \frac{1}{4}\sqrt{\frac{3}{5}}$**

$$
u(x_{q3})^{S3} = P^{'}(x_{q3}) = 3Ax_{q3}^2 + 2Bx_{q3} + C = \frac{3}{10}A + \frac{1}{2}B + C + \sqrt{\frac{3}{5}}\left\{\frac{3}{8}A + \frac{1}{2}B\right\}
$$

$$
u(x_{q3})^{S3} = \left\{\frac{19}{30}\overline{u_i} + \frac{29}{60}\overline{u_{i+1}} - \frac{7}{60}\overline{u_{i+2}}\right\} + \sqrt{\frac{3}{5}}\left\{-\frac{5}{16}\overline{u_i} + \frac{3}{8}\overline{u_{i+1}} - \frac{1}{16}\overline{u_{i+2}}\right\} \tag{2.71}
$$

(d) **Limiting coefficient equation**

$$
IS_3 = \frac{13}{12}\left\{\overline{u_i} - 2\overline{u_{i+1}} + \overline{u_{i+2}}\right\}^2 + \frac{1}{4}\left\{3\overline{u_i} - 4\overline{u_{i+1}} + \overline{u_{i+2}}\right\}^2 \tag{2.72}
$$

Full stencil interpolations - Finding γ values

A fifth-order primitive function $P(x)$ is defined for the full stencil as depicted in fig. 2.17

$$\begin{aligned} P(x) &= Ax^5 + Bx^4 + Cx^3 + Dx^2 + Ex + F \\ P\Big(-\frac{5}{2}\Big) &= 0 \\ P\Big(-\frac{3}{2}\Big) &= \overline{u_{i-2}} \\ P\Big(-\frac{1}{2}\Big) &= \overline{u_{i-2}} + \overline{u_{i-1}} \\ P\Big(\frac{1}{2}\Big) &= \overline{u_{i-2}} + \overline{u_{i-1}} + \overline{u_i} \\ P\Big(\frac{3}{2}\Big) &= \overline{u_{i-2}} + \overline{u_{i-1}} + \overline{u_i} + \overline{u_{i+1}} \\ P\Big(\frac{5}{2}\Big) &= \overline{u_{i-2}} + \overline{u_{i-1}} + \overline{u_i} + \overline{u_{i+1}} + \overline{u_{i+2}} \end{aligned} \tag{2.73}$$

$$\begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \end{bmatrix} = \begin{bmatrix} -1/120 & 1/24 & -1/12 & 1/12 & -1/24 & 1/120 \\ 1/48 & -1/16 & 1/24 & 1/24 & -1/16 & 1/48 \\ 1/48 & -13/48 & 17/24 & -17/24 & 13/48 & -1/48 \\ -5/96 & 13/32 & -17/48 & -17/48 & 13/32 & -5/96 \\ -3/640 & 25/384 & -75/64 & 75/64 & -25/384 & 3/640 \\ 3/256 & -25/256 & 75/128 & 75/128 & -25/256 & 3/256 \end{bmatrix} \times \begin{bmatrix} 0 \\ \overline{u_{i-2}} \\ \overline{u_{i-2}} + \overline{u_{i-1}} \\ \overline{u_{i-2}} + \overline{u_{i-1}} + \overline{u_i} \\ \overline{u_{i-2}} + \overline{u_{i-1}} + \overline{u_i} + \overline{u_{i+1}} \\ \overline{u_{i-2}} + \overline{u_{i-1}} + \overline{u_i} + \overline{u_{i+1}} + \overline{u_{i+2}} \end{bmatrix} \tag{2.74}$$

The full stencil interpolation at any one of the quadrature points can be written as a sum of individual stencil contributions to that quadrature point using eq. 2.75.

$$u(x_{q1})^{S^0} = \gamma_{q1}^{S^1} u(x_{q1})^{S^1} + \gamma_{q1}^{S^2} u(x_{q1})^{S^2} + \gamma_{q1}^{S^3} u(x_{q1})^{S^3} \tag{2.75}$$

Hence, to compute the corresponding γ values for a quadrature point we need to compare the full five point stencil interpolation with the three individual three point stencil formulations. I will carry out the exercise for Quadrature point 2 and just write the results for the quadrature points 1 & 3.

Interpolation at point $\mathbf{x_{q2} = \frac{1}{4}}$ using five point stencil

$$u(x_{q2})^{S0} = P^{'}(x_{q2}) = 5Ax_{q2}^4 + 4Bx_{q2}^3 + 3Cx_{q2}^2 + 2Dx_{q2} + E = \frac{5}{256}A + \frac{1}{16}B + \frac{3}{16}C + \frac{1}{2}D + E$$

$$u(x_{q2})^{S0} = \frac{263}{10240}\overline{u_{i-2}} - \frac{483}{2560}\overline{u_{i-1}} + \frac{15767}{15360}\overline{u_i} + \frac{1231}{7680}\overline{u_{i+1}} - \frac{731}{30720}\overline{u_{i+2}} \tag{2.76}$$

The next step is computing γ_{q2} values by feeding eq. 2.76, 2.58, 2.64 & 2.70 into eq. 2.75. Comparing the coefficients of $\overline{u_{i-2}}$ in the LHS and RHS of eq. 2.75, we get eq. 2.77.

$$\frac{11}{96} \times \gamma_{q2}^{S^1} = \frac{263}{10240}$$
$$\rightarrow \gamma_{q2}^{S^1} = \frac{789}{3520} \tag{2.77}$$

Similarly, we can write all the γ coefficients for x_{q2}. They have been listed in eq. 2.78

$$\{\gamma_{q2}^{S^1}, \gamma_{q2}^{S^2}, \gamma_{q2}^{S^3}\} = \left\{ \frac{789}{3520}, \frac{13731}{22880}, \frac{731}{4160} \right\} \tag{2.78}$$

Handling negative γ values

Following a similar procedure the γ values for the other two quadrature points can be evaluated. They are listed in eq. 2.79 & 2.80.

$$\{\gamma_{q1}^{S^1}, \gamma_{q1}^{S^2}, \gamma_{q1}^{S^3}\} = \left\{ \; -0.869020 \;\; , \;\; 1.849014 \;\; , \;\; 0.020006 \; \right\} \tag{2.79}$$

$$\{\gamma_{q3}^{S^1}, \gamma_{q3}^{S^2}, \gamma_{q3}^{S^3}\} = \left\{ \; 0.118235 \;\; , \;\; 0.609552 \;\; , \;\; 0.272212 \; \right\} \tag{2.80}$$

We notice that there are negative γ values for interpolation to quadrature point 1, and here we use the splitting scheme as introduced by Shi et al. (2002), and already described in eq. 2.47, using which we obtain the split weights, which are given in eq. 2.81

$$\begin{aligned} \{\gamma_{q1+}^{S^1}, \gamma_{q1+}^{S^2}, \gamma_{q1+}^{S^3}\} &= \left\{ \; 0.188627 \;\; , \;\; 0.802687 \;\; , \;\; 0.0086850 \; \right\} \\ \{\gamma_{q1-}^{S^1}, \gamma_{q1-}^{S^2}, \gamma_{q1-}^{S^3}\} &= \left\{ \; 0.481844 \;\; , \;\; 0.512609 \;\; , \;\; 0.0055463 \; \right\} \end{aligned} \tag{2.81}$$

$$\sigma^+ = 4.60706 \quad \& \quad \sigma^- = 3.60706$$

For a one-dimensional case, using eq. 2.55 to 2.81, a fifth-order interpolation on all three quadrature points can be built by utilizing the WENO formulations given by either eq. 2.26 for continuous functions (Non-Limited-Prolongation - $\mathcal{O}(5)$), or by eq. 2.27 for discontinuous functions (limited-prolongation - $\mathcal{O}(3)$). Finally, the fine cell line-average value can be computed by a simple application of the three-point Gaussian quadrature formulation given in eq. 2.82.

$$\overline{u_{fine}} = \frac{5}{18} \times u_{q1} + \frac{8}{18} \times u_{q2} + \frac{5}{18} \times u_{q3} \tag{2.82}$$

Having described the $\mathcal{O}(5)$ prolongation function in 1D, I next layout the generalized formulation for higher dimension in the next subsection.

2D $\mathcal{O}(5)$ prolongation function

The idea here is to build surface-averages (volume-averages for 3D problems) in the fine cells from the surface-averages in the coarse cells. In the fig. 2.21 there is the coarse cell representation on the left, the four child cells in the middle out of which we are looking for cell-average values in the hatched cell. On the rightmost part of the fig. are the nine points marked in red which are chosen to accomplish this computation. These 9 points are the respective Gaussian-quadrature points in 2D. The idea is to reconstruct the point values at these 9 points from the surface averages of the coarse cells. Once that is obtained a Gaussian quadrature sum can be carried out to obtain the fine cell surface average. The exercise is repeated for the other fine cells as well. In 3D, the process involves computation at 27 quadrature points.

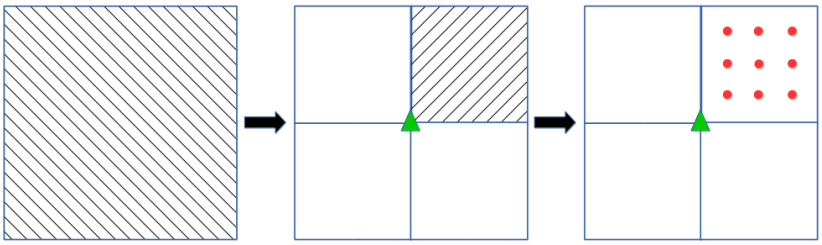


Figure 2.21: Schematic figure illustrating the procedure for prolongation - (a).

A 1D reconstruction-sweep reduces the order of dimension of the average by 1, which means a surface-average polynomial reconstructed once would yield a line average, while a volume average would yield a surface-average. Hence for a 2D case it requires 2 sweeps to get to point values, whereas for 3D cases it requires 3 sweeps (one in each distinct direction) to compute point values from volume averages. We start with an x-directional sweep, to get three line averages per fine-cell at the quadrature x-locations (shown in fig. 2.22).

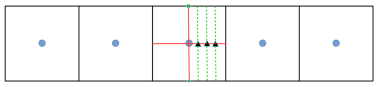


Figure 2.22: Prolongation (b).

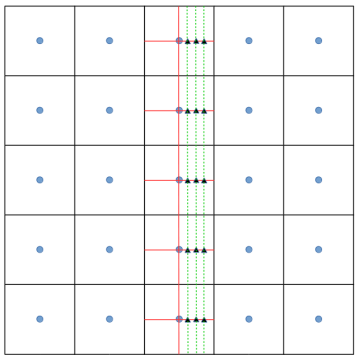


Figure 2.23: Prolongation (c).

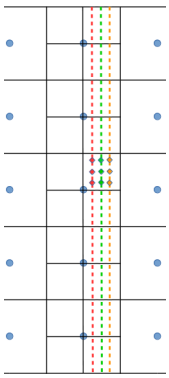


Figure 2.24: Prolongation (d).

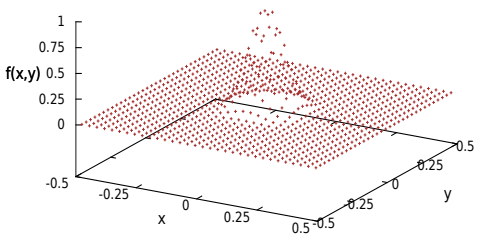
Then the same set of interpolation equations are again run at the location of these three quadrature lines in the transverse direction (using five transverse points for each of the three quadrature points, thus using 15 line-averages), as shown in fig. 2.23, to interpolate the quadrature point-values at nine points (marked in colored dots in fig. 2.24) . These nine point values are now used to build the surface average in the fine cell. For a 3D case, there is a requirement of three interpolation sweeps, and finally we end up with point values at 27 quadrature points, which are subsequently used to build the volume average in the fine cell.

2.10.6 Testing the order of the prolongation operator

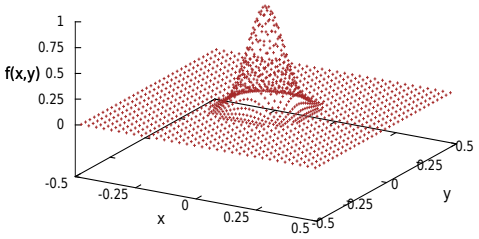
We begin by testing the prolongation operator on a 2D continuous function. For this test case, we take a two dimensional domain, defined as $x \subset [-0.5 : 0.5]$ and $y \subset [-0.5 : 0.5]$. The domain is then split using a uniform grid density. A scalar field is then initialized with the function given in eq. 2.83.

$$f(x) = \begin{cases} (1-(x/0.25)^2)^7 \times (1-(y/0.25)^2)^7, & \text{if } -0.25 \leq x \leq 0.25 \ \ \& \\ & \quad -0.25 \leq y \leq 0.25 \\ 0, & \text{otherwise} \end{cases} \tag{2.83}$$

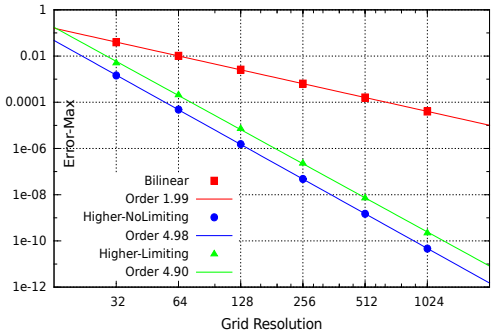
The grid is then refined by one level inside a circular patch of radius 0.25 with center at (0,0). The fifth-order prolongation function is called to fill in the refined cell values, and these interpolated values are then matched with the exact analytical values. The errors are noted and the exercise is repeated with different initial grid densities. Subsequently the error convergence is plotted to reveal the numerical order of the scheme, which demonstrates the functionality of the improved prolongation operator. This exercise is carried out for the bilinear prolongation, the $\mathcal{O}(5)$ limited prolongation, and the $\mathcal{O}(5)$ non-limited prolongation. The fig. 2.25c & 2.25d shows the comparative performance of the different prolongation schemes.



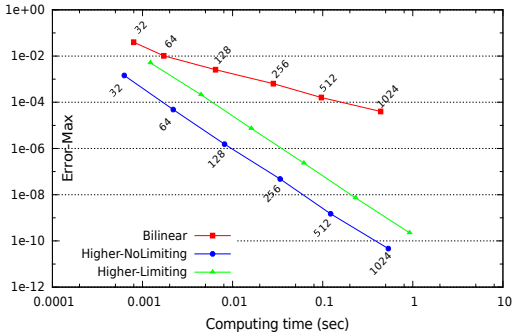
(a) Before refinement.



(b) After refinement.



(c) Error convergence.



(d) Computing Time.

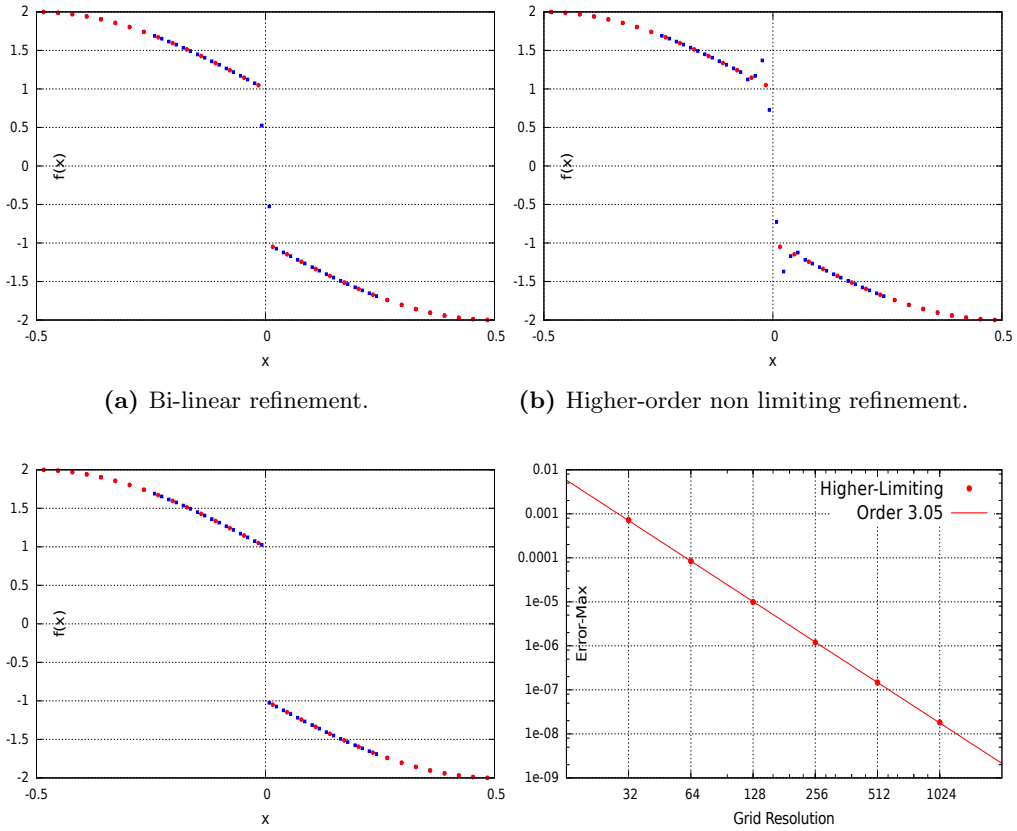
Figure 2.25: Prolongation function (Bilinear vs. Higher order).

We obtain a $\mathcal{O}(5)$ convergence for both the limited and the non-limited higher-order schemes. Of course, the limited version will be more computationally expensive, since it needs to compute the smoothness indicators and weight functions. Hence, while solving a problem which has only smooth and continuously differentiable field variables, it is advised to use the non-limited version of the Basilisk code, to save on computational resource.

Prolongation of a field with a discontinuity

In this section, we highlight the importance of the limited adaptive scheme, by looking at the performance of the prolongation function when there is a field discontinuity. For this test case, we take a simple 1D polynomial function as expressed in eq. 2.84. The grid is refined by one level in the sub-domain : $-0.25 \leq x \leq 0.25$. The refinement region includes the field discontinuity. The prolongated fine cell values (marked by blue dots) along with the coarse cell values (marked by red dots) are plotted in fig. 2.26a , 2.26b & 2.26c for the bi-linear, higher-order unlimited and the higher order limited prolongation schemes respectively.

$$f(x) = \begin{cases} 1 - sin(\pi x), & \text{if } -0.5 \leq x < 0 \\ -1 - sin(\pi x), & \text{if } 0 \leq x \leq 0.5 \end{cases} \tag{2.84}$$



(a) Bi-linear refinement.

(b) Higher-order non limiting refinement.

(c) Higher-order limiting refinement.

(d) Error convergence (limiting version).

Figure 2.26: Prolongation of a discontinuous field (Bilinear vs. Higher-order non-limited vs. Higher-order Limited scheme).

The bilinear prolongation introduces numerical dissipation close to the field discontinuity, while the higher-order method without limiting would lead to numerical oscillations thus leading to erroneous computations of the wavelet difference function. The higher-order limited prolongation function, does not introduce such oscillations, precisely because it is equipped with limiting functions. It is also convergent to an $\mathcal{O}(3)$ (fig. 2.26d). The fall in convergence-order is precisely what we expect of a three stencil limiting scheme while solving for discontinuous fields.

2.11 Advection of a tracer under rotation and stretching

This test case, pushes the advection solver to its maximum limits. We have a square domain defined by $(x, y) \subset [-0.5 : 0.5] \times [-0.5 : 0.5]$. We have a passive tracer field which is kind of a compact bump function located at a particular offset from the center as the initial condition. The initial tracer field is given by eq. 2.85 and displayed in fig. 2.27.

$$Tr(x,y) = \begin{cases} \left\{1 - \left(\frac{x+0.2}{0.15}\right)^2\right\}^7 \times \left\{1 - \left(\frac{y+0.236338}{0.15}\right)^2\right\}^7, & \text{if } (x+0.2)^2 \leq 0.0225 \text{ and} \\ & (y+0.236338)^2 \leq 0.0225 \\ 0, & \text{otherwise} \end{cases} \tag{2.85}$$

The time-dependent velocity field can be expressed using the set of eq. 2.86 and displayed in fig. 2.28 at $t = 1.25$.

$$\begin{aligned} u_x &= \frac{1.5}{\pi\Delta} sin\Big(\frac{2\pi t}{5}\Big) cos(\pi x)\Big\{cos\Big(\pi\Big(y + \frac{\Delta}{2}\Big)\Big) - cos\Big(\pi\Big(y - \frac{\Delta}{2}\Big)\Big)\Big\} \\ u_y &= -\frac{1.5}{\pi\Delta} sin\Big(\frac{2\pi t}{5}\Big) cos(\pi y)\Big\{cos\Big(\pi\Big(x + \frac{\Delta}{2}\Big)\Big) - cos\Big(\pi\Big(x - \frac{\Delta}{2}\Big)\Big)\Big\} \end{aligned} \tag{2.86}$$

The velocity field, which varies sinusoidally throughout the spatial domain has a reversible temporal component given by a sinusoidal function which has a time period of 5. All simulations are run at a CFL value of 0.8. The analytical solution should be the tracer rotating and stretching from t = 0 to 2.5, at which point the tracer will have maximum deformation (fig 2.29 & 2.30), and then in absence of physical viscosity in the system the tracer should return back to its exact original initial formulation at t = 5, which is when we end the simulation. We begin by solving the test case on uniform grids and will then move to the adaptive grid implementations.

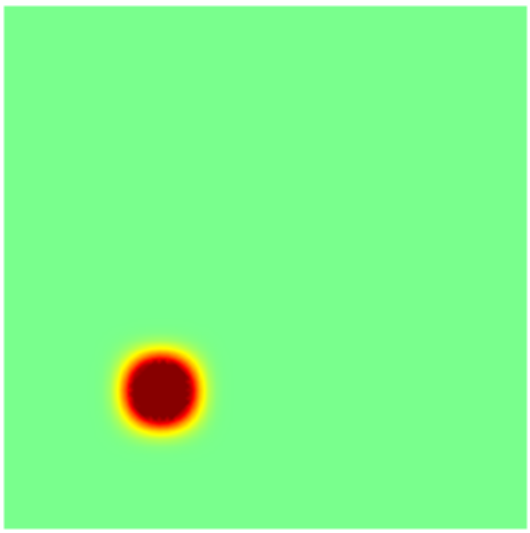


Figure 2.27: Tracer field at $t = 0$.

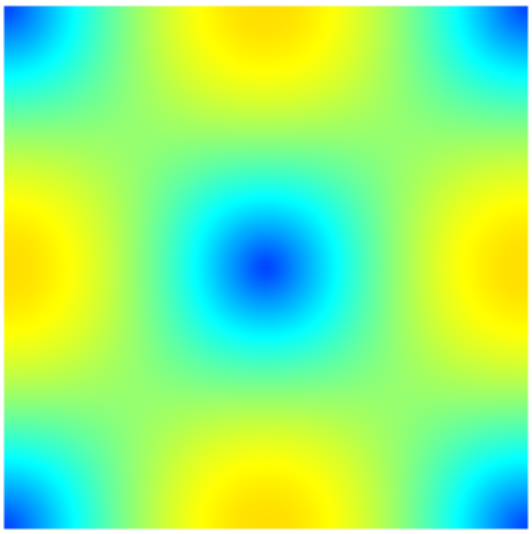


Figure 2.28: Velocity field at $t = 1.25$.

2.11.1 Uniform grid computations

This case is run on three uniform grid resolutions given by 128×128, 256×256 & 512×512, and the final solution tracer is compared to the initial condition, to compute the error on the domain. The figures 2.31 & 2.32 show the domain distribution of the error at the end of the simulation, for the BCG advection scheme and the WENO advection scheme respectively, for

a grid resolution of 512×512. The error signature immediately reveals the higher-order of the WENO schemes compared to the $\mathcal{O}(2)$ BCG scheme. Figure 2.33 plots the error norms vs the spatial resolutions for both the BCG and WENO schemes, which reveals the order of convergence of the new WENO-based scheme on a uniform grid to be $\mathcal{O}(4.25)$, while that of the BCG scheme is $\mathcal{O}(2.17)$.

Figure 2.29: *Tracer (BCG) at $t = 2.5$ 512×512 grid points.*

Figure 2.30: *Tracer (WENO) at $t = 2.5$ 512×512 grid points.*

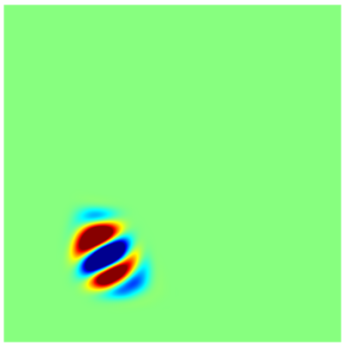


Figure 2.31: *Error field (BCG) at $t = 5$.*
RED(maximum) = 1.586e-02
BLUE(minimum) = -1.555e-02

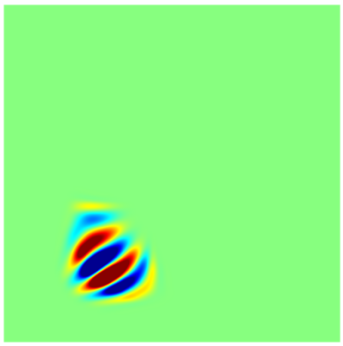


Figure 2.32: *Error field (WENO) at $t = 5$.*
RED(maximum) = 2.306e-04
BLUE(minimum) = -2.142e-04

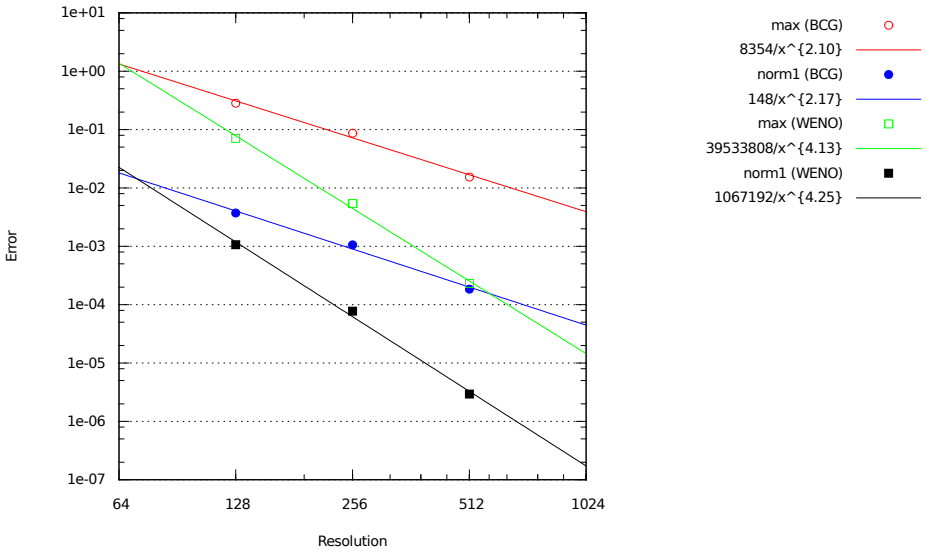
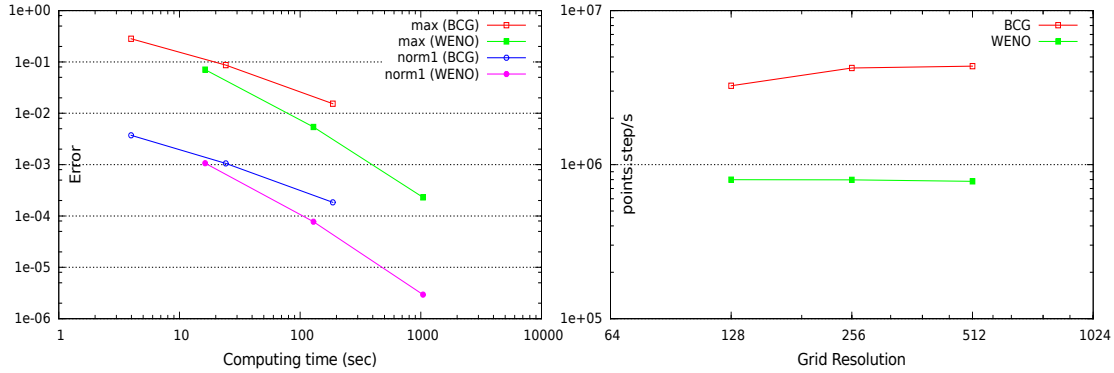


Figure 2.33: Error convergence of the advection solver on uniform grids (BCG vs WENO).

The computational cost has been cut down by the new WENO scheme compared to the existing BCG scheme, as shown in fig. 2.34a. The WENO graph-line is towards the left of the BCG graph-line, for all three spatial resolutions. For instance, to get to a L_1 error of 2.5×10^{-4}, the BCG advection scheme takes 186 seconds while the WENO advection scheme takes 68 seconds (an overall time speedup by a factor of 2.8), and the computational cost widens further for lower L_1 error values, as the two graph-lines diverge further.



(a) Error vs computing time. **(b)** Computing speed vs resolution.

Figure 2.34: 2D advection with rotation and stretching - BCG vs WENO.

The fig. 2.34b shows the computing cost expressed in terms of *points.steps/sec* plotted against the grid resolution. The cost saturates at 4.4×10^6 for the BCG scheme, while it saturates at 8×10^5 for the WENO scheme, making each WENO scheme timestep 5 times more expensive compared to the corresponding BCG scheme at equal grid resolutions.

2.11.2 Adaptive grid computation for the tracer advection problem

In this section we carry out simulations of the same advection problem on an adaptive grid. This is the most comprehensive test case, which uses fifth-order WENO formulations with Gaussian quadrature interpolations for 2D flux computations, a RK-4 time marching scheme and implementation of adaptive grids. The second-order BCG scheme is coupled with the bilinear prolongation operator for adaptivity.

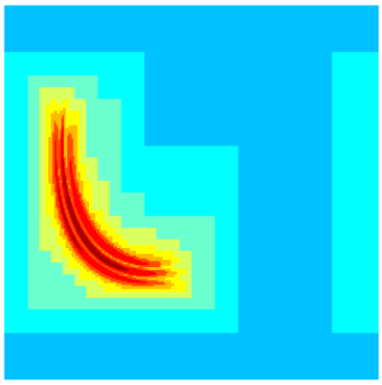


Figure 2.35: Grid-level (BCG), t = 2.5.
C_{max} =1.5625e-03
Max level = 11
Number of cells: 41925

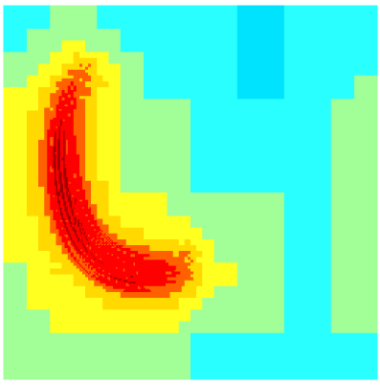


Figure 2.36: Grid-level (WENO), t = 2.5.
C_{max} =7.8125e-05
Max Level = 10
Number of cells: 23767

The grid levels at the moment of maximum tracer deformation, for the BCG scheme is plotted

in fig. 2.35 for a C_{max} value of 1.562e-03, while the grid levels for the WENO5-rk4 scheme is plotted in fig 2.36 for a C_{max} value of 7.812e-05. While the BCG case has 41925 grid cells at $t = 2.5$ sec, the WENO scheme has only 23767 grid cells, for a c_{max} value which is an order lower than the c_{max} value chosen for the BCG case. This gives a qualitative assessment that indeed the fifth-order adaptive scheme describes the evolving solution and hence the wavelet coefficient of the difference signal more precisely than the bilinear prolongation method. We will quantify this observation in the later part of this section. The tracer fields for the two schemes are shown in their position of maximum deformations viz. ($t = 2.5$), in figures 2.37 & 2.38, while an L1 error domain distribution plot at $t = 5$ is shown in figures 2.39 & 2.40. We can infer qualitatively from the error signatures, the higher-order nature of both the WENO scheme as well as the adaptive method.

Figure 2.37: *Tracer (BCG), t = 2.5.*
C_{max} =1.5625e-03

Figure 2.38: *Tracer (WENO), t = 2.5.*
C_{max} =7.8125e-05

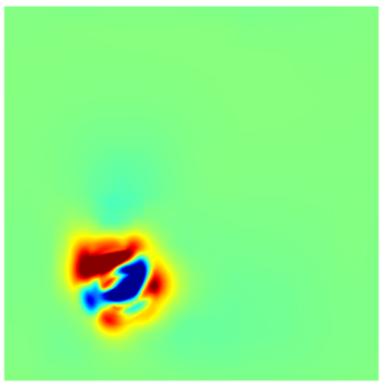


Figure 2.39: *L1 Error (BCG), t = 5.*
RED(maximum) = 4.178e-02
Blue(minimum) = -5.479e-02

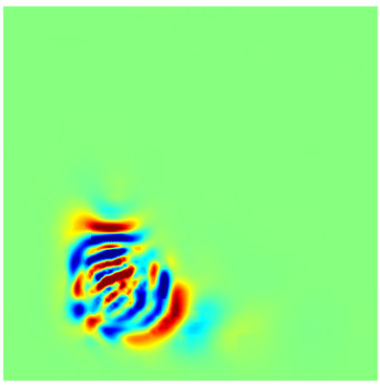


Figure 2.40: *L1 Error (WENO), t = 5.*
RED(maximum) = 9.284e-04
Blue(minimum) = -8.679e-04

In the context of adaptive grids it is important to note that we define an *equivalent resolution*. This is computed for a 2D problem like this one by taking the square root of the total number of grid-cells used at all time steps divided by the number of timesteps. It can be mathematically expressed using eq. 2.87. Figure 2.41 shows the range of C_{max} values over which the simulation has been carried out. The max level of the grid at the end of the simulation is plotted on the y-axis. The data points marked represent the equivalent resolution of each simulation.

$$Res_{eq} = \frac{1}{N}\sum_{n=1}^{N}\sqrt{Cells_n} \tag{2.87}$$

Figure 2.42 shows the error convergence of the solver. While the L_∞ norm for the BCG scheme converges at $\mathcal{O}(1.12)$, for the WENO scheme it converges at $\mathcal{O}(3.52)$.

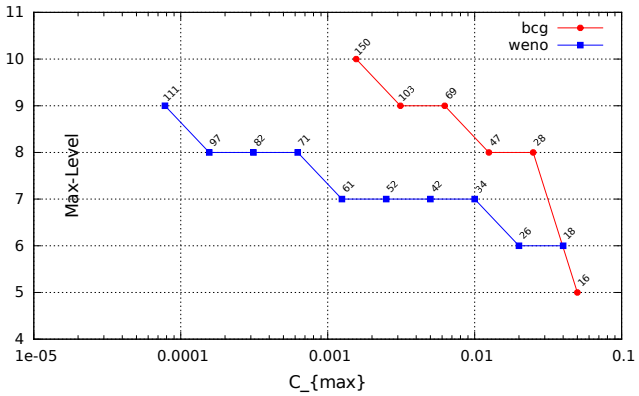


Figure 2.41: Plot of the maximum level at the end of the simulation vs C_{max}. Data points indicate the equivalent resolution of the simulation.

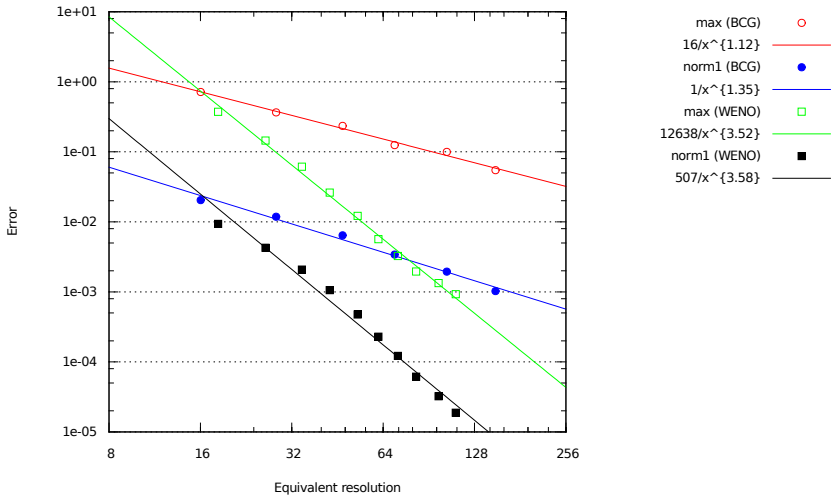
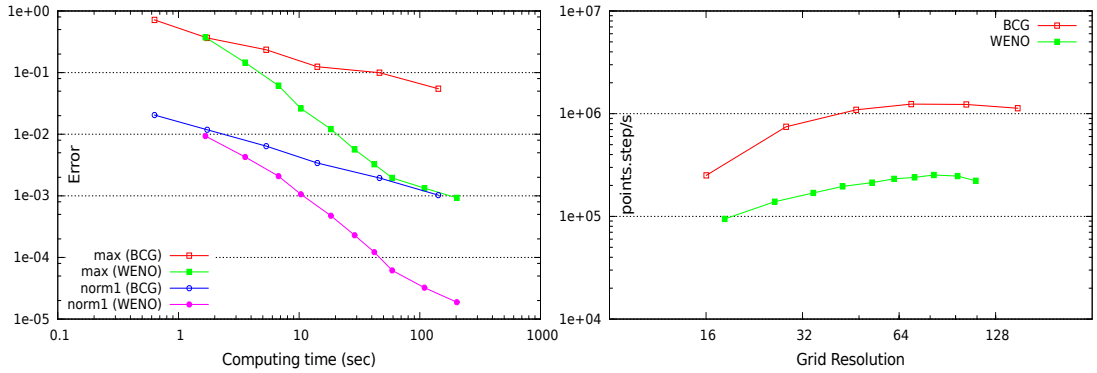


Figure 2.42: Error convergence of the advection solver on adaptive grids.



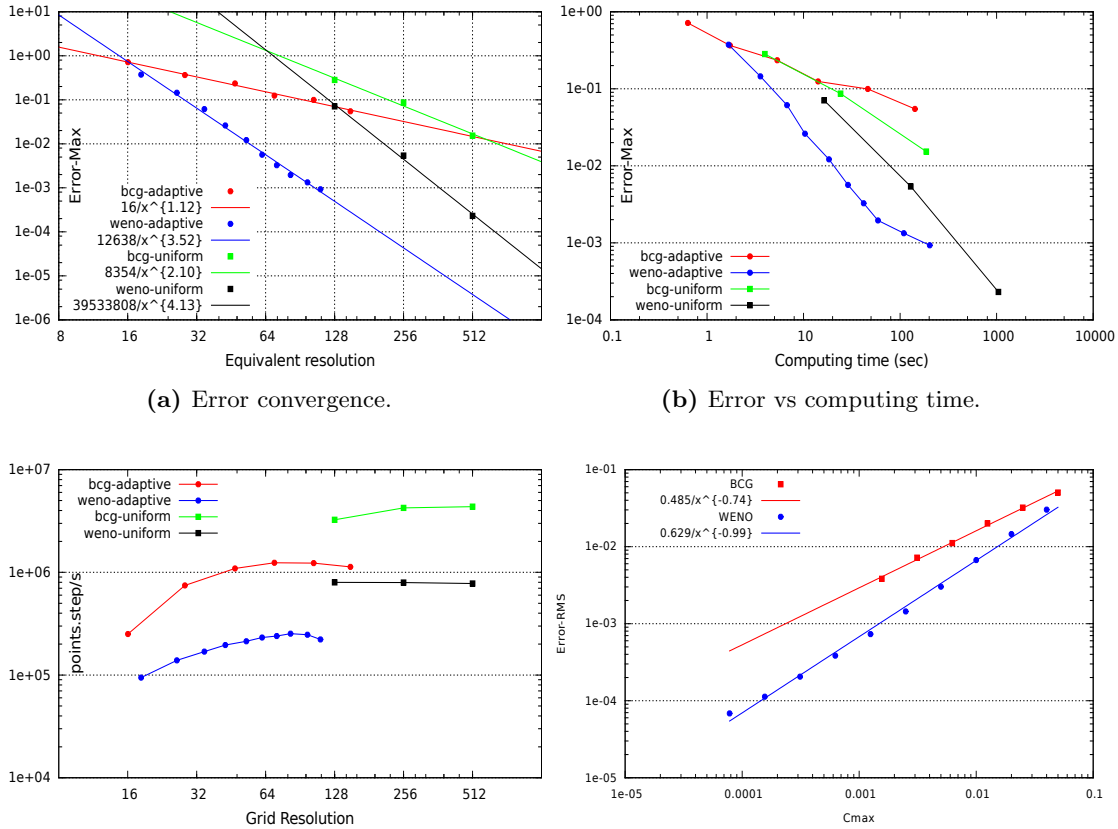
(a) Error vs computing time.

(b) Computing speed vs resolution.

Figure 2.43: Performance graphs - Adaptive grids : 2D advection with rotation and stretching - BCG vs WENO.

The performance metrics of the simulations are plotted in fig. 2.43a & 2.43b. We straightaway observe the superior performance of the new WENO based scheme. For instance, to get to a L1 error of 10^{-3}, the BCG advection scheme with bilinear prolongation takes 143 seconds while the WENO advection scheme takes only 10 seconds. Also the computational cost widens further for lower L1-error values, as the two graph-lines diverge further.

Once combined with a higher-order prolongation operator, the WENO scheme demonstrates a far better speedup relative to the BCG scheme, when compared to the uniform grid computations. This is illustrated through figures 2.44a, 2.44b & 2.44c where the uniform grid results have been juxtaposed over the adaptive grid results.



(a) Error convergence.

(b) Error vs computing time.

(c) Computing speed vs resolution.

(d) Error vs C_{max} on an adaptive grid.

Figure 2.44: Performance graphs - Adaptive grids : 2D advection with rotation and stretching - BCG vs WENO.

While the adaptive-grid order of convergence has come down compared to the uniform grid order for both the schemes as shown in fig. 2.44a, the important thing to note is the far greater leftward shift of the WENO adaptive plot from the WENO uniform-grid plot, when compared to the shift of BCG adaptive plot compared to the BCG uniform-grid plot. This signifies a drop in requirement of equivalent resolution required, for solving a problem to reach a given L_∞ error norm, and this is attributed to the performance of the adaptive algorithm. A far greater leftward shift (signifying a far greater drop in the equivalent resolution requirement), for the higher-order prolongation proves quantitatively its improved performance compared to the bilinear prolongation operator.

The same shift can be observed in fig. 2.44b for the WENO adaptive scheme but not for the BCG-adaptive scheme, which in fact shifts rightward to the uniform BCG scheme as the L_∞ error reduces for lower C_{max} values, signifying a better performance for the uniform grid computations. From the fig. 2.44c, we note that for the BCG-uniform-grid case the performance saturates at 4.3×10^6 points.steps/sec, while the BCG-adaptive-grid case the saturation occurs at 1.05×10^6 points.steps/sec, signifying each computational step is roughly 4 times faster for the uniform grid case compared to the adaptive grid case. For the WENO-uniform-grid case

the performance saturates at 7.8×10^5 points.steps/sec, while the WENO-adaptive-grid case the saturation occurs at 2.2×10^5 points.steps/sec, signifying each computational step is roughly 3.5 times faster for the uniform grid case compared to the adaptive grid case.

Finally, we look at the correlation between the error norm and the c_{max} value in fig. 2.44d. Ideally we expect a convergence of 1, when they are plotted on a log-log scale. We do observe the fifth-order scheme validating the same, whereas the bilinear prolongation operator falls short.

2.12 Conclusion

In conclusion, a fifth-order finite-volume WENO based advection scheme along with classical and total-variation-diminishing Runge–Kutta time marching schemes to solve the hyperbolic advection equations on a uniform grid have been succesfully implemented on Basilisk. Both these works have been coded in Basilisk using classical mathematical derivations from existing literature. Subsequently, a range of test cases carried out on 1D and 2D domains, using both continuous and discontinuous tracer fields, show that the uniform grid $\mathcal{O}(5)$ implementation of the WENO advection scheme outperforms the BCG based $\mathcal{O}(2)$ scheme, both in terms of error norms as well as computation time requiremnents.

One of the core achievements of this PhD has been to derive the algorithm for a novel $\mathcal{O}(5)$ prolongation function, and successfully implement it in Basilisk. Based on a limiting polynomial reconstruction methodology and Gaussian-quadrature sums, this algorithm can now be used on both smooth as well as non-smooth solutions to implement adaptivity. A complex test case involving a passive tracer field, being advected by a rotational and stretching advection current, demonstrates the superior performance of the novel higher-order adaptive method compared to the bi-linear prolongation method.

The WENO based advection solver will be used in chapter 4, for solving the convective acceleration terms in the Navier–Stokes equations, and also for computing the face-fluxes in an explicit solver for Saint–Venant equations described in chapter 5. The novel adaptive methodology will be applied for the Poisson–Hemholtz solver discussed in chapter 3, and can be theoretically applied to the Navier–Stokes or Saint-Venant solvers but it has not been done in this current work.

Chapter 3

Poisson–Helmholtz Solver

Contents

In this chapter we introduce two different higher-order schemes for solving the Poisson–Helmholtz equations. The algorithms are based on multigrid implementations and can be solved on adaptive quadtrees. The error convergence and performance studies are carried out and comparisons are made between the new higher-order solver and the existing classical $\mathcal{O}(2)$ solver on Basilisk.

3.1 Context

The Poisson equation is a partial differential equation of an elliptical nature, which has diverse applications ranging across many streams of sciences, engineering and economics. Morton (1996) lists ten sample applications which range from semiconductor simulation to financial modeling. In CFD, the Poisson equation finds application while solving the Navier–Stokes equations. It is used for estimating the viscous dissipation term as well as for projecting intermediate velocities to divergence free vector spaces.

$$\nabla^2\phi = \psi \tag{3.1}$$

$$\nabla.(\alpha\nabla\phi) + \lambda\phi = \psi \tag{3.2}$$

In its most basic form the Poisson equation can be expressed using eq. 3.1. A more generalized form of the Poisson equation is the Poisson–Helmholtz equation given by eq. 3.2. When the equation is numerically discretized we end up with a series of linear equations, and the subsequent coefficient matrix is large as well as sparse. While computing numerical solutions to an elliptic

problem, it is important to note, that these algorithms demand long CPU times and are memory intensive as well. Hence, the performance of a Poisson solver depends heavily on the choice of the linear system solver. As an example, while a solution method like Gaussian elimination has a computational complexity of $\mathcal{O}(n^3)$, algorithms like cyclic reduction & successive over-relaxation have a reduced complexity of $\mathcal{O}(n^{1.5})$, and then there exist other algorithms like FFT based methods which have a further lower complexity of $\mathcal{O}(nlog(n))$, and finally there are optimal multigrid method which can be computed with a complexity of $\mathcal{O}(n)$ (see Brandt (1977)). The majority of solvers for such linear-systems are built to find faster algorithms for these large sparse matrices (Saad and Schultz (1985) details out a number of such methods).

To obtain higher accuracy of the solution, one can either increase the grid density, thereby increasing the CPU time as well as the memory requirements or one can build a higher-order method, based on a wider stencil, which effectively widens the non-zero band of the sparse coefficient matrix. Different researchers, in the past, have built different versions of higher-order schemes for the Poisson equation.

Gupta et al. (1997) used a multigrid algorithm (a detailed review of multigrid algorithms can be found in this review paper by Fulton et al. (1986)) to solve the convection-diffusion equation using a nine-point compact finite-difference scheme, called as the Mehrstellenverfahren, which was in turn introduced by Collatz (1960). Barad and Colella (2005) used a finite-volume formulation of the classical Mehrstellen methods along with the block structured local refinement algorithm of Berger and Colella (1989) to obtain solutions to Poisson's equations which are fourth-order accurate. Later, Zhang et al. (2012), (using the ideas of connecting cell-averaged quantities to face-averaged quantities for higher order finite-volume discretizations of hyperbolic schemes given in Colella and Woodward (1984)), built a fourth-order finite-volume discretization scheme for solving the advection-diffusion equations using a multigrid algorithm on block-structured adaptively refined grids.

The approach in this work was to derive/look for a suitable higher-order discretization scheme for solving a finite-volume Poisson–Helmholtz equation, and compute it using a v-cycle multigrid algorithm on adaptive-quad/octrees (implemented using the higher order prolongation function developed in the last chapter). A higher-order method complicates individual stencil computations and boundary condition implementations. However, while each computing step becomes more expensive, the total number of steps required to reach a desired level of accuracy for the solution comes down. This provides a motivation to derive and implement a higher-order scheme for the Poisson–Helmholtz solver and to study its comparative computing performance with respect to the existing second-order scheme in Basilisk (described in the next section).

3.2 State of the Art: October 2015

The state of the art Poisson solver implementation on Basilisk at the start of my PhD was a second-order multigrid-based solver for the Poisson–Helmholtz equation. The multigrid implementation relied on an $\mathcal{O}(2)$ bi-linear prolongation function, and so did the adaptive mesh refinement algorithm. The work undertaken during the course of this thesis was aimed at improving the quality and the performance of this solver for solution on uniform & adaptive grids.

3.3 Numerical Algorithm – Poisson Solver

A generic numerical Poisson solver comprises of three different components, namely

1. **A Discretization scheme**: This step gives a numerical formulation to the elliptic partial differential equation. How we implement this step determines the effective order of accuracy of the solver.
2. **An Iterative Method**: An elliptic solver is very different from a hyperbolic solver, and here the usage of an iterative method becomes imperative. The iterative method is applied to invert a big and sparse matrix for finding the solution to the linear system of equations.
3. **Multigrid accelerator**: This is a numerical method which is very useful in problems which exhibit multiple scales of behavior. It accelerates a standard iterative algorithm through the usage of a hierarchy of discretizations.

I will be discussing in detail each of these different components of the solver in the following subsections. I will start with Iterative methods, then cover Multigrid methods and will finally discuss two different higher order discretization schemes.

3.3.1 Iterative Methods

In computational methods, an iterative method is a mathematical procedure that generates a sequence of improving approximate solutions for a class of problems, in which the n-th approximation is derived from the previous ones. A specific implementation of an iterative cycle, including the termination criteria, is an algorithm of an iterative method. An iterative method is called convergent, if the corresponding sequence converges for a given initial approximation. Such methods are largely used for non-linear problems, but they can be extended to linear problems involving a large number of variables (sometimes of the order of millions), where direct methods would be prohibitively expensive (and in some cases impossible) even with the best available computing power.

For our case we will use the well known Jacobi iteration method. This is essentially an algorithm for determining the solutions of a diagonally dominant system of linear equations. Suppose, we have to find a solution to the linear system given by eq. 3.3, where A and B are known and X needs to be computed.

$$AX = B \tag{3.3}$$

The first step is to start with an initial guess X_0, and then compute the residual of the linear system. The formulation for the residual is given using the eq. 3.4.

$$Res = B – AX_0 \tag{3.4}$$

Now, $X_{solution}$ can be thought of a $X_0 + dX$, (dX is the increment to the solution at each step). Given the linear nature of the problem, this can be cast into the new eq. 3.5.

$$AdX = Res \tag{3.5}$$

At this stage, A can be split into a diagonal component D and a remainder R. A couple of relaxation operations are done on dX using the formulation given in eq. 3.6.

$$dX^{k+1} = D^{-1}[Res - RdX^k] \tag{3.6}$$

The dX is then added to modify the initial solution, and the residual is re-computed. The cycle is repeated till the time the residual falls below an acceptable tolerance limit, indicating that the solution has converged. The detailed construction of the residual operator will be discussed under the subsection on discretization schemes.

3.3.2 Multigrid Methods

For a normal iterative solver, the errors which have a wavelength comparable to the grid spacing, decay at a very fast pace, while the errors which have larger wavelength decay asymptotically, thus requiring a larger number of iterations and a higher CPU time. Most fluid problems we encounter exhibit multiple scales of behavior and therefore this leaves room for improvement. The main idea behind multigrid solvers is to accelerate the convergence of a basic iterative method (known as relaxation, which generally reduces short wavelength error) by an overall correction of the fine grid solution approximation from time to time, accomplished by solving a coarse problem.

To accomplish this, instead of using one layer of grid, we use multiple layers of grid, with increasing grid spacings, and the solution is relaxed at each grid level, thus improving the solution increment by reducing errors of all wavelengths. The coarse problem is cheaper to solve, and that is how the computation is accelerated. The multigrid method can be split into three different steps, which denote the solution corrections at each step and the interaction between different grid levels. The steps are namely

- **Smoothing**: This step reduces the high frequency errors and is accomplished by using using a few iterations of the Jacobi method.
- **Restriction**: This step involves interpolating the error of the residual from fine grid levels to the coarser grid levels.
- **Prolongation**: Finally, this step involves interpolating a correction to the solution computed on a coarser grid into a finer grid.

The restriction and the prolongation methods in the context of the multigrids are the exact same methods that have been derived and used for adaptive grids in Chapter 2, section 2.10. At this stage, I will introduce another prolongation method that I worked on while building the Poisson solver. It is called the biquartic prolongation, and unlike the fifth-order volume-average-based solver described in chapter 2, this operator is a cell-centered point value based $\mathcal{O}(5)$ prolongation operator.

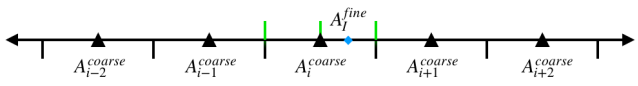


Figure 3.1: *Biquartic operator coarse and fine stencil.*

$$A_I^{fine} = \frac{105A_{i-2}^{coarse} - 756A_{i-1}^{coarse} + 5670A_i^{coarse} + 1260A_{i+1}^{coarse} - 135A_{i+2}^{coarse}}{24 \times 256} \tag{3.7}$$

The equation 3.7 provides the quartic prolongation operator in 1D. Its 2D and 3D extensions are built by extending this formulation. Figure 3.2 demonstrates the fifth-order convergence of the biquartic operator on a 2D periodic domain, where a sinusoidal function is prolongated by refining a circular patch inside the domain. The error is then computed by taking the difference of the prolongated values and the analytical values.

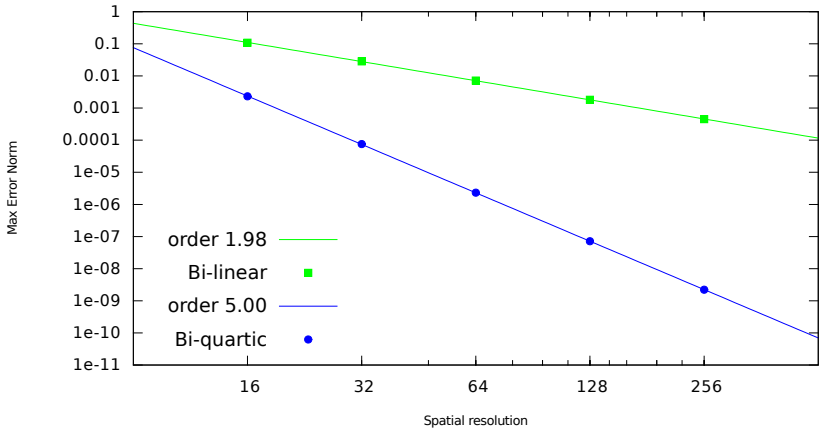


Figure 3.2: *Error convergence of a biquartic prolongation operator vs bilinear operator.*

Having described the multigrid methods and Jacobi iterative method, next I will be describing three different discretization schemes. The first will be the classical second-order solver, which was the state of the art Basilisk implementation, when I started my PhD. The other two are both fourth-order schemes, which were derived and implemented on Basilisk by me during the course of my PhD.

3.3.3 Discretization Scheme – Second-order solver

The choice of the discretization scheme determines the choice of the stencil for the scheme. The terminology stencil is used with respect to a discretization scheme, and it refers to the grid cells which are required to compute the Numerical operator which we are trying to discretize using our scheme. Fig. 3.3 shows the stencil required for computing the classical $\mathcal{O}(2)$ numerical formulation of the Laplacian operator in a 2D domain. We require five different cell values for this computation, and hence it can be referred to as the 5-point stencil.

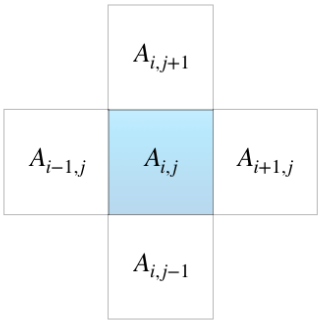


Figure 3.3: *A 2d Stencil for the Poisson 2^{nd} order solver.*

$$\nabla^2 A_{i,j} = \frac{A_{i+1,j} + A_{i-1,j} - 4A_{i,j} + A_{i,j+1} + A_{i,j-1}}{\delta^2} \tag{3.8}$$

The formulation for the corresponding classical Laplacian operator is provided in equation 3.8. It should be noted that for a second-order approximation, the volume-average of a field is equal to the cell centered value of that field, which is the reason that the finite volume scheme looks exactly like a finite difference scheme would look like. However, once we move to higher-order schemes, the same does not hold true.

3.3.4 Discretization Scheme – Fourth-order solver

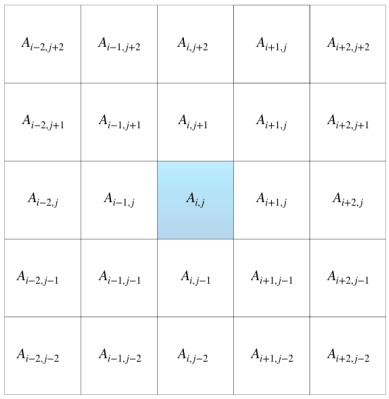
In this section, I will be illustrating two different fourth-order discretization schemes that I have derived for the Poisson–Helmholtz equation during the course of my PhD, and subsequently I will state which particular scheme fits our purposes accurately.

Discretization scheme 1 – The 25-point scheme

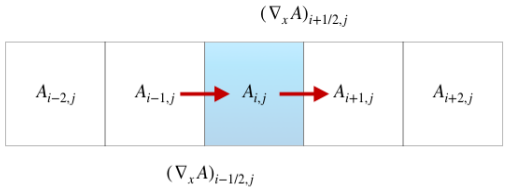
Given a finite volume cell, $(x, y) \subset [x_i - \frac{\Delta x}{2}, x_i + \frac{\Delta x}{2}] \times [y_j - \frac{\Delta y}{2}, y_j + \frac{\Delta y}{2}]$, this method, starts with cell-centered values of the function $A(x, y)$, represented as $A_{i,j}$ and defined at the location (x_i, y_j). The objective of this scheme is to build a numerical operator for the volume-averaged Laplacian defined by eq 3.9.

$$\overline{\nabla^2 A_{i,j}} = \int_{y_j-\Delta y/2}^{y_j+\Delta y/2} \int_{x_i-\Delta x/2}^{x_i+\Delta x/2} \left\{ \frac{\partial^2 A}{\partial x^2} + \frac{\partial^2 A}{\partial y^2} \right\} dxdy \tag{3.9}$$

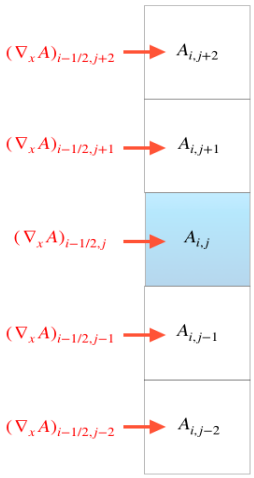
The stencil required for building this operator is a 25-point stencil as shown in figure 3.4a. The details of the computation are described subsequently.



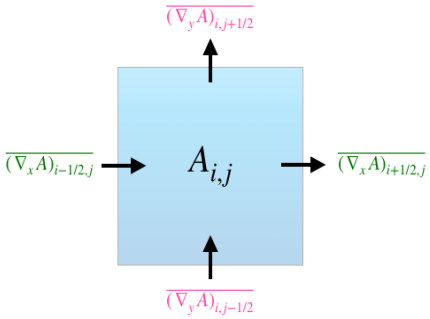
(a) A 2D 25-point stencil for a $\mathcal{O}(4)$ Poisson–Helmholtz solver.



(b) Stencil for computing $\mathcal{O}(4)$ face-centered gradient fields.



(c) Stencil for computing $\mathcal{O}(4)$ face-averaged gradient fields.



(d) Stencil for Computing $\mathcal{O}(4)$ surface-averaged Laplacian.

Figure 3.4: 25-point scheme – Stencils.

The first step is to construct gradients of $A(x, y)$ at all face centers. Since the $A(x, y)$ field data is stored as cell-centered values in the finite volume cells, a simple fourth-order polynomial can be constructed, and then evaluated, by fitting the values of the neighbouring cells. This is done in a dimension by dimension manner e.g. to compute the face-centered gradient at $(x_{i+1/2}, y_j)$, we start with a polynomial $P(x) = Ax^3 + Bx^2 + Cx + D$, and fit the values of $A_{i-1,j}$, $A_{i,j}$, $A_{i+1,j}$ and $A_{i+2,j}$ to the polynomial $P(x)$ to evaluate the coefficients. Once done, we compute the first derivative of the polynomial and find out its value at the face location $(x_{i+1/2}, y_j)$. Since, the values $A_{i,j}$ are point location values and not surface-averaged values, the gradients we end up with are also face centered gradients, and these values approximate the face-averaged gradients upto an $\mathcal{O}(2)$ accuracy. The stencil for this computation is shown in fig. 3.4b. The numerical formulation for these face centered gradients are given by eq. 3.10.

$$
\begin{aligned}
(\nabla_x A)_{i+1/2,j} &= \frac{A_{i-1,j} - 27A_{i,j} + 27A_{i+1,j} - A_{i+2,j}}{24\Delta} \\
(\nabla_y A)_{i,j+1/2} &= \frac{A_{i,j-1} - 27A_{i,j} + 27A_{i,j+1} - A_{i,j+2}}{24\Delta}
\end{aligned}
\tag{3.10}
$$

This process is repeated to evaluate the x-face-centered gradients at the faces $(x_{i-1/2}, y_{j+k})$ and $(x_{i+1/2}, y_{j+k})$, and the y-face-centered gradients at the faces $(x_{i+k}, y_{j-1/2})$ and $(x_{i+k}, y_{j+1/2})$ $\forall k \subset \{-2, -1, 0, 1, 2\}$. However, to compute a volume-averaged Laplacian, we need to compute the face-averaged gradients, and to compute the face-averaged gradients we need to take the transverse face-centered gradients as shown in figure 3.4c and using these values build a smooth $\mathcal{O}(5)$ polynomial distribution over the face $(x_{i-1/2}, y_j)$ and then use this polynomial distribution to compute the face-averaged gradient value at that face. This is done using eq. 3.11.

$$
\begin{aligned}
\overline{\nabla_x A}_{i+1/2,j} = \frac{1}{5760} \times \Big\{ & -17 \times (\nabla_x A)_{i+1/2,j-2} + 308 \times (\nabla_x A)_{i+1/2,j-1} \\
& + 5178 \times (\nabla_x A)_{i+1/2,j} + 308 \times (\nabla_x A)_{i+1/2,j+1} \\
& - 17 \times (\nabla_x A)_{i+1/2,j+2} \Big\} \\
\overline{\nabla_y A}_{i,j+1/2} = \frac{1}{5760} \times \Big\{ & -17 \times (\nabla_y A)_{i-2,j+1/2} + 308 \times (\nabla_x A)_{i-1,j+1/2} \\
& + 5178 \times (\nabla_x A)_{i,j+1/2} + 308 \times (\nabla_x A)_{i+1,j+1/2} \\
& - 17 \times (\nabla_x A)_{i+2,j+1/2} \Big\}
\end{aligned}
\tag{3.11}
$$

Once the face-averaged gradients have been computed using equation 3.11 over all four faces shown in fig. 3.4d, we use these values to compute the volume-averaged Laplacian by a simple application of the divergence theorem, as shown in equation 3.12.

$$
\overline{\nabla^2 A}_{i,j} = \frac{\overline{\nabla_x A}_{i+1/2,j} - \overline{\nabla_x A}_{i-1/2,j} + \overline{\nabla_y A}_{i,j+1/2} - \overline{\nabla_y A}_{i,j-1/2}}{\Delta} \tag{3.12}
$$

Using the expression for the numerical discretization of the Laplacian operator, the equivalent relaxation and residual functions can be constructed, for solving a Poisson problem. For instance, while solving the generalized Poisson–Helmholtz equation : $\nabla \cdot (\alpha \nabla \phi) + \lambda \phi = \psi$, the Residual at a particular iterative step N, is given by eq. 3.13, where L represents the numerical discretization of the Laplacian operator : $\nabla \cdot (\alpha \nabla \phi)$.

$$
Res^N_{i,j} = \psi^N_{i,j} - \lambda_{i,j}\phi^N_{i,j} - L^N_{i,j}(\alpha, \phi) \tag{3.13}
$$

The relaxation operator essentially uses the residual obtained at iterative step N, to predict a solution increment, so as to compute the solution for iterative step $N+1$. This is done by splitting the discrete Laplacian operator into a diagonal operator, operating on the stencil at iterative step $N+1$, and given by the expression $LD_{i,j}$, and a remainder operator, operating on the stencil at timestep N, and given by the expression $LR_{i,j}$. The numerical expression for the Relaxation step can be expressed using eq. 3.14

$$\begin{aligned} \Delta\phi^N_{i,j} &= [LD_{i,j}]^{-1} \times \{Res^N_{i,j} - LR_{i,j}(\alpha, \phi^N) \times \phi^N_{i,j}\} \\ \phi^{N+1} &= \phi^N + \Delta\phi^N \end{aligned} \tag{3.14}$$

Hence, in summary the algorithm for the inverse problem involves starting with a initial guess solution $\phi^0_{i,j}$, and using it to evaluate the residual at the finest grid level. This residual is then restricted to obtain the residuals at all grid levels, down to the coarsest level. Now, starting from the coarsest level, a number of relaxation operations are carried out to compute the solution increment $\Delta\phi_{i,j}$, and the improved solution $\phi^1_{i,j}$ at the coarsest level. This solution is interpolated to the next finer grid level using the prolongation function which happens to be the bi-quartic prolongation algorithm for this scheme. A number of relaxation steps are run to further improve the solution. These two steps of interpolation and relaxation are carried out till be obtain a solution at the finest grid level, which ends one iterative step of the algorithm.

Using the new fine grid solution at the end of iterative step 1, the residual is re-computed and its L_∞ norm is computed. If this norm is lower than an acceptable tolerance limit set by the user, the algorithm stops and gives the current $\phi_{i,j}$ as the final solution to the Inverse problem. If not, the iteration is repeated multiple times, till the step when the Residual norm falls within the acceptable tolerance limit.

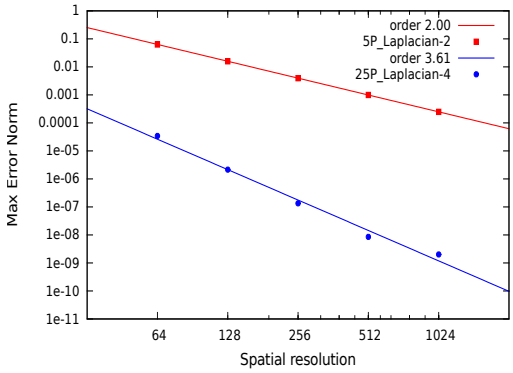
Having derived and implemented this 25-point $\mathcal{O}(4)$ scheme on Basilisk, to verify its functionality, I will be discussing test cases for both the Laplacian problem as well as the Poisson problem, in the next subsection.

Test case for the direct 25-point Laplacian problem – Uniform Grids

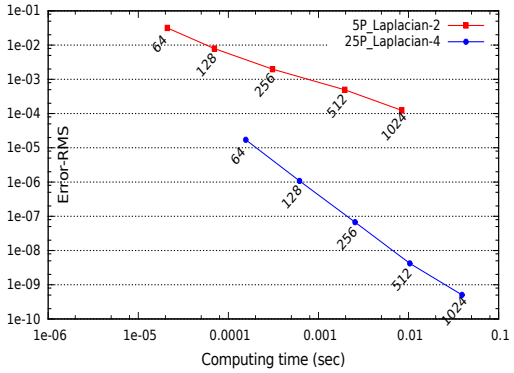
We begin with testing the 25-point scheme, using this direct Laplacian operator. We solve the system of eq. 3.15, to compute B numerically. and subsequently compute the error on the field. We then use the error norms to compute the order of the discretization and compare the result with the classical 2nd order scheme.

$$\begin{aligned} \nabla^2 A &= \overline{B} \\ A &= cos(2\pi x)cos(2\pi y) \\ B_{An} &= \frac{sin(2\pi(x+\frac{\Delta}{2})) - sin(2\pi(x-\frac{\Delta}{2}))}{\Delta} \times \frac{sin(2\pi(y+\frac{\Delta}{2})) - sin(2\pi(y-\frac{\Delta}{2}))}{\Delta} \\ Error(grid) &= B_{Num} - B_{An} \end{aligned} \tag{3.15}$$

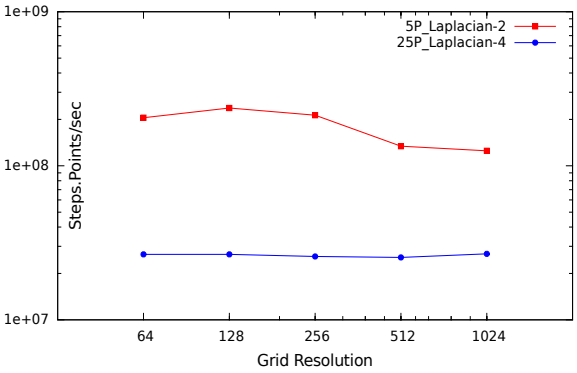
We observe an error convergence of $\mathcal{O}(3.61)$ for the 25-point solver as shown in fig. 3.5a. The computing time vs. error plot shows that to reach to an L_∞ error norm of 10^{-4}, the $\mathcal{O}(2)$ scheme has to use a 1024×1024 grid, and requires roughly 0.01 seconds, while the $\mathcal{O}(4)$ scheme reaches an Ł$_{infty}$ error norm of 2×10^{-5} using a 64×64 grid, and requires roughly 0.0002 seconds. This represents a speedup by 50 times for the 25-point solver.



(a) Error convergence vs resolution.



(b) Computing time vs resolution.



(c) Computing speed vs resolution.

Figure 3.5: Direct Laplacian problem – $\mathcal{O}(2)$ vs $\mathcal{O}(4)$ scheme.

The computing cost as plotted in fig. 3.5c, shows that the classical 5-point scheme is five times faster compared to the 25-point scheme. This is quite obvious given the fact that the 25-point stencil has to access five times the number of field variables compared to the 5-point scheme.

Test case for the 25-point Poisson problem – Uniform grids

The scheme is now applied to an inverse problem or the Poisson solver. Using the 25-point scheme for residual formulation and using the biquartic prolongation the solution is obtained for the equation system 3.16, where the $\overline{B}$ value is fed as an input and the A is computed numerically. While, the 25-point scheme uses the bi-quartic prolongation operator, the classical $\mathcal{O}(2)$ uses the bi-linear prolongation operator. A residual tolerance of 10^{-6} is set for the $\mathcal{O}(2)$ scheme whereas the $\mathcal{O}(4)$ scheme has the tolerance set at 10^{-9}.

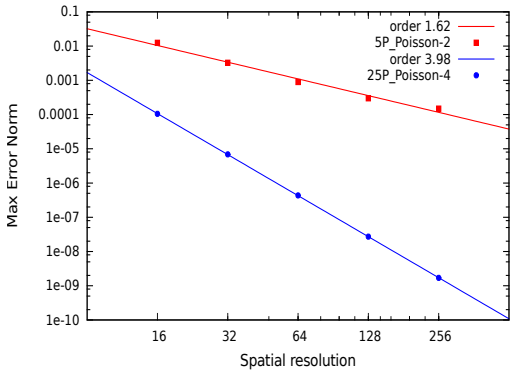
$$\nabla^2 A = \overline{B}$$

$$\overline{B} = \frac{sin(2\pi(x+\frac{\Delta}{2})) - sin(2\pi(x-\frac{\Delta}{2}))}{\Delta} \times \frac{sin(2\pi(y+\frac{\Delta}{2})) - sin(2\pi(y-\frac{\Delta}{2}))}{\Delta} \tag{3.16}$$

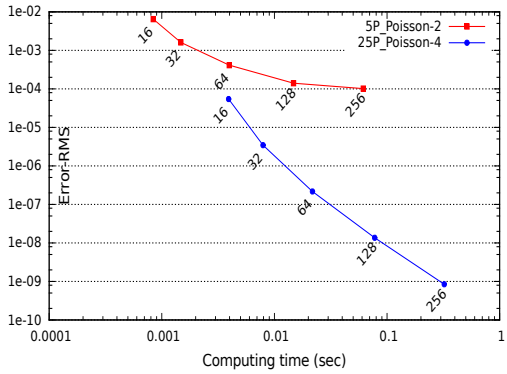
$$A_a n = cos(2\pi x) cos(2\pi y)$$

$$Error(grid) = A_{Num} - A_{An}$$

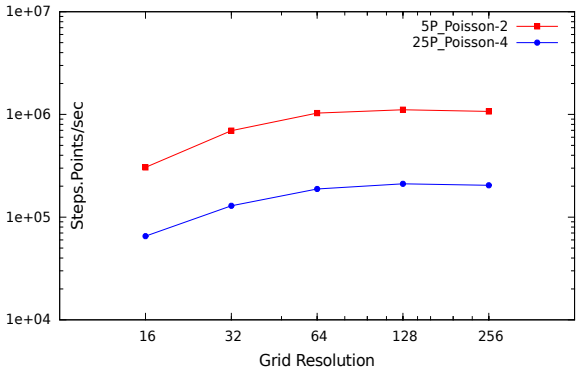
The error norm plotted in fig. 3.6a shows an error convergence of $\mathcal{O}(4)$ for the 25-point scheme. The error is plotted against the computing time in fig. 3.6b, and it is here that we observe the gradual saturation of the solution error norm for the $\mathcal{O}(2)$ scheme around error levels of 10^{-4},



(a) Error convergence vs resolution.



(b) Computing time vs resolution.



(c) Computing speed vs resolution.

Figure 3.6: Poisson problem - $\mathcal{O}(2)$ vs $\mathcal{O}(4)$ scheme

while the $\mathcal{O}(4)$ scheme has a lower error norm for a resolution of 16×16, compared to the $\mathcal{O}(2)$ scheme for a resolution of 256×256. The fig. 3.6c shows again that the computing time goes up by 5 times for the 25-point $\mathcal{O}(4)$ scheme compared to the $\mathcal{O}(2)$ scheme. Having obtained satisfactory results for the uniform grid test case, we turn our attention to a non-uniform grid case, so as to eventually have a full Poisson solver with adaptivity.

Test case for the direct Laplacian problem – Non-uniform grids

In this test case, we use a periodic domain, and solve the same problem that we solved for the Uniform grid direct problem given by eq. 3.15, but here we use a non-uniform / refined grid for studying this test case. The grid is defined by the equation 3.17, where the grid spacing is given by $\Delta = 1/2^{level(x,y)}$.

$$[level(x,y)]_N = \begin{cases} N+1 & \text{, if } \sqrt{x^2+y^2} \leq 0.25 \\ N & \text{, otherwise} \end{cases} \tag{3.17}$$

The error distribution profile shows sharp jumps in the neighborhood of the refinement boundary, where the error does not even converge as the resolution is increased (fig. 3.7). The error converges in regions away from the refinement boundaries (fig. 3.8a), however the overall error diverges (fig. 3.8b). This of course renders the biquartic prolongation scheme incompatible with the 25-point scheme and hence this becomes ineffective for our usage, since a major goal of this work is to solve fluid systems on adaptive grids and not simply on uniform grids.

Another major reason for discarding this scheme in favor of the one I am going to introduce in the next section, is that the scheme by its very design converts a point value to a volume-average Laplacian and vice versa for the inverse Poisson problem, and hence the scheme becomes undesirable for continued use in further chapters, as we intend to build all future solvers premised on

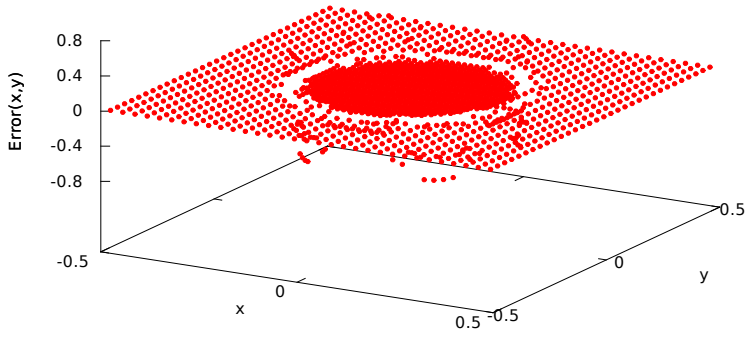
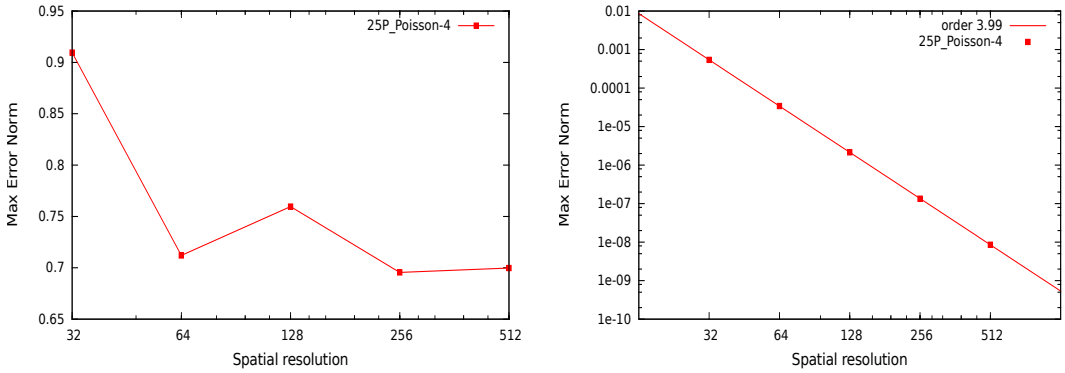


Figure 3.7: *Direct Laplacian - Refined grid - Error convergence of the $\mathcal{O}(4)$ 25 Point scheme vs the $\mathcal{O}(2)$ classical scheme.*



(a) Error vs resolution (Full domain)

(b) Error convergence vs resolution (Partial domain)

Figure 3.8: Error vs resolution plots for the (a) full domain & (b) partial domain : $\sqrt{x^2 + y^2} \leq 0.15$

the assumption that all cell quantities are volume-averages. Interestingly the scheme we develop next not only satisfies this property, but also has a smaller stencil with lesser computations.

Discretization scheme 2 – The Nine-Point Scheme

We start with a 1D derivation, and subsequently we will generalize the derivation to higher dimensions. We use a uniform mesh $x_i = i\Delta x$ with cell centres being $x_i = \frac{x_{i-1/2}+x_{i+1/2}}{2}$. Instead of assuming that the cell-centred grid values u_i of a given function u(x) are known, we assume that the volume-averages given by $\overline{u_i}$ are known, where

$$\overline{u_i} = \frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} u(x)dx$$

We would like to find out a fourth-order expression to numerically compute the derivative of u(x) interpolated to the face points $x_{i-1/2}$ from the given line-averages $\overline{u_i}$. The idea is to first implement it in 1D and then extend the mathematics to 2D and 3D respectively.

We start with defining a primitive function to u(x) using the eq. 3.18

$$U(x) = \int_{x_{-1/2}}^{x} u(\epsilon)d\epsilon \tag{3.18}$$

Where the lower limit $x_{-1/2}$ is irrelavant and can be replaced by any fixed point, which results in eq. 3.19.

$$U(x_{i+1/2}) = \int_{x_{-1/2}}^{x_{i+1/2}} u(\epsilon)d\epsilon = \sum_{l=0}^{i}\int_{x_{l-1/2}}^{x_{l+1/2}} u(\epsilon)d\epsilon = \sum_{l=0}^{i}\Delta x\overline{u_i} \tag{3.19}$$

Hence, with the knowledge of the cell-averages $\overline{u_l}$, we gain knowledge of the point values of the primitive function at cell faces i.e. $U(x_{i+1/2})$. Using these point values, interpolation polynomials of required orders can be constructed for the primitive function U(x). Once a polynomial construction formulation for the primitive function is available it can be differentiated twice to obtain the first derivative of u(x) at the cell faces as: $u^{'}(x) = U^{''}(x)$ at $x = x_{i+1/2}$.

A fourth-order polynomial P(x), given by eq. 3.20, is chosen to represent the Primitive function.

$$U(x) = Ax^4 + Bx^3 + Cx^2 + Dx + E \tag{3.20}$$

Hence, a fourth-order derivative construction can be obtained using eq. 3.21. To compute C, we use the fig. 3.9 to formulate the linear system in eq. 3.22, which is inverted to give eq. 3.23.

$$u^{'}(x_{i+1/2}) = U^{''}(x_{i+1/2}) = U^{''}(0) = 2C \tag{3.21}$$

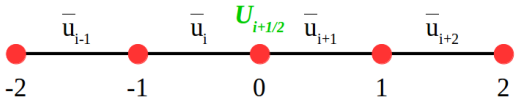


Figure 3.9: Stencil for computing gradient at $x_{i+1/2}$.

$$\begin{aligned}
U_{i-3/2} = U(-2) &= 16A - 8B + 4C - 2D + E = 0\\
U_{i-1/2} = U(-1) &= A - B + C - D + E = \overline{u_{i-1}}\\
U_{i+1/2} = U(0) &= E = \overline{u_{i-1}} + \overline{u_i}\\
U_{i+3/2} = U(1) &= A + B + C + D + E = \overline{u_{i-1}} + \overline{u_i} + \overline{u_{i+1}}\\
U_{i+5/2} = U(2) &= 16A + 8B + 4C + 2D + E = \overline{u_{i-1}} + \overline{u_i} + \overline{u_{i+1}} + \overline{u_{i+2}}
\end{aligned} \tag{3.22}$$

$$\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \frac{1}{\Delta x} \begin{bmatrix} 1/24 & -1/6 & -1/6 & 1/24 \\ -1/12 & 1/6 & -1/6 & 1/12 \\ -1/24 & 2/3 & 2/3 & -1/24 \\ 1/12 & -2/3 & 2/3 & -1/12 \end{bmatrix} \begin{bmatrix} -\overline{u_{i-1}} - \overline{u_i} \\ -\overline{u_i} \\ \overline{u_{i+1}} \\ \overline{u_{i+1}} + \overline{u_{i+2}} \end{bmatrix} \tag{3.23}$$

Hence, using eq. 3.23 & 3.21, we estimate the derivative at face $x_{i+1/2}$ to $\mathcal{O}(4)$ accuracy (eq. 3.24), while the derivative at face $x_{i-1/2}$ is calculated by simply shifting the stencil to the left by one unit (eq. 3.25). Finally, these two derivatives are used to estimate the 1D $\mathcal{O}(4)$ Laplacian operator (eq. 3.26).

$$u^{'}(x_{i+1/2}) = \frac{1}{12\Delta x}[\overline{u_{i-1}} - 15\overline{u_i} + 15\overline{u_{i+1}} - \overline{u_{i+2}}] \tag{3.24}$$

$$u^{'}(x_{i-1/2}) = \frac{1}{12\Delta x}[\overline{u_{i-2}} - 15\overline{u_{i-1}} + 15\overline{u_i} - \overline{u_{i+1}}] \tag{3.25}$$

$$\nabla^2 u(x_i) = \frac{u^{'}(x_{i+1/2}) - u^{'}(x_{i-1/2})}{\Delta x} = \frac{-\overline{u_{i-2}} + 16\overline{u_{i-1}} - 30\overline{u_i} + 16\overline{u_{i+1}} - \overline{u_{i+2}}}{12(\Delta x)^2} \tag{3.26}$$

3.3.5 Higher dimension cases

In this section, we generalize the 9-point scheme to higher dimensions. In a 2D case, the reconstruction problem we face is the following: Given, the spatial averages of a function $A(x_i, y_j)$ in all grid cells as expressed in eq. 3.27, we intend to compute the volume-averaged discrete Laplacian operator on the cell $I_{i,j}$.

$$\overline{A(x_i, y_j)} = \frac{1}{\Delta x \Delta y} \int_{y_{i-1/2}}^{y_{i+1/2}} \int_{x_{i-1/2}}^{x_{i+1/2}} A(\epsilon, \omega) d\epsilon d\omega \tag{3.27}$$

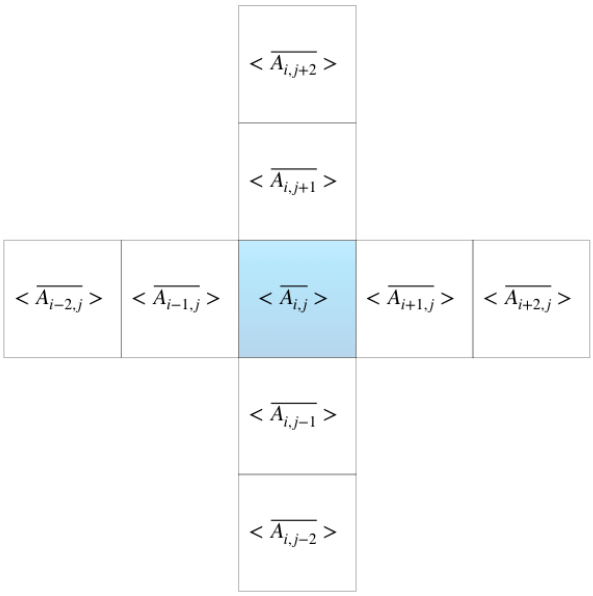


Figure 3.10: A 2D stencil for Poisson 4^{th} order solver - Scheme 2.

The idea is to first reconstruct face-averaged gradients of $A(x, y)$ on each cell face, viz. a total of four face gradients, expressed using eq. 3.28, and subsequently use these face gradients to evaluate the $\mathcal{O}(4)$ Laplacian operator.

$$\begin{aligned} \text{Face } X_{\pm} \quad &: \int_{y_{i-1/2}}^{y_{i+1/2}} \frac{\partial A(x_{i\pm1/2}, \omega)}{\partial x} d\omega \\ \text{Face } Y_{\pm} \quad &: \int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial A(\epsilon, y_{i\pm1/2})}{\partial y} d\epsilon \end{aligned} \tag{3.28}$$

There are two ways to accomplish this. We can either resort to genuine multidimensional reconstruction, in which we have to construct multivariate functions for the polynomial interpolations, and this becomes computationally expensive (especially for 3D problems), or we could use the dimension by dimension reconstruction method in which the problem is broken down into three individual 1D problems. So essentially the same stencil is used as during the computation of the 1D case for computing the x direction gradients, and likewise in the y (and z: in 3D applications) direction. The formulations are expressed in eq. 3.29.

$$\begin{aligned} FaceX_{+} \quad &: \int_{y_{i-1/2}}^{y_{i+1/2}} \frac{\partial A(x_{i+1/2}, \omega)}{\partial x} d\omega = \frac{1}{12\Delta x}[\overline{A_{i-1,j}} - 15\overline{A_{i,j}} + 15\overline{A_{i+1,j}} - \overline{A_{i+2,j}}] \\ FaceX_{-} \quad &: \int_{y_{i-1/2}}^{y_{i+1/2}} \frac{\partial A(x_{i-1/2}, \omega)}{\partial x} d\omega = \frac{1}{12\Delta x}[\overline{A_{i-2,j}} - 15\overline{A_{i-1,j}} + 15\overline{A_{i,j}} - \overline{A_{i+1,j}}] \\ FaceY_{+} \quad &: \int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial A(\epsilon, y_{i+1/2})}{\partial y} d\epsilon = \frac{1}{12\Delta y}[\overline{A_{i,j-1}} - 15\overline{A_{i,j}} + 15\overline{A_{i,j+1}} - \overline{A_{i,j+2}}] \\ FaceY_{-} \quad &: \int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial A(\epsilon, y_{i-1/2})}{\partial y} d\epsilon = \frac{1}{12\Delta y}[\overline{A_{i,j-2}} - 15\overline{A_{i,j-1}} + 15\overline{A_{i,j}} - \overline{A_{i,j+1}}] \end{aligned} \tag{3.29}$$

The Laplacian can henceforth be computed from the face gradients by an application of divergence theorem (eq. 3.30). The case can be easily generalized to 3D, which would require computation of two more face fluxes. Compared to a second-order scheme which uses 3, 5 and 7 stencils for 1D, 2D and 3D cases respectively to compute the Laplacian, a fourth-order scheme will require 5, 9 and 13 stencil values for 1D, 2D and 3D cases respectively. I chose to call this scheme as the 9-point scheme, since most tests, which will be presented in the subsequent sections will be 2D cases, which will use a 9-point stencil.

$$\begin{aligned} \nabla^2 A_{i,j} &= \frac{A_x(x_{i+1/2}, y_j) - A_x(x_{i-1/2}, y_j)}{\Delta x} + \frac{A_y(x_i, y_{j+1/2}) - A_y(x_i, y_{j-1/2})}{\Delta y} \\ &= \frac{-\overline{A_{i-2,j}} + 16\overline{A_{i-1,j}} - 30\overline{A_{i,j}} + 16\overline{A_{i+1,j}} - \overline{A_{i+2,j}}}{12\Delta x^2} + \\ &\quad \frac{-\overline{A_{i,j-2}} + 16\overline{A_{i,j-1}} - 30\overline{A_{i,j}} + 16\overline{A_{i,j+1}} - \overline{A_{i,j+2}}}{12\Delta y^2} \end{aligned} \tag{3.30}$$

Using the expression for the discrete Laplacian operator, the residual and relaxation operators can be built for a Poisson–Helmholtz solver, by using eq. 3.13 & 3.14 respectively. Next, we discuss the implementation of the non-trivial boundary conditions for the Poisson problem.

3.3.6 Boundary Conditions

In case of periodic or symmetry boundary conditions the implementation is quite trivial. However when the boundary conditions are Dirichlet or Neumann, then the need arises to interpolate the values in two layers of ghost cells to close the problem. The derivation is presented for the 1D case and the higher dimension cases can be easily generalized.

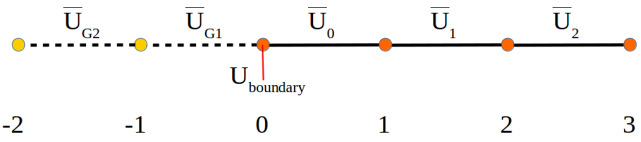


Figure 3.11: Schematic for implementing boundary conditions.

The idea is quite similar to what was done for computation of the gradients in the previous section. For computing the volume-average at U_{G1} we take the three domain cells and the point value at the boundary (fig. 3.11). Once U_{G1} is computed we use this value and two inner volume-average values and the boundary conditions and compute the value of U_{G2}. We start with the primitive function of U(x), which we will call P(x) (eq. 3.31).

$$\begin{aligned} P(x) &= Ax^4 + Bx^3 + Cx^2 + Dx + E \\ U_{Dirichlet} &= P^{'}(0) = D \\ U_{Neumann} &= P^{''}(0) = 2C \end{aligned} \tag{3.31}$$

COMPUTING GHOST VALUE $\overline{U_{G1}}$:-

For this computation, we refer to fig. 3.11, and write down the linear system by using interpolations for the Primitive function. (eq. 3.32). We invert the linear system and obtain eq. 3.33. Finally, using eq. 3.31, we write down the formulations for the ghost-cell-1 value, for both Dirichlet and Neumann condition in eq. 3.34.

$$\begin{aligned} P(-1) &= A - B + C - D + E = 0 \\ P(0) &= E = \overline{U_{G1}} \\ P(1) &= A + B + C + D + E = \overline{U_{G1}} + \overline{U_0} \\ P(2) &= 16A + 8B + 4C + 2D + E = \overline{U_{G1}} + \overline{U_0} + \overline{U_1} \\ P(3) &= 81A + 27B + 9C + 3D + E = \overline{U_{G1}} + \overline{U_0} + \overline{U_1} + \overline{U_2} \end{aligned} \tag{3.32}$$

$$\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} 1/24 & 1/4 & -1/6 & 1/24 \\ -1/4 & -1 & 1/2 & -1/12 \\ 11/24 & 1/4 & 1/6 & -1/24 \\ -1/4 & 3/2 & -1/2 & 1/12 \end{bmatrix} \begin{bmatrix} -\overline{U_{G1}} \\ \overline{U_0} \\ \overline{U_0} + \overline{U_1} \\ \overline{U_0} + \overline{U_1} + \overline{U_2} \end{bmatrix} \tag{3.33}$$

$$\begin{aligned} Dirichlet \quad &: \quad \overline{U_{G1}} = 4U_{Dirichlet} - \frac{13}{3}\overline{U_0} + \frac{5}{3}\overline{U_1} - \frac{1}{3}\overline{U_2} \\ Neumann \quad &: \quad \overline{U_{G1}} = -\frac{12}{11}U_{Neumann} + \frac{9}{11}\overline{U_0} + \frac{3}{11}\overline{U_1} - \frac{1}{11}\overline{U_2} \end{aligned} \tag{3.34}$$

COMPUTING GHOST VALUE $\overline{U_{G2}}$:-

The procedure of interpolation is almost similar to the one described just above, except that now two internal points $\overline{U_0}$ & $\overline{U_1}$, the already computed ghost-cell ($\overline{U_{G1}}$), and the boundary condition are used to compute the ghost-cell-2 value. Using the above conditions on the primitive function, the linear system can be set up and inverted to give eq. 3.35, which can be used to compute the $\overline{U_{G2}}$ values by using eq. 3.36.

$$\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} 1/24 & -1/6 & -1/6 & 1/24 \\ -1/12 & 1/6 & -1/6 & 1/12 \\ -1/24 & 2/3 & 2/3 & -1/24 \\ 1/12 & -2/3 & 2/3 & -1/12 \end{bmatrix} \begin{bmatrix} -\overline{U_{G2}} - \overline{U_{G1}} \\ -\overline{U_{G1}} \\ \overline{U_0} \\ \overline{U_0} + \overline{U_1} \end{bmatrix} \tag{3.35}$$

$$\begin{aligned} Dirichlet \quad &: \quad \overline{U_{G2}} = 16U_{Dirichlet} - \frac{70}{3}\overline{U_0} + \frac{32}{3}\overline{U_1} - \frac{7}{3}\overline{U_2} \\ Neumann \quad &: \quad \overline{U_{G2}} = -\frac{48}{11}U_{Neumann} - \frac{30}{11}\overline{U_0} + \frac{56}{11}\overline{U_1} - \frac{15}{11}\overline{U_2} \end{aligned} \tag{3.36}$$

The generalization to higher dimensions is quite straight forward as we follow the similar dimension by dimension analysis. The boundary conditions that are provided for the problem (Dirichlet/Neumann) are both line-averaged values for 2D problems and surface-averaged values for 3D problems. In a 3D domain, the Dirichlet boundary conditions on the left face can be implemented using eq. 3.37.

$$\begin{aligned} \overline{U_{G1,y_j,z_k}} &= 4 \oiint_{S(x_b,y_j,z_k)} [U_{Dirichlet}(S)]dS - \frac{13}{3} \iiint_{V(x_0,y_j,z_k)} [U(V)]dV \\ &\quad + \frac{5}{3} \iiint_{V(x_1,y_j,z_k)} [U(V)]dV - \frac{1}{3} \iiint_{V(x_2,y_j,z_k)} [U(V)]dV \\ \overline{U_{G2,y_j,z_k}} &= 16 \oiint_{S(x_b,y_j,z_k)} [U_{Dirichlet}(S)]dS - \frac{70}{3} \iiint_{V(x_0,y_j,z_k)} [U(V)]dV \\ &\quad + \frac{32}{3} \iiint_{V(x_1,y_j,z_k)} [U(V)]dV - \frac{7}{3} \iiint_{V(x_2,y_j,z_k)} [U(V)]dV \end{aligned} \tag{3.37}$$

Although, I have the formulations for implementing boundary conditions for the 9-point $\mathcal{O}(4)$ Poisson–Helmholtz scheme, however we will not be using these boundary conditions as currently Basilisk is not equipped with implementing boundary values for problems with two ghost cells. This is a technical issue, which is handled at fundamental grid level programming. Once, this issue gets resolved, the non-trivial conditions can be implemented. For now, we will work on test cases which will implement the trivial periodic or symmetry boundary conditions.

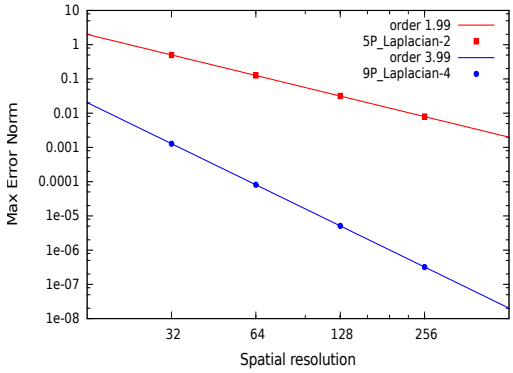
3.4 Results for the 9-point stencil

We begin with uniform grid test cases, where we use a periodic function to solve for both the direct problem and the inverse problem. Here, we work with full finite volume formulation, unlike the 25-point stencil computation case. The direct problem is represented by eq. 3.38, while for the inverse problem, we feed the B values and compute the A values.

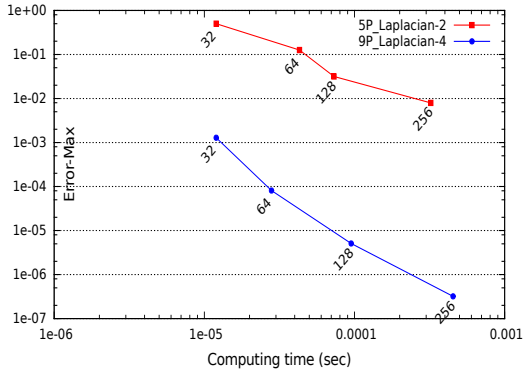
$$\begin{gathered} \nabla^2 \overline{A} = \overline{B} \\ A = \frac{sin(2\pi(x+\frac{\Delta}{2})) - sin(2\pi(x-\frac{\Delta}{2}))}{2\pi\Delta} \times \frac{sin(2\pi(y+\frac{\Delta}{2})) - sin(2\pi(y-\frac{\Delta}{2}))}{2\pi\Delta} \\ B_{an} = 2 \times \frac{sin(2\pi(x+\frac{\Delta}{2})) - sin(2\pi(x-\frac{\Delta}{2}))}{\Delta} \times \frac{sin(2\pi(y+\frac{\Delta}{2})) - sin(2\pi(y-\frac{\Delta}{2}))}{\Delta} \\ Error = B_{num} - B_{an} \end{gathered} \tag{3.38}$$

3.4.1 Uniform grid – Direct problem

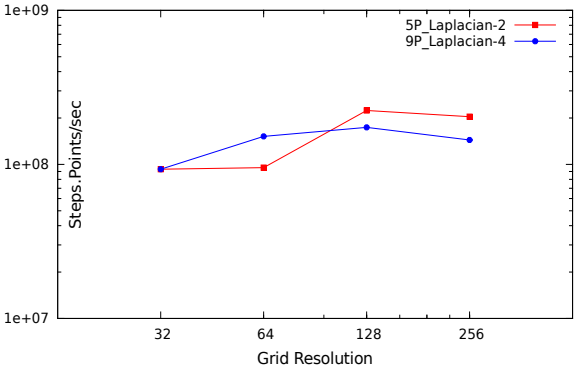
The fig. 3.12a shows an error convergence of $\mathcal{O}(4)$, which is satisfactory. The fig. 3.12b shows that the computation cost is almost similar for the $\mathcal{O}(4)$ & the $\mathcal{O}(2)$ schemes for constant grid resolutions. For instance, at a resolution of 256×256, while the computation cost for the $\mathcal{O}(2)$ scheme is 2.04×10^8 points.steps/sec, the computation cost for the $\mathcal{O}(4)$ scheme is 1.44×10^8 points.steps/sec.



(a) Error convergence vs resolution.



(b) Computing time vs resolution.



(c) Computing speed vs resolution.

Figure 3.12: Direct Laplacian problem - $\mathcal{O}(2)$ vs $\mathcal{O}(4)$ scheme.

3.4.2 Uniform grid – Inverse problem

The inverse problem is solved on uniform grids of varying resolution and the error norms are plotted in fig. 3.13. We get a satisfactory $\mathcal{O}(4)$ convergence for the 9-point scheme. The error norm vs computing time plot in fig. 3.14a shows a superior computing performance for the $\mathcal{O}(4)$ scheme.

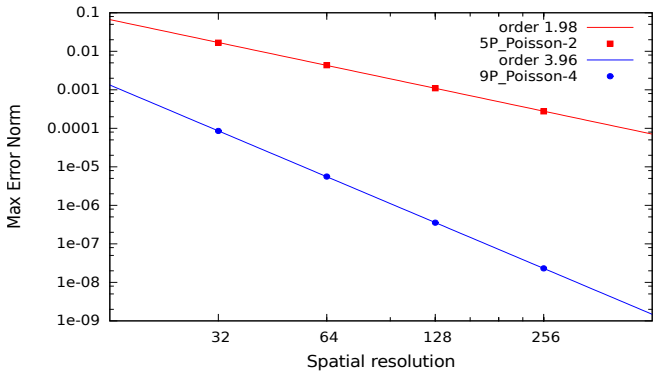
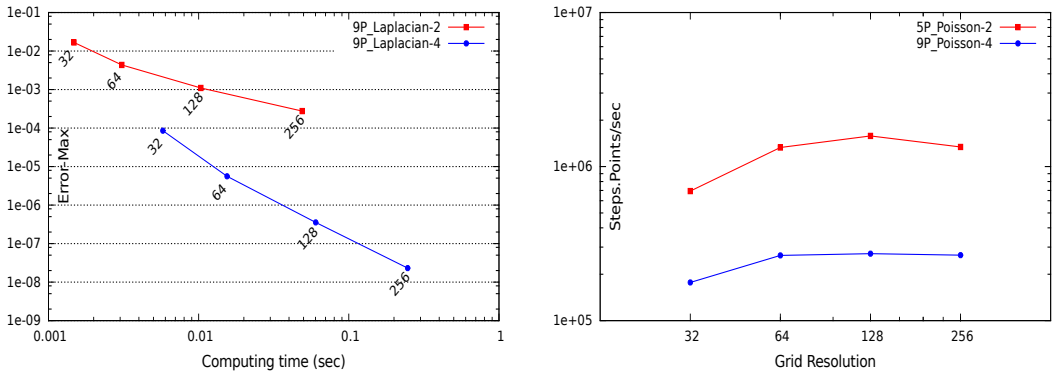


Figure 3.13: Computing speed vs resolution.



(a) Computing time vs resolution.

(b) Error convergence vs resolution.

Figure 3.14: Inverse problem - $\mathcal{O}(2)$ vs $\mathcal{O}(4)$ scheme.

For instance, a 32×32 grid using a $\mathcal{O}(4)$ scheme will generate an error norm which is less than what a 256×256 grid using a $\mathcal{O}(2)$ scheme will generate, and it will do so within a simulation time approximately 10 times smaller. The fig. 3.14b compares the computing cost, and we notice that the $\mathcal{O}(4)$ scheme is roughly five times costlier compared to the $\mathcal{O}(2)$ scheme when compared at similar grid resolutions. Having successfully demonstrated the Poisson problem on a uniform grid, we move on to demonstrating the solution on a refined grid, for which our solution had failed with the 25-point scheme.

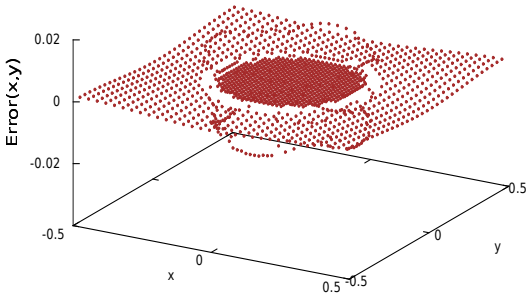
3.4.3 Non-uniform grid – Direct problem

Here we solve the periodic problem defined by eq. 3.38, but use a refined grid as defined by the eq. 3.39.

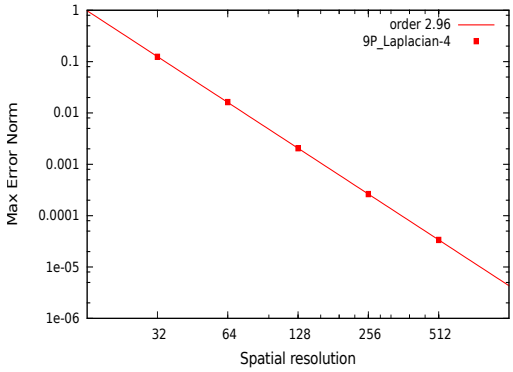
$$[level(x,y)]_N = \begin{cases} N+1 & \text{, if } \sqrt{x^2+y^2} \leq 0.25 \\ N & \text{, otherwise} \end{cases} \tag{3.39}$$

Solving the direct problem we plot the error distribution in fig. 3.15a. We observe that the Laplacian operator shows a discrete jump for the error values at the refinement boundaries,

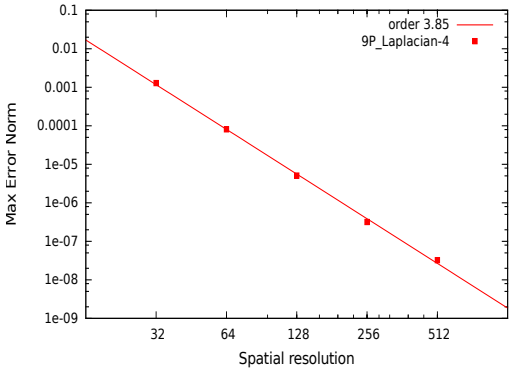
where the error converges to $\mathcal{O}(3)$ as reflected in the full domain error convergence plot in fig. 3.15b, while the error in the domain excluding the immediate neigborhood of the refinement boundaries converges to $\mathcal{O}(4)$ as shown in fig. 3.15c.



(a) Error distribution in the domain.



(b) Error convergence (Full domain).



(c) Error convergence (Partial domain).

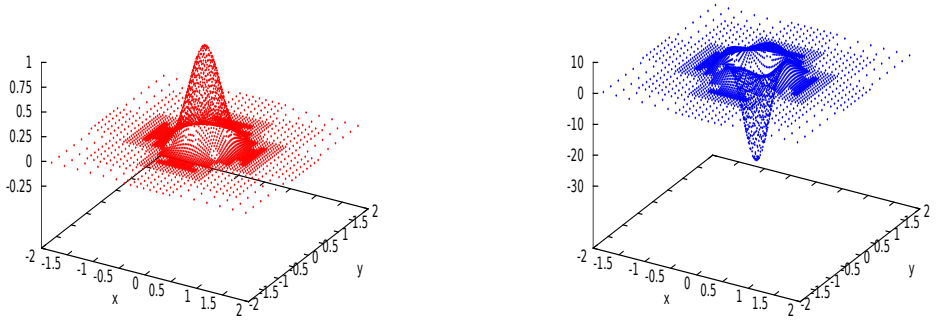
Figure 3.15: Direct Laplacian problem on a refined grid using the 9-point $\mathcal{O}(4)$ scheme
(c) The partial domain is defined as $\sqrt{x^2 + y^2} \leq 0.15$
whereas the refinement boundary is located at $\sqrt{x^2 + y^2} = 0.25$.

This reduction in the order of error convergence of the Laplacian operator near the refinement boundary can be explained by the fact that in the ghost cells at the refinement boundary, the prolongation function interpolates fine cell values to $\mathcal{O}(5)$ accuracy, while the Laplacian (which is a second derivative of the field variable) is an $\mathcal{O}(4)$ formulation and this is exactly the cause for the jump in error and the reduced error convergence. Mathematically a variable interpolated to an accuracy of $\mathcal{O}(5)$ simply cannot have a second derivative higher than $\mathcal{O}(3)$ accuracy, which is why the current 9-point $\mathcal{O}(4)$ stencil discretization leads to a reduced error convergence wherever the Laplacian is computed using at least one of these prolongated values, viz. the neighborhood of the refinement boundary. A method to rectify this would be to build a prolongation scheme which is sixth-order or higher, which would require a six-point stencil or higher, but this is not possible to accomplish (simply) in Basilisk due to the quadtree data structure, whereby all the directly accessible data points lie within a 5×5 stencil, and a $\mathcal{O}(5)$ prolongation function uses the full stencil already.

Hence, using our nine-point scheme, we can arrive at $\mathcal{O}(3)$ solutions for the direct Laplacian problem in case of a refined/adaptive grid, while we will get $\mathcal{O}(4)$ solutions for a similar problem on a uniform grid. We now turn our attention to solving the inverse problem on an adaptive grid and observe the convergence of the error norms.

3.5 Convergence studies on adaptive grids

For studying the performance of the Poisson equation on an adaptive grid, we choose a domain defined by: $(x, y) \subset [-2:2] \times [-2:2]$, with homogenous Neumann conditions on the boundary. The equation we are solving is $\nabla^2\overline{\phi} = \overline{\psi}$. We will be choosing a compact function given by the eq. 3.40, such that the analytical solution for the Poisson equation is given by eq. 3.41. Fig 3.16a & 3.16b gives the domain distribution of the ϕ & ψ functions respectively.



(a) $\phi(x, y)$ **(b)** $\psi(x, y)$

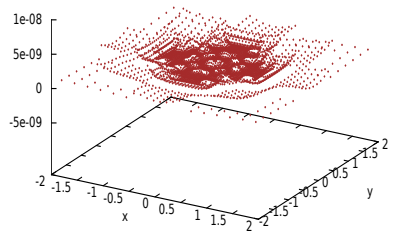
Figure 3.16: The analytical $\phi(x, y)$ & the numerical $\psi(x, y)$ function

$$\psi(x,y) = \begin{cases} 14\big(1-x^2\big)^5\big(1-y^2\big)^7\big(13x^2-1\big)+ \\ 14\big(1-y^2\big)^5\big(1-x^2\big)^7\big(13y^2-1\big), & \text{if} -1 \le x \le 1 \text{ and} -1 \le y \le 1 \\ \\ 0, & \text{otherwise} \end{cases} \tag{3.40}$$

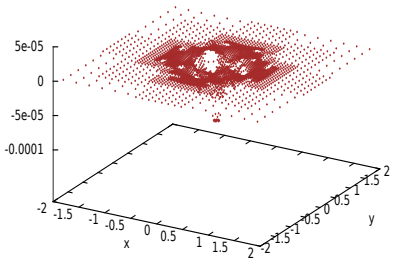
$$\phi(x,y) = \begin{cases} \big(1-x^2\big)^7\big(1-y^2\big)^7, & \text{if} -1 \le x \le 1 \text{ and} -1 \le y \le 1 \\ \\ 0, & \text{otherwise} \end{cases} \tag{3.41}$$

The grid is initialized with a uniform resolution level of 7, and then the Poisson algorithm is run till a tolerance level on the residual is reached. This tolerance level is set at 10^{-5} for the $\mathcal{O}(2)$ solver, and at 10^{-10} for the $\mathcal{O}(4)$ solver. Thereafter, we make calls to the adapt-wavelet function to adaptively resolve the grid on the basis of the difference function computed on ϕ values. After each grid adaptation step the Poisson solver is again run till residuals are reduced. This process is repeated till the adapt-wavelet function stops refining or coarsening the mesh, due to the difference function falling completely in the range of the tolerance limits set by C_{max}.

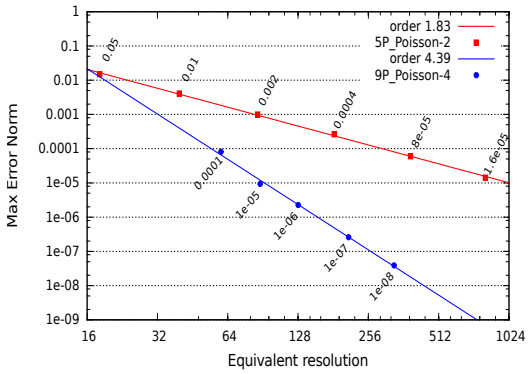
Only the parameter c_{max} controls the level of adaptation, as the other parameter maxlevel is kept at a high arbitrary value, so that it does not interfere with the adaptivity. The c_{max} values are varied from 0.02 to 0.00016 for the second-order scheme, whereas for the $\mathcal{O}(4)$ scheme, it is varied from 10^{-4} to 10^{-7}. The error and residual distribution for a 9-point $\mathcal{O}(4)$ simulation with C_{max} set at a value of 10^{-4} is plotted in fig. 3.17a & 3.17b respectively. The rest of the figures in 3.17 show the convergence and performance statistics. The order of the convergence of the solver is plotted in fig. 3.17c. The orders of the schemes are computed by interpolating a linear function between the max error norm and the equivalent spatial resolution of the adapted grid ($Res_{eq} = \sqrt{N_{ad}}$), on a log-log scale.



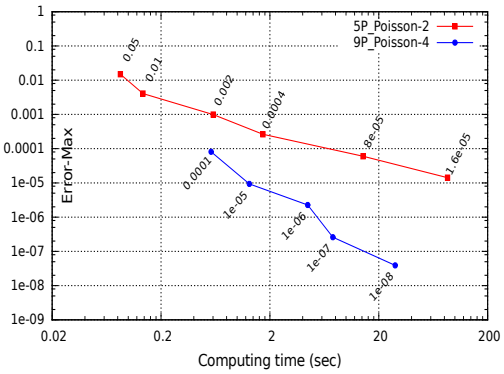
(a) *Residue*(x, y) for $C_{max} = 10^{-4}$.



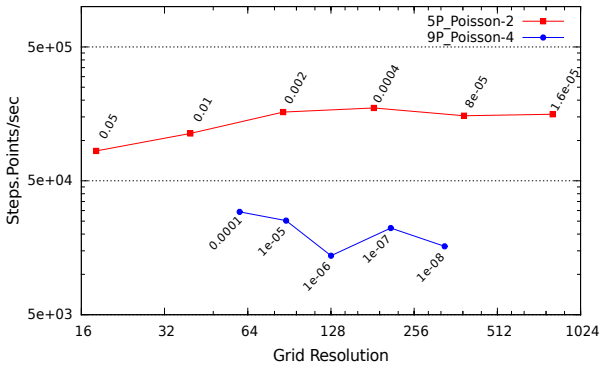
(b) *Error*(x, y) for $C_{max} = 10^{-4}$.



(c) Error convergence vs resolution.



(d) Computing time vs resolution.



(e) Computing speed vs resolution.

Figure 3.17: Poisson problem on adaptive grids - $\mathcal{O}(2)$ vs 9-point $\mathcal{O}(4)$ scheme
(a) & (b) : Results obtained using $\mathcal{O}(4)$ scheme.
(c), (d) & (e) : Chart label texts indicate the C_{max} values.

While the second-order method with bilinear prolongation presents a convergence of order 1.88, the improved fourth-order method with a fifth-order prolongation operator has an error convergence of 4.3. The chart label texts written near the data points as shown in figures 3.17c, 3.17d & 3.17e are the respective C_{max} values of the simulations run.

From fig. 3.17d, we can infer the superior performance of the higher order scheme. For instance, to achieve a max error value of 10^{-5}, the $\mathcal{O}(2)$ scheme requires 658240 cells and performs the computation in roughly 85 seconds, while the $\mathcal{O}(4)$ scheme requires only 7760 cells and does the computation in roughly 1.3 seconds. Finally, we plot the computing cost vs resolution in fig. 3.17e. In this context, a step includes one whole set of iterations till the residual reaches convergence followed by one call to the adapt wavelet function. We observe that each such step is roughly 6-8 times costlier for a $\mathcal{O}(4)$ scheme compared to a $\mathcal{O}(2)$ scheme.

3.6 Conclusion

In conclusion, a higher-order finite-volume method to solve the Poisson-Helmholtz equation on hierarchical cartesian meshes is proposed. The main idea behind this work was to combine the advantages of a higher-order stencil discretization with the efficiency of multigrid methods using higher-order prolongation functions; to achieve lower levels of errors and faster computation times when compared to the existing Basilisk implementation of the classical second-order stencil with a bilinear projection method for multigrids. In this aspect, the new 9-point $\mathcal{O}(4)$ method provides success on both fronts. I must point out that the 9-point stencil, which I derived independently is exactly similar to the one derived in a recent paper by Zhang et al. (2012), however the methodology applied for the derivation of the scheme is completely different in my work. While I took motivation from primitive function based stencil reconstructions (which I had picked up from the literature survey on WENO based advection schemes), the work by Zhang et al. (2012) uses Taylor series constructions.

Another finding of this work was a negative result for the 25-point volume-averaged $\mathcal{O}(4)$ discretization scheme for Poisson–Helmholtz equations and the accompanying biquartic prolongation function for multigrids/adaptivity. This method diverges for non-uniform grids, and should not be used in its current form. It also is undesirable to use this scheme, since the scheme by design, uses both cell centered point values and cell-averaged values in the algorithm.

The prolongation operator, developed in chapter 2, is also successfully applied to implement adaptive algorithms for the Poisson solver and the superior performance is demonstrated here as well. With the new adaptive Poisson-Helmholtz solver, the numerical solutions have lower error-norms and are computed in lesser time to desired error levels, when compared to the classical $\mathcal{O}(2)$ solver. To highlight the importance of this solver , I list down the applications of this Poisson-Helmholtz solver in other areas of my research for this PhD.

3.6.1 Applications of the Poisson–Helmholtz solver

In relation to the solution of the Navier–Stokes equations, the Poisson solver is applied to the numerical computation of the approximate projection operator. At the same time it is also applied to the solution of the implicitly computed viscous terms in the Navier–Stokes momentum equations. We will be going over both in the subsequent subsections.

1. **Approximate projection operator** : One of the equations that need to be solved while numerically computing the Navier–Stokes equations is the eq. 3.42. A Poisson equation needs to be solved with the right hand side being the divergence of a known face velocity field u^*.

$$\nabla \cdot (\alpha \nabla p) = \Delta t \times \nabla \cdot \mathbf{u}^* \tag{3.42}$$

2. **Implicit viscosity solver** : The viscosity solver transforms the viscous equation into a Poisson–Helmholtz type equation through the following mathematical steps:

$$\frac{u^* - u^{adv}}{\Delta t} = \frac{1}{\rho} \nabla \cdot (2\mu D^*)$$

$$\left(\frac{\Delta t}{\rho}\right) \nabla \cdot \left(2\mu D^*\right) + u^* = u^{adv} \tag{3.43}$$

where $D^* = \frac{1}{2}\left(\nabla u + \nabla u^T\right)$ and the multiplication of D^* and μ is a tensor multiplication, in the most general case of an anisotropic flow. The extensive stencil derivations for the same will be carried out in the next chapter on the Navier–Stokes equations.

Chapter 4

Navier–Stokes Solver

Contents

In this chapter, we introduce a higher-order Navier–Stokes solvers. We borrow the concepts and methods developed in Chapters 2 & 3 for developing these schemes. We compare the performance of the newly built higher-order Navier–Stokes solver with the $\mathcal{O}(2)$ existing solver.

4.1 Governing equations

The Navier–Stokes equations are the basic equations of fluid dynamics. For a flow at constant temperature, the NS equations represent two conservation laws, namely the conservation of mass (eq. 4.1) and the conservation of linear momentum (eq. 4.2). The linear stress constitutive relation is given by eq. 4.3

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{4.1}$$

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = -\nabla p + \nabla \cdot \sigma \tag{4.2}$$

$$\sigma = \lambda (\nabla \cdot u) I + 2\mu D \tag{4.3}$$

Where, λ is the coefficient of bulk viscosity and μ is the coefficient of dynamic viscosity.

$$\text{Velocity Deformation Tensor : } D = \frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^T)$$

With the **Incompressible flow** assumption, these equations can be simplified and written as eqs. 4.4 to 4.6.

$$\nabla \cdot \mathbf{u} = 0 \tag{4.4}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{\nabla p}{\rho} + \nabla \cdot \tau \tag{4.5}$$

$$\tau = 2\epsilon D \tag{4.6}$$

4.2 Literature survey

At the heart of a Navier–Stokes solver is a solver for non-linear convection terms, a solver for diffusion terms, a projection algorithm and a time marching scheme. Detailed literature reviews for higher-order schemes have been carried out for the convection term discretizations in section. 2.6 & 2.6.4, for the Poisson algorithm in section. 3.1 (on which the projection algorithm and a semi-implicit diffusion solver are built) and for temporal schemes in section. 2.3. In this section I will provide a background to different Navier–Stokes solvers built over the years.

Over the years, the central issue in a full Navier–Stokes solver has been the implementation of a discrete form for forcing the incompressibility constraint of eq. 4.4. One of the first methods to do so was the artificial pressure-boundary method by Harlow and Welch (1965). Later Krzywicki and Ladyzhenskaya (1966) eliminated the need for an aritificial pressure-boundary by discretizing eq. 4.4 & 4.5 simultaneously, to obtain weak solutions of NS equations. Then came the works of Chorin (1967) & Temam (1969), where the calculation of a discrete hodge projection operator essentially de-coupled the momentum and pressure terms. This idea has been discussed in detail in section. 1.3. Subsequently, Kim and Moin (1985) improved on Chorin's algorithm by using a staggered grid with a $\mathcal{O}(2)$ discretization for non-linear convection terms using the explicit Adams-Bashfort method to obtain $\mathcal{O}(2)$ solutions for NS equations. The work by Bell et al. (1989) (on which Basilisk $\mathcal{O}(2)$ NS-solver implementation is based) uses a second order Godunov unsplit method (see sec. 2.5.1) for discretization of the non-linear convection terms, and a semi-discrete time stepping scheme as discussed in section 4.3.1.

4.3 Navier–Stokes solver by Bell, Colela and Glaz

Basilisk implements the Incompressible Navier–Stokes solver derived by Bell, Colela and Glaz to generate $\mathcal{O}(2)$ solutions to different problems of fluid dynamics. In Chapter 2, I had covered in detail the treatment of the non-linear convection term under the BCG scheme. In this section, I will talk about the implementation of the rest of the steps of the BCG algorithm, viz. the viscous dissipation term discretization, the projection method and the time-marching scheme.

4.3.1 Temporal discretization

We start by re-organizing eq. 4.5 to get to eq. 4.7, and thereafter if we assume temporal smoothness that would imply that the time derivative of $\mathbf{u}$ is divergence free. We can now apply the projection operator to give eq. 4.8, thus eliminating the pressure variable completely from the equation.

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla p = \epsilon \Delta \mathbf{u} - (\mathbf{u} \cdot \nabla)\mathbf{u} \tag{4.7}$$

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{P}(\epsilon \Delta \mathbf{u} - (\mathbf{u} \cdot \nabla)\mathbf{u}) \tag{4.8}$$

We know from the section on convection term discretization in chapter 2, that the non-linear convection term at half time $t^{n+1/2}$ has been computed to $\mathcal{O}(2)$ accuracy from velocity data available at t^n. Similar to the original projection algorithm as derived in Chorin (1967), the bcg projection method also involves two steps, viz. firstly, an auxillary velocity field is constructed and subsequently a discrete projection algorithm is applied. The time stepping strategy as illustrated in eq. 4.9 is based on an iterative algorithm which eventually converges to the $\mathcal{O}(2)$ modified Crank–Nicholson scheme, where $p^{n+1/2,k}$ is the kth iterate approximating $p^{n+1/2}$. We start with the first iterate $p^{n+1/2,0} = p^{n-1/2}$, then use equation 4.9 to compute the intermediate velocity $u^{*,1}$ and use an appropriate projection method (eq. 4.10) to get the next iterate $p^{i+1/2,1}$, which is then re-fed into the equation 4.9, to get the second iterate till the solution gets to an $\mathcal{O}(2)$ convergent solution for velocity and pressure.

$$\frac{\mathbf{u}^{*,\mathbf{k}} - \mathbf{u}^{\mathbf{n}}}{\Delta t} = \frac{\epsilon}{2}\Delta(\mathbf{u}^{\mathbf{n}} + \mathbf{u}^{*,\mathbf{k}}) - [(\mathbf{u} \cdot \nabla)\mathbf{u}]^{n+1/2} - \nabla p^{n+1/2,k} \tag{4.9}$$

It is proven in the same paper that one iteration of intermediate velocity and projection is enough to have a convergent solution of $\mathcal{O}(2)$

4.3.2 Projection Algorithm

Basilisk implements an approximate projection algorithm for the BCG scheme. After the computation of the intermediate cell-centered velocity $\mathbf{u}^*$, a linear interpolation function is used to build the intermediate face-velocity field $\mathbf{u}_f^*$. using the knowledge that u_f^{n+1} is a divergence free vector field, we get a Poisson equation relating the unknown pressure field $p^{n+1/2}$ to the computed intermediate face-velocity field. The classical 5 stencil $\mathcal{O}(2)$ Poisson–Helmholtz solver described in chapter 3 is used to solve this Poisson equation. This pressure field $p^{n+1/2}$, is then used to compute the divergence free face velocity field $\mathbf{u}_f^{n+1}$, which is then interpolated to give the cell-centered velocity field at t_{n+1}, given by $\mathbf{u}^{n+1}$. All the formulations are listed in equations 4.10.

$$\begin{aligned} \nabla \cdot \mathbf{u}_f^{n+1} &= 0 \\ \nabla \cdot (\alpha \nabla p^{n+1/2}) &= \frac{\nabla \cdot \mathbf{u}_f^*}{\Delta t} \\ \mathbf{u}_f^{n+1} &= \mathbf{u}_f^n - \Delta t \times (\alpha \nabla p^{n+1/2}) \end{aligned} \tag{4.10}$$

4.3.3 Viscous dissipation terms

The BCG scheme requires an implicit solution for the viscous dissipation term. While computing the intermediate velocity $\mathbf{u}^*$, we first estimate the velocity after the contribution of the convection terms, viz. u^{adv} given by eq. 4.11

$$\frac{u^{adv} - u^n}{\Delta t} = [(\mathbf{u} \cdot \nabla)\mathbf{u}]^{n+1/2} \tag{4.11}$$

Using the velocity field $\mathbf{u}^{adv}$, the intermediate velocity is solved by casting the equation into a Poisson–Helmholtz equation of the form given by eq. 4.12

$$\begin{aligned} \frac{u^* - u^{adv}}{\Delta t} &= \frac{1}{\rho}\nabla \cdot (\mu D^* + \mu D^n) \\ \Rightarrow -\frac{\Delta t}{\rho}\nabla \cdot (\mu D^*) + u^* &= \frac{\Delta t}{\rho}\nabla \cdot (\mu D^n) + u^{adv} \end{aligned} \tag{4.12}$$

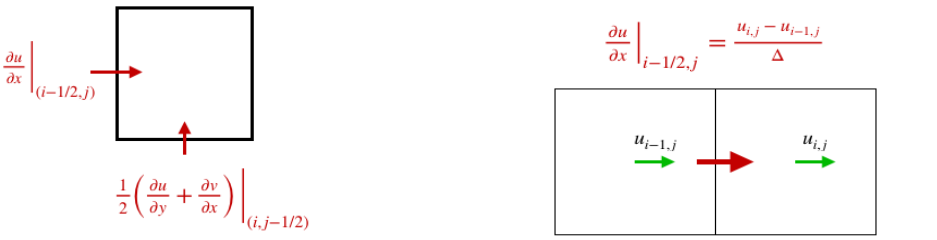
On taking the volume-average of the term $\nabla \cdot (\mu D)$, and using the divergence theorem, we get eq. 4.13

$$\begin{aligned} \frac{1}{\Delta^2}\int\int\int_V (\nabla \cdot (\mu D))dV = &\frac{\mu_{i+1/2,j}D_{i+1/2,j} - \mu_{i-1/2,j}D_{i-1/2,j}}{\Delta^2} \\ &+ \frac{\mu_{i,j+1/2}D_{i,j+1/2} - \mu_{i,j-1/2}D_{i,j-1/2}}{\Delta^2} \end{aligned} \tag{4.13}$$

For a 2D velocity field $\mathbf{u} = u\hat{i} + v\hat{j}$, the velocity deformation tensor D can be expressed using the eq. 4.14.

$$D = \frac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^{\mathbf{T}}) = \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{1}{2}(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}) \\ \frac{1}{2}(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}) & \frac{\partial v}{\partial y} \end{bmatrix} \tag{4.14}$$

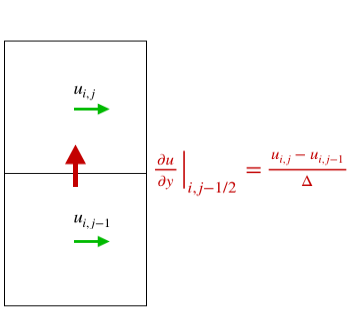
The X equation : The flux of the deformation tensor is calculated using the stencil shown in fig. 4.1a, and eq. 4.13 is expanded into eq. 4.15, which, using the stencil formulations shown in fig. 4.1b, 4.2a & 4.2b gives the final formulation for the viscous dissipation term (eq. 4.16).



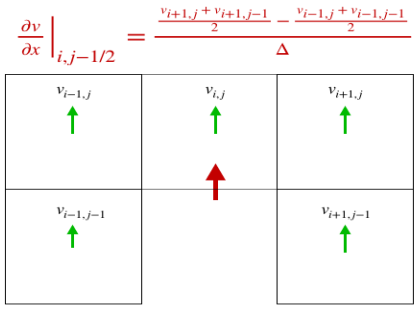
(a) Face-flux terms. **(b)** X-Face-flux computation stencil - $\frac{\partial u}{\partial x}$

Figure 4.1: Viscous Deformation Tensor (Part 1)

$$\begin{aligned} \frac{1}{\Delta^2}\int\int\int_V (\nabla \cdot (\mu D))dV \;=\; &\frac{\mu_{i+1/2,j}\frac{\partial u}{\partial x}\Big|_{i+1/2,j} - \mu_{i-1/2,j}\frac{\partial u}{\partial x}\Big|_{i-1/2,j}}{\Delta} \;+ \\ &\frac{\mu_{i,j+1/2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\Big|_{i,j+1/2} - \mu_{i,j-1/2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\Big|_{i,j-1/2}}{2\Delta} \end{aligned} \tag{4.15}$$



(a) Y-Face-flux computation stencil - $\frac{\partial u}{\partial y}$



(b) Y-Face-flux computation stencil- $\frac{\partial v}{\partial x}$

Figure 4.2: Viscous Deformation Tensor (Part 2)

$$\rightarrow \frac{1}{\Delta^2}\int\int\int_V (\nabla\cdot(\mu D))dV = \frac{\mu_{i+1/2,j}(u_{i+1,j}-u_{i,j}) - \mu_{i-1/2,j}(u_{i,j}-u_{i-1,j})}{\Delta^2} +$$

$$\mu_{i,j+1/2}\frac{(u_{i,j+1}-u_{i,j}) + (\frac{v_{i+1,j+1}+v_{i+1,j}}{2} - \frac{v_{i-1,j+1}+v_{i-1,j}}{2})}{2\Delta^2} -$$

$$\mu_{i,j-1/2}\frac{(u_{i,j}-u_{i,j-1}) + (\frac{v_{i+1,j}+v_{i+1,j-1}}{2} - \frac{v_{i-1,j}+v_{i-1,j-1}}{2})}{2\Delta^2} \tag{4.16}$$

Equation 4.16 is also used to expand the known term $\nabla\cdot(\mu D^*)$ and then the appropriate relaxation and residual operators are constructed. With everything in place the implicit viscosity equation is solved by a call to the $\mathcal{O}(2)$ Poisson–Helmholtz solver defined in chapter 3.

Summarizing the BCG algorithm for the full Navier–Stokes equations

1. Using the velocity and grid spacing data, a suitable timestep is calculated which satisfies the CFL condition.
$$\Delta t \leq MAX_{i,j}\left(\frac{\Delta x}{u_{i,j}}, \frac{\Delta y}{v_{i,j}}\right)$$

2. Use the algorithm described in section 2.5.1 to compute the contribution of the convection term at half time step
$$\mathbf{u}^{adv} = \mathbf{u}^n - \Delta t \times [(\mathbf{u}\cdot\nabla)\mathbf{u}]^{n+1/2}$$

3. Correct the computed intermediate advection velocity using the pressure gradient and source acceleration terms from $t = t^{n-1/2}$, to form the first iterate for the time-marching modified Crank–Nicholson scheme (eq. 4.9).
$$u^{adv} \leftarrow u_{adv} + \Delta t \times \left(\mathbf{a} - \frac{\nabla p}{\rho}\right)\Big|_{t^{n-1/2}}$$

4. Use the corrected u^{adv} to compute the intermediate velocity u^* by feeding it into the implicit viscous solver as discussed in section 4.3.3

5. Correct the intermediate velocity $\mathbf{u}^*$ by removing the contributions of the pressure gradient and source acceleration terms from $t = t^{n-1/2}$
$$u^* \leftarrow u_* - \Delta t \times \left(\mathbf{a} - \frac{\nabla p}{\rho}\right)\Big|_{t^{n-1/2}}$$

6. Use the approximate projection operator (eq. 4.10) to compute the terms $\nabla p^{n+1/2}$ and u_f^{n+1}. Interpolate u_f^{n+1} to build cell-centered velocity field u^{n+1}
7. Correct u^{n+1} using the contributions of the source acceleration and computed pressure gradient terms at $t = t^{n+1/2}$.
8. Finally, Perform $\mathcal{O}(2)$ adaptive refinement, thus finishing one cycle of time-marching.

4.4 Higher-order method for Navier–Stokes equations

The essential components of the higher-order method for the Navier–Stokes equations are a $\mathcal{O}(5)$ WENO interpolations algorithm along with a suitable Riemann solver for computing the convection term, an $\mathcal{O}(4)$ implicit viscosity solver for calculating the dissipation term, an $\mathcal{O}(4)$ projection method based on the $\mathcal{O}(4)$ Poisson–Helmholtz solver, an explicit $\mathcal{O}(4)$ Runge–Kutta based schemes for performing time-marching and a higher-order adaptive method based on a $\mathcal{O}(5)$ prolongation operator. We will discuss each component briefly in the following subsections. This method will highlight bringing together all the numerical schemes developed in chapters 2 and 3.

4.4.1 Time-marching schemes

The time-marching scheme is one based on a Runge–Kutta solver. The user has the choice to use either the classical RK-2, the classical RK-4 or the SSP-RK3 scheme, all part of the header file Runge-Kutta.h. In case of a smooth solution, the classical RK-4 scheme is picked, while for a solution expected to have discontinuities, the SSP-RK3 method is chosen. See section 2.2 for more details on the different time-marching schemes.

4.4.2 Convection term - WENO interpolation and Riemann Solver

Unlike the BCG scheme, where the convection terms are discretized at half time steps $t = t_{n+1/2}$, in the higher-order method the discretization is done at $t = t^n$. Of course, if a RK-4 method is used then the time step is itself split according to Runge–Kutta scheme, and then there are individual time-marching computations for the sub timesteps.

While starting the run from $t = 0$, it is assumed that the cell-centered volume-average velocity u^0 is provided, as an initial condition to the problem. We use the classical five point $\mathcal{O}(5)$ WENO interpolation scheme presented in subsection 2.6.2, to compute the face velocity field $\mathbf{u}_f^0$, which is then used to advect the cell-centered velocity field $\mathbf{u}^0$, quite similar to how a passive tracer is advected. The only difference is that the current advected quantity is a vector and will have two components in 2D or three components for a 3D flow.

While solving the equation for a given time-step $t^n \to t^{n+1}$, the $\mathbf{u}_f^n$ data is used from the previous time step iteration, and is used to advect the cell-centered velocity field $\mathbf{u}^n$. The estimation of the exact face flux requires the solution of a Riemann problem. In case of an incompressible flow the Riemann solver is quite straight-forward. Its a simple upwind method, given by the eq. 4.17. For compressible flows we have to solve problem-dependant Riemann solvers to determine the exact face fluxes.

$$u_{i-1/2} = \begin{cases} u^L_{weno}, & \text{if } u_{i-1} + u_i \geq 0 \\ \\ u^R_{weno}, & \text{otherwise} \end{cases} \tag{4.17}$$

4.4.3 Projection Algorithm

The projection algorithm follows the implementation of equations 4.10 to compute the divergence free face-velocity field at the end of the time-step as well as the pressure at the next half time step $t = t^{n+1/2}$. However, where the method differs from the BCG scheme, is that here a nine-point $\mathcal{O}(4)$ Poisson–Helmholtz solver as developed in Chapter 3 is used to solve the Poisson equation to estimate the pressure. While computing the face velocity at the next timestep from the previous timestep, the pressure gradient term is discretized with an $\mathcal{O}(4)$ expression given by eq. 4.19.

$$u_f^{n+1} = u_f^n - \Delta t \alpha \nabla p \tag{4.18}$$

$$\frac{\partial p}{\partial x}\Big|_{i-1/2,j} = \frac{p_{i-2,j} - 15p_{i-1,j} + 15p_{i,j} - p_{i+1,j}}{12\Delta} \tag{4.19}$$

4.4.4 Viscous dissipation term

The solution for the viscous term is carried out using an implicit formulation and is given by the eq. 4.20. The higher-order formulation uses higher-order interpolation functions to derive $\mathcal{O}(4)$ discretizations for the viscous dissipation terms and uses the subsequent $\mathcal{O}(4)$ Poisson–Helmholtz solver as the linear-solver.

$$\begin{aligned} \frac{u^* - u^{adv}}{\Delta t} &= \frac{1}{\rho}\nabla \cdot (\mu D^* + \mu D^n) \\ \Rightarrow -\frac{\Delta t}{\rho}\nabla \cdot (\mu D^*) + u^* &= \frac{\Delta t}{\rho}\nabla \cdot (\mu D^n) + u^{adv} \end{aligned} \tag{4.20}$$

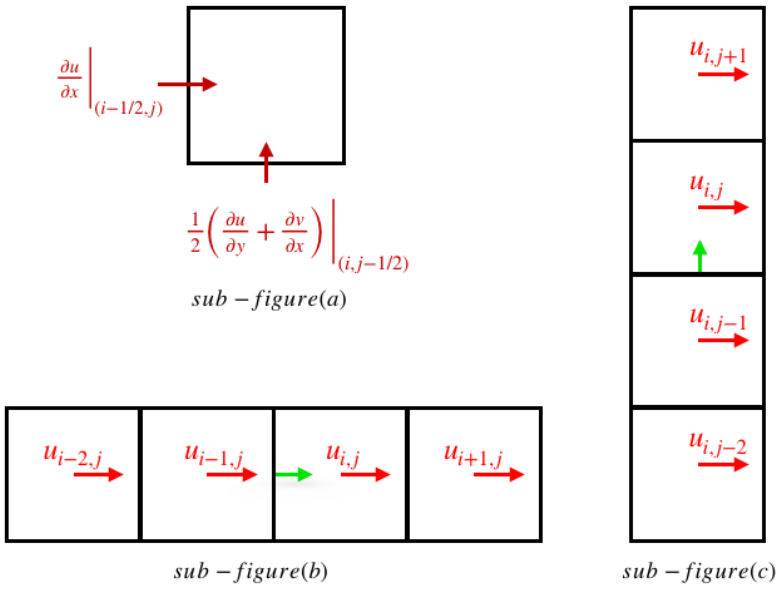


Figure 4.3: Computing the Velocity deformation tensor

The figure 4.3 shows the terms which need to be computed to formulate an expression for the deformation tensor D (subfigure (a)), the stencil required for the computation of the gradient on the x face $\frac{\partial u}{\partial x}$ in subfigure (b), the formulation for which is given by eq. 4.21, and the sub-figure (c) shows the stencil required for computing the gradient $\frac{\partial u}{\partial y}$ on the y face, and the formulation for the same is given by eq. 4.22. Similar derivations have been carried out in subsection 3.3.4, while deriving the gradient operator required for solving the Poisson problem to higher-order using the 9-point stencil.

$$\left.\frac{\partial u}{\partial x}\right|_{i-1/2,j} = \frac{u_{i-2,j} - 15u_{i-1,j} + 15u_{i,j} - u_{i,j+1}}{12\Delta} \tag{4.21}$$

$$\left.\frac{\partial u}{\partial y}\right|_{i,j-1/2} = \frac{u_{i,j-2} - 15u_{i,j-1} + 15u_{i,j} - u_{i,j+1}}{12\Delta} \tag{4.22}$$

The derivation of the formulation for the gradient $\frac{\partial v}{\partial x}$ at the y face is more involved and the stencil required for it is shown in fig. 4.4. The first step is to derive the line-average values of v at the faces marked by the green arrows, from the corresponding surface-averaged values along the vertical direction. For eg. to compute the line-average $v^{line}_{i-2,j-1/2}$ we use the volume-average values from $v_{i-2,j-2}$, $v_{i-2,j-1}$, $v_{i-2,j}$ & $v_{i-2,j+1}$.

We make similar computations for approximating line-average values $v^{line}_{i-1,j-1/2}$, $v^{line}_{i+1,j-1/2}$ and $v^{line}_{i+2,j-1/2}$. Using these four line-average values we build the gradient function $\frac{\partial v}{\partial x}$, at the face $(i, j-1/2)$, marked by the blue arrow. The derivations for the same are carried out subsequently.

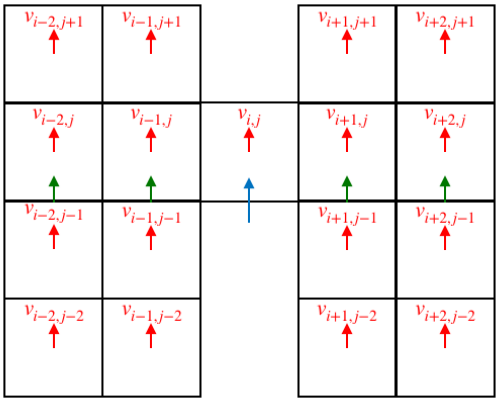


Figure 4.4: Computing the Velocity deformation tensor - $\left.\frac{\partial v}{\partial x}\right|_{i,j-1/2}$

To compute the line-average velocities v on the y-faces, we start with a transverse direction primitive function $P(y)$, defined as in eq. 4.23, similar to the one defined in sections on WENO schemes, but here we use a four point stencil instead of a five point one. We arrive at eq. 4.25, by substituting conditions on the primitive function as listed in eq. 4.24

$$P(y) = Ay^4 + By^3 + Cy^2 + Dy + E \tag{4.23}$$

$$\begin{aligned} P(-2) &= 0 \\ P(-1) &= v_{i-2,j-2} \\ P(0) &= v_{i-2,j-2} + v_{i-2,j-1} \\ P(1) &= v_{i-2,j-2} + v_{i-2,j-1} + v_{i-2,j} \\ P(2) &= v_{i-2,j-2} + v_{i-2,j-1} + v_{i-2,j} + v_{i-2,j+1} \end{aligned} \tag{4.24}$$

$$\begin{bmatrix} 16 & -8 & 4 & -2 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 16 & 8 & 4 & 2 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 0 \\ v_{i-2,j-2} \\ v_{i-2,j-2} + v_{i-2,j-1} \\ v_{i-2,j-2} + v_{i-2,j-1} + v_{i-2,j} \\ v_{i-2,j-2} + v_{i-2,j-1} + v_{i-2,j} + v_{i-2,j+1} \end{bmatrix} \tag{4.25}$$

$$\begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \frac{1}{\Delta} \times \begin{bmatrix} 1/24 & -1/6 & 1/4 & -1/6 & 1/24 \\ -1/12 & 1/6 & 0 & -1/6 & 1/12 \\ -1/24 & 2/3 & -5/4 & 2/3 & -1/24 \\ 1/12 & -2/3 & 0 & 2/3 & -1/12 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ v_{i-2,j-2} \\ v_{i-2,j-2} + v_{i-2,j-1} \\ v_{i-2,j-2} + v_{i-2,j-1} + v_{i-2,j} \\ v_{i-2,j-2} + v_{i-2,j-1} + v_{i-2,j} + v_{i-2,j+1} \end{bmatrix} \tag{4.26}$$

The linear system 4.25 is inverted to obtain eq. 4.26, from where the coefficients can be evaluated. The interpolation for $v^{line}_{i-2,j-1/2}$ is given by eq. 4.27. Similar calculations are made on the other three y-faces to obtain the line-averages : $v^{line}_{i-1,j-1/2}$, $v^{line}_{i+1,j-1/2}$ & $v^{line}_{i+2,j-1/2}$.

$$v^{line}_{i-2,j-1/2} = P'(0) = D = \frac{-v_{i-2,j-2} + 7v_{i-2,j-1} + 7v_{i-2,j} - v_{i-2,j+1}}{12\Delta} \tag{4.27}$$

The next step is to use these four line-average interpolation values of v on the y faces to build the line-average derivative $\frac{\partial v}{\partial x}$ for the face $(i, j-1/2)$. The calculation here is a bit tricky since we start with line-averages, and have to build a line-averaged derivative. The first step is to choose the standard three quadrature points (marked by blue points in fig. 4.5) and subsequently sum up the quadrature interpolations to build the line-average velocity derivative.

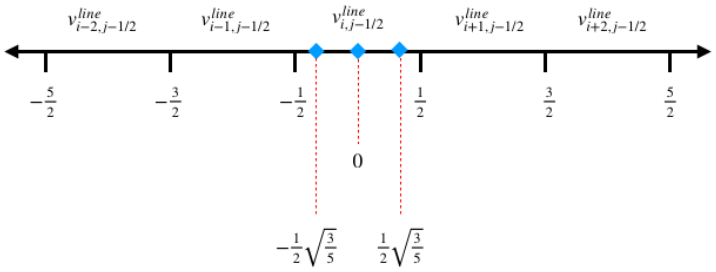


Figure 4.5: Computing the Velocity deformation tensor - $\left.\frac{\partial v}{\partial x}\right|_{i,j-1/2}$

We start with the primitive function $P(x)$, given by eq. 4.28 and impose the conditions given by eq. 4.29, to get the linear system of eq. 4.30, which can be inverted to get to eq. 4.31.

$$P(x) = Ax^5 + Bx^4 + Cx^3 + Dx^2 + Ex + F \tag{4.28}$$

$$\begin{aligned} P(-5/2) &= 0 \\ P(-3/2) &= v^{line}_{i-2,j-1/2} \\ P(-1/2) &= v^{line}_{i-2,j-1/2} + v^{line}_{i-1,j-1/2} \\ P(1/2) &= v^{line}_{i-2,j-1/2} + v^{line}_{i-1,j-1/2} + v^{line}_{i,j-1/2} \\ P(3/2) &= v^{line}_{i-2,j-1/2} + v^{line}_{i-1,j-1/2} + v^{line}_{i,j-1/2} + v^{line}_{i+1,j-1/2} \\ P(5/2) &= v^{line}_{i-2,j-1/2} + v^{line}_{i-1,j-1/2} + v^{line}_{i,j-1/2} + v^{line}_{i+1,j-1/2} + v^{line}_{i+2,j-1/2} \end{aligned} \tag{4.29}$$

$$
\begin{aligned}
\rightarrow & \begin{bmatrix} -3125/32 & 625/16 & -125/8 & 25/4 & -5/2 & 1 \\ -243/32 & 81/16 & -27/8 & 9/4 & -3/2 & 1 \\ -1/32 & 1/16 & -1/8 & 1/4 & -1/2 & 1 \\ 1/32 & 1/16 & 1/8 & 1/4 & 1/2 & 1 \\ 243/32 & 81/16 & 27/8 & 9/4 & 3/2 & 1 \\ 3125/32 & 625/16 & 125/8 & 25/4 & 5/2 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \end{bmatrix} \\
= & \begin{bmatrix} 0 \\ v^{line}_{i-2,j-1/2} \\ v^{line}_{i-2,j-1/2} + v^{line}_{i-1,j-1/2} \\ v^{line}_{i-2,j-1/2} + v^{line}_{i-1,j-1/2} + v^{line}_{i,j-1/2} \\ v^{line}_{i-2,j-1/2} + v^{line}_{i-1,j-1/2} + v^{line}_{i,j-1/2} + v^{line}_{i+1,j-1/2} \\ v^{line}_{i-2,j-1/2} + v^{line}_{i-1,j-1/2} + v^{line}_{i,j-1/2} + v^{line}_{i+1,j-1/2} + v^{line}_{i+2,j-1/2} \end{bmatrix} = RHS
\end{aligned}
\tag{4.30}
$$

$$
\rightarrow \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \end{bmatrix} = \begin{bmatrix} -1/120 & 1/24 & -1/12 & 1/12 & -1/24 & 1/120 \\ 1/48 & -1/16 & 1/24 & 1/24 & -1/16 & 1/48 \\ 1/48 & -13/48 & 17/24 & -17/24 & 13/48 & -1/48 \\ -5/96 & 13/32 & -17/48 & -17/48 & 13/32 & -5/96 \\ -3/640 & 25/384 & -75/64 & 75/64 & -25/384 & 3/640 \\ 3/256 & -25/256 & 75/128 & 75/128 & -25/256 & 3/256 \end{bmatrix} \times RHS \tag{4.31}
$$

From the three point Gaussian quadrature formula and the primitive function design 4.28, we can build the line-averaged derivative by substituting the coefficients from eq. 4.31. The mathematical steps are illustrated in eq. 4.32, and the final line-averaged derivative formulation is given by eq. 4.33

$$
\begin{aligned}
\left.\overline{\frac{\partial v}{\partial x}}\right|_{i,j-1/2} &= \frac{5}{18}P''\left(-\frac{1}{2}\sqrt{\frac{3}{5}}\right) + \frac{8}{18}P''(0) + \frac{5}{18}P''\left(\frac{1}{2}\sqrt{\frac{3}{5}}\right) \\
&= \frac{5}{18}\left\{12B\left(\frac{1}{2}\sqrt{\frac{3}{5}}\right)^2 + 2D\right\} \times 2 + \frac{8}{18} \times 2D \\
&= B + 2D \\
&= \left\{-\frac{1}{48}v^{line}_{i-2,j-1/2} + \frac{1}{24}v^{line}_{i-1,j-1/2} - \frac{1}{24}v^{line}_{i+1,j-1/2} + \frac{1}{48}v_{i+2,j-1/2}\right\} \\
&\quad + 2 \times \left\{\frac{5}{96}v^{line}_{i-2,j-1/2} - \frac{17}{48}v^{line}_{i-1,j-1/2} + \frac{17}{48}v^{line}_{i+1,j-1/2} - \frac{5}{96}v_{i+2,j-1/2}\right\}
\end{aligned}
\tag{4.32}
$$

$$
\rightarrow \left.\overline{\frac{\partial v}{\partial x}}\right|_{i,j-1/2} = \frac{1}{12}v^{line}_{i-2,j-1/2} - \frac{8}{12}v^{line}_{i-1,j-1/2} + \frac{8}{12}v^{line}_{i+1,j-1/2} - \frac{1}{12}v_{i+2,j-1/2} \tag{4.33}
$$

The dissipation term, after volume-averaging and application of the divergence theorem can be expressed using eq. 4.13. The values of the various derivatives can now be filled in by using equations 4.21, 4.22 & 4.33, and the full higher-order approximation for the viscous dissipation term can be expressed using eq. 4.34. The residual operator for the semi-implicit viscosity solver is expressed in eq. 4.35. The corresponding relaxation operators is derived by splitting the Viscosity operator $L(\mathbf{u}, \mu)$ into a diagonal operator $L^D(\mathbf{u}, \mu)$ and a remainder operator $L^R(\mathbf{u}, \mu)$, and is expressed using eq. 4.36. Finally, these operators are fed into the multigrid cycles of the fourth-order Poisson–Helmholtz solver to obtain a new fourth-order implicit viscosity solver.

$$
\begin{aligned}
L_{i,j}(\mu,u) \quad &= \quad \frac{1}{\Delta^2}\int\int\int_V (\nabla\cdot(\mu D))dV = \\
&\frac{1}{12\Delta^2}\Big[\mu_{i+1/2,j}(-u_{i+2,j}+15u_{i+1,j}-15u_{i,j}+u_{i-1,j}) \\
&\qquad -\mu_{i-1/2,j}(-u_{i+1,j}+15u_{i,j}-15u_{i-1,j}+u_{i-2,j})\Big] \\
&+\frac{1}{24\Delta^2}\Big[\mu_{i,j+1/2}(-u_{i,j+2}+15u_{i,j+1}-15u_{i,j}+u_{i,j-1}) \\
&\qquad -\mu_{i,j-1/2}(-u_{i,j+1}+15u_{i,j}-15u_{i,j-1}+u_{i,j-2})\Big] \\
&+\frac{\mu_{i,j+1/2}}{288\Delta^2}\Big[\{-v_{i-2,j-1}+7v_{i-2,j}+7v_{i-2,j+1}-v_{i-2,j+2}\} \\
&\qquad -8\times\{-v_{i-1,j-1}+7v_{i-1,j}+7v_{i-1,j+1}-v_{i-1,j+2}\} \\
&\qquad -8\times\{-v_{i+1,j-1}+7v_{i+1,j}+7v_{i+1,j+1}-v_{i+1,j+2}\} \\
&\qquad +\{-v_{i+2,j-1}+7v_{i+2,j}+7v_{i+2,j+1}-v_{i+2,j+2}\}\Big] \\
&-\frac{\mu_{i,j-1/2}}{288\Delta^2}\Big[\{-v_{i-2,j-2}+7v_{i-2,j-1}+7v_{i-2,j}-v_{i-2,j+1}\} \\
&\qquad -8\times\{-v_{i-1,j-2}+7v_{i-1,j-1}+7v_{i-1,j}-v_{i-1,j+1}\} \\
&\qquad -8\times\{-v_{i+1,j-2}+7v_{i+1,j-1}+7v_{i+1,j}-v_{i+1,j+1}\} \\
&\qquad +\{-v_{i+2,j-2}+7v_{i+2,j-1}+7v_{i+2,j}-v_{i+2,j+1}\}\Big]
\end{aligned}
\tag{4.34}
$$

$$
Res_{i,j} = \overline{\mathbf{u}^{adv}} - \overline{\mathbf{u}^{*}} + \frac{\Delta t}{\rho}L_{i,j}(\mathbf{u}^{adv},\mu^{adv}) - \frac{\Delta t}{\rho}L_{i,j}(\mathbf{u}^{*},\mu^{*}) \tag{4.35}
$$

$$
\begin{aligned}
d\mathbf{u}^{*} &= [L^{D}_{i,j}]^{-1}\times\{Res_{i,j} - L^{R}_{i,j}(\mathbf{u}^{adv},\mu^{adv})\} \\
\mathbf{u}^{*} &= \mathbf{u}^{adv} + d\mathbf{u}^{*}
\end{aligned}
\tag{4.36}
$$

4.4.5 Test case: higher-order semi-implicit viscosity solver

Basilisk is now equipped with a $\mathcal{O}(4)$ viscosity solver. To test the solver performance we run a synthetic test case. We start with choosing a synthetic 2D velocity field called $\mathbf{u}^{adv}$, which is defined by eq. 4.37. We have made sure that the divergence of this chosen velocity field is a non-zero value, so that the cross terms are non-trivial. This is the case in a full NS-solver, because the intermediate velocities calculated after applying the advection step have non-zero divergence, and it is only after application of the projection step later that the divergence of the velocity field is forced to zero. A good synthetic test case should mimic real simulation fields.

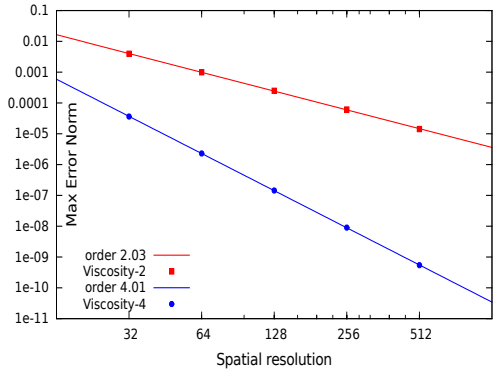
$$
\begin{aligned}
\mathbf{u}^{adv}_{x} &= cos(2\pi x)sin(2\pi y) + \frac{16\pi}{\Delta}sin(\pi\Delta)cos(2\pi x)sin(2\pi y) \\
\mathbf{u}^{adv}_{y} &= sin(2\pi x)cos(2\pi y) + \frac{16\pi}{\Delta}sin(\pi\Delta)sin(2\pi x)cos(2\pi y)
\end{aligned}
\tag{4.37}
$$

The equation we are solving is the implicit viscous-dissipation equation expressed in eq. 4.38. For the purposes of this test case, we assume the values of the density and viscosity fields as unity, and set the timestep as 1. We know for the eq. 4.38, with a known u^{adv} given by eq. 4.37, the analytical solution will be given by eq. 4.39.

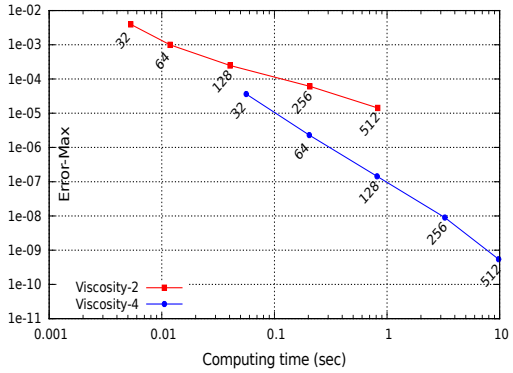
$$\frac{\mathbf{u}^{diff} - \mathbf{u}^{adv}}{\Delta t} = \frac{1}{\rho} L(\mathbf{u}^{diff}, \mu) \tag{4.38}$$

$$\begin{aligned} \mathbf{u}_x^{adv} &= cos(2\pi x) sin(2\pi y) \\ \mathbf{u}_y^{adv} &= sin(2\pi x) cos(2\pi y) \end{aligned} \tag{4.39}$$

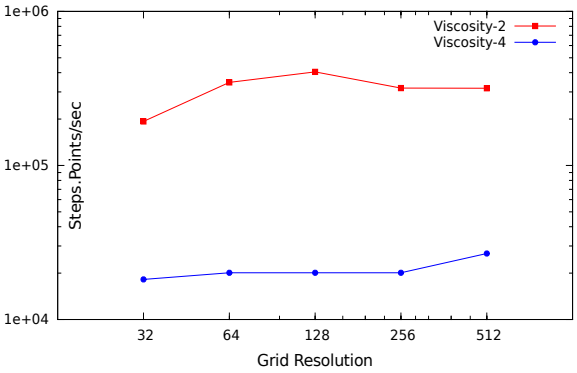
We run the test case and compare the performance statistics of the new $\mathcal{O}(4)$ solver with the existing $\mathcal{O}(2)$ viscosity solver.



(a) Error convergence vs resolution.



(b) Computing time vs resolution.



(c) Computing speed vs resolution.

Figure 4.6: Direct Laplacian problem - $\mathcal{O}(2)$ vs $\mathcal{O}(4)$ scheme.

From fig. 4.6a, we can see the new solver has the desired error convergence order. From fig. 4.6b, we see the $\mathcal{O}(4)$ solver gives the same levels of error as the $\mathcal{O}(2)$ solvers in shorter times. For instance, to reach an L_∞ norm of 10^{-5}, the $\mathcal{O}(2)$ solver requires a 512×512 grid resolution, and makes the computation in roughly 1 second, whereas the same error norm is reached by the $\mathcal{O}(4)$ solver using a 64×64 grid in 0.2 seconds approximately. So the overall $\mathcal{O}(4)$ code has been speed up by fife times. From fig. 4.6c, we notice that each convergence cycle for the $\mathcal{O}(2)$ scheme will be 10 times faster compared to the $\mathcal{O}(4)$ counterpart for fixed resolutions.

Having introduced all the components necessary for solving the full Navier–Stokes equations, we discuss next the well-known test case of a Taylor–Green vortex, and benchmark the performance statistics of the full NS solver.

4.5 Taylor–Green Vortex

Introduced in the classical paper by Taylor and Green (1937), the Taylor–Green vortex is an unsteady flow of a decaying vortex field, which has an exact closed form solution of the incompressible Navier–Stokes equations. Due to this property, this particular test case is used for error convergence studies of different NS solvers.

In this section, we study the case of an inviscid TG vortex, whose solution, in the absence of viscous dissipation should be a steady field. The initial conditions are provided on the velocity field and the pressure field. The initial conditions in their cell-centered point value formulations are given in eq. 4.40 and the corresponding initial conditions in their volume-averaged form are expressed in eq. 4.41. While the former is used to solve the NS equations using the BCG scheme, the latter formulation is used while solving the NS equations using the higher-order scheme.

$$\begin{aligned} u_x(t=0) &= -cos(2\pi x) \times sin(2\pi y); \\ u_y(t=0) &= sin(2\pi x) \times cos(2\pi y); \\ p(t=0) &= -\frac{cos(4\pi x) + cos(4\pi y)}{4} \end{aligned} \tag{4.40}$$

$$u_x(t=0) = \frac{\{sin(2\pi(x+\Delta/2)) - sin(2\pi(x-\Delta/2))\} \times \{cos(2\pi(y+\Delta/2)) - cos(2\pi(y-\Delta/2))\}}{(2\pi\Delta)^2}$$

$$u_y(t=0) = \frac{\{cos(2\pi(x+\Delta/2)) - cos(2\pi(x-\Delta/2))\} \times \{sin(2\pi(y+\Delta/2)) - sin(2\pi(y-\Delta/2))\}}{(2\pi\Delta)^2}$$

$$p(t=0) = -\frac{\{sin(4\pi(x+\Delta/2)) - sin(4\pi(x-\Delta/2)) + sin(4\pi(y+\Delta/2)) - sin(4\pi(y-\Delta/2))\}}{16\pi\Delta} \tag{4.41}$$

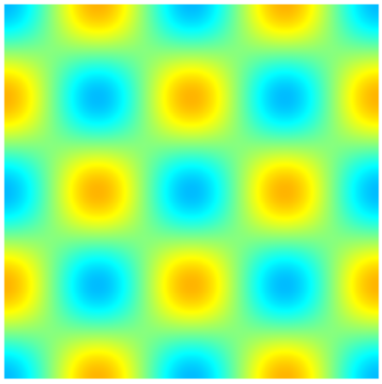


Figure 4.7: *Taylor–Green Vortex Initial kinetic Energy field*

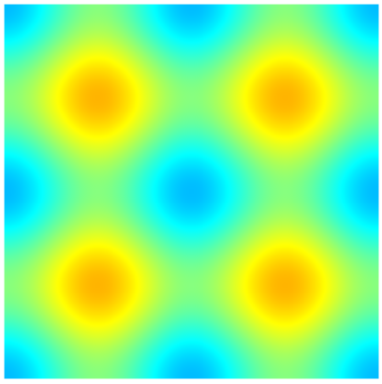


Figure 4.8: *Taylor–Green Vortex Initial Pressure field*

The initial kinetic energy and pressure fields are shown in fig. 4.7 & 4.8 respectively. The domain is periodic and is defined by : $(x, y) \subset [-0.5 : 0.5] \times [-0.5 : 0.5]$. Four different grid resolutions are chosen at 32×32, 64×64, 128×128 and 256×256. The BCG solver and the higher-order solver with RK4 time-marching is used to simulate the flow from $t = 0$ to $t = 2$. The CFL number is fixed at 0.8 for the WENO scheme. The analytical solution in the absence of viscosity will be the exact initial condition. The numerical solution is compared with the analytical solution, and an error convergence study is carried out to observe the order of the schemes.

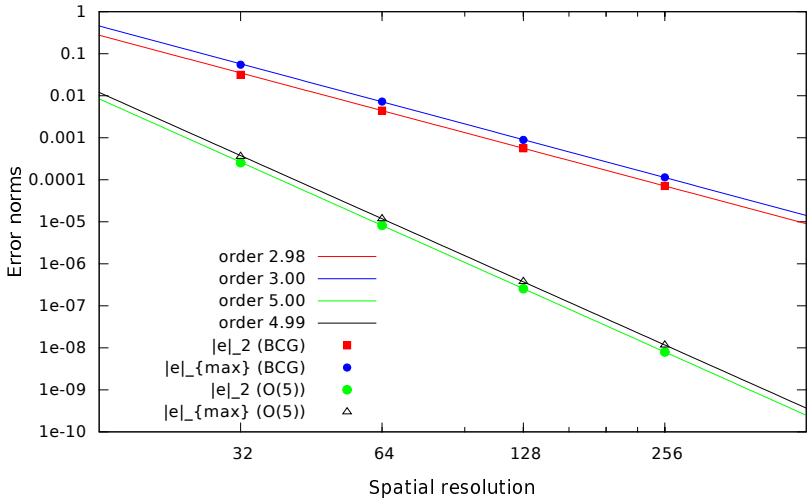


Figure 4.9: Taylor–Green vortex - Error convergence of the NS solver (BCG vs higher-order).

The fig. 4.9 shows an order 3 for the BCG solver and an order 5 for the higher-order method. Firstly, even though the solution is a trivial steady state, this test case is important because it represents a crucial balance between the convection terms and the pressure gradient terms, both of which therefore need to be computed correctly and need to cancel each other out up to the relevant order of the scheme. In the BCG scheme the gradients required for the computation of the convection terms are $\mathcal{O}(3)$, while the projection method is an $\mathcal{O}(2)$ method. A similar comment can be made on the WENO scheme, where the convection term is based on WENO interpolations computed at $\mathcal{O}(5)$, however the projection method and the corresponding poisson solver are $\mathcal{O}(4)$. Since the order of the entire scheme is 3 for the BCG and 5 for the WENO, we can conclude that the errors generated due to the pressure corrections in both cases are smaller compared to the errors generated by the convection term computations.

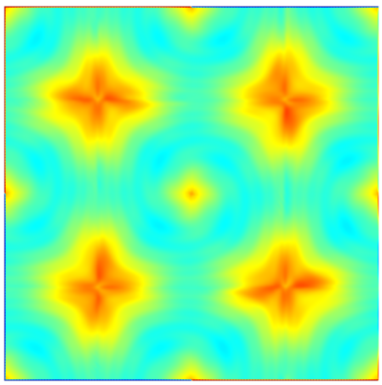


Figure 4.10: *Taylor–Green Vortex (BCG) - Error on kinetic energy field at* $t = 2$, Grid resolution 256×256, RED(maximum) $= 1.5 \times 10^{-4}$

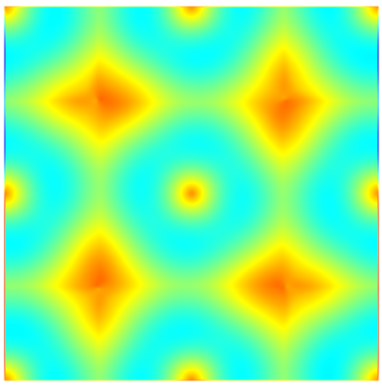


Figure 4.11: *Taylor–Green Vortex (WENO) - Error on kinetic energy field at* $t = 2$, Grid resolution 256×256, RED(maximum) $= 2.89 \times 10^{-8}$

The fig. 4.10 & 4.11 compare the error distribution across the domain, for the BCG scheme and the WENO scheme respectively. All errors are computed on the kinetic energy field. The fig. 4.12 studies the evolution of the max-error on the kinetic energy, with the progress of simulation time. Of course we see a gradual build up of the max-error, as individual time-marching steps contribute to the total error, and hence the overall error increases with time.

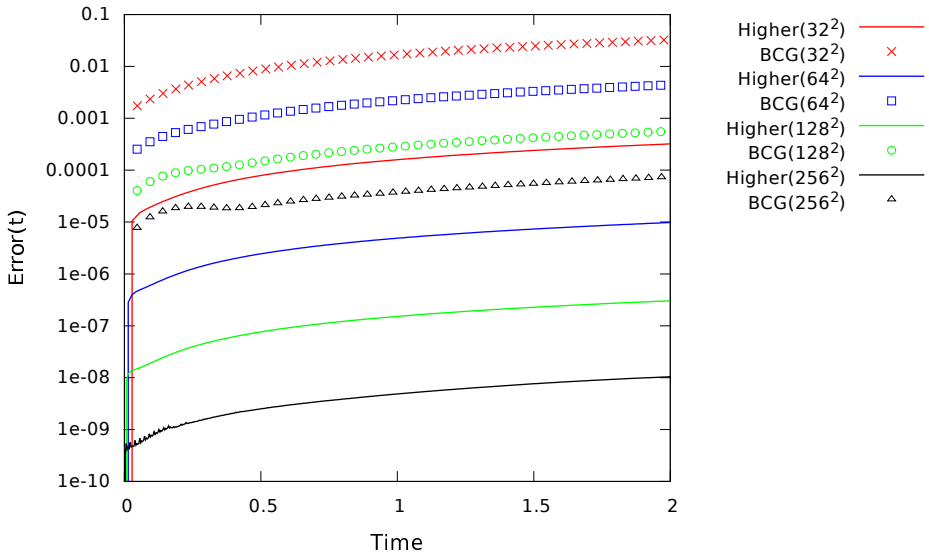


Figure 4.12: Taylor–Green vortex - Evolution of the L2-Error of the NS solver (BCG vs higher-order).

Another quantity of interest is the evolution of the divergence of the cell center velocity field. This is important, since the projection operator is an approximate projection operator, where the divergence of the face averaged velocity field is forced to zero, instead of the divergence of the cell-centered velocity field. The face averaged velocity fields are built by interpolating the cell-centered velocity fields, using different schemes for the BCG and the higher-order algorithms. Therefore, it is important to ensure that the divergence of the cell-centered velocity field, is bounded and converges to lower and lower values as the grid spacing is reduced. This is verified in fig. 4.13.

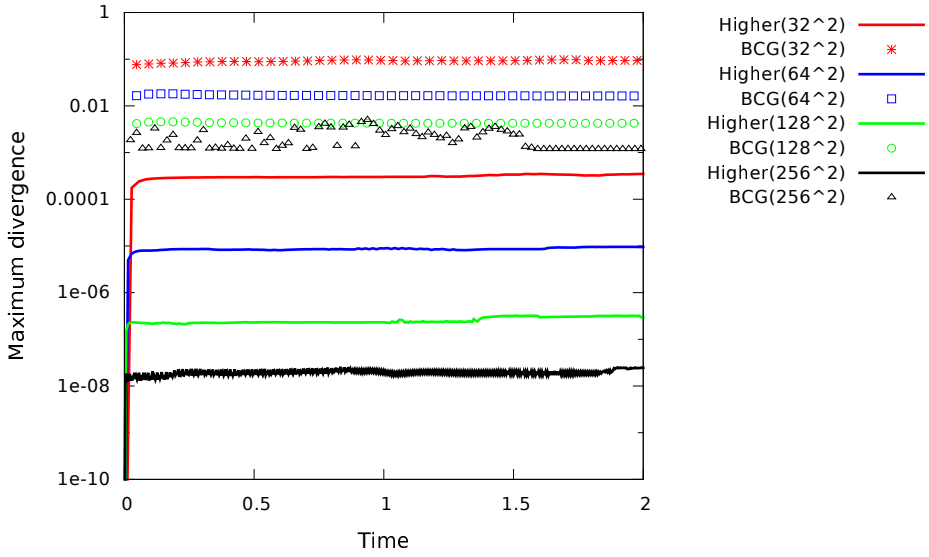


Figure 4.13: Taylor–Green vortex - Time evolution of the Maximum divergence of the centered velocity field.

The case under study is an inviscid case, however every numerical scheme introduces some amount of numerical dissipation. A better scheme would dissipate less. We study the dissipation of the TG vortex by looking at the time series data of the volume-average kinetic energy of the whole domain. Theoretically there should be no drop in this quantity. Numerically we measure the drop and co-relate it to the numerical dissipation factor. We know from the solution of the viscous Taylor–Green vortex, that the general solution at any time t can be expressed using eq. 4.42.

The expression for the numerical Reynolds number can be worked out by fitting the time series data of the kinetic energy into an exponential function, as presented in eq. 4.43.

$$\begin{aligned} u_x &= -e^{-2\nu t} cos(2\pi x) sin(2\pi y) \\ u_y &= e^{-2\nu t} sin(2\pi x) cos(2\pi y) \\ ke &= \frac{1}{2}\{u_x^2 + u_y^2\} \end{aligned} \tag{4.42}$$

$$\begin{aligned} \rightarrow ke(t) &= ke(0) e^{-4\nu t} \\ Re &= \frac{4\pi^2}{\nu} \end{aligned} \tag{4.43}$$

The fig. 4.14 clearly demonstrates the superior performance of the higher-order solver, which has higher numerical Reynolds number compared to the BCG solver, which implies a lower numerical dissipation coefficient for the higher scheme.

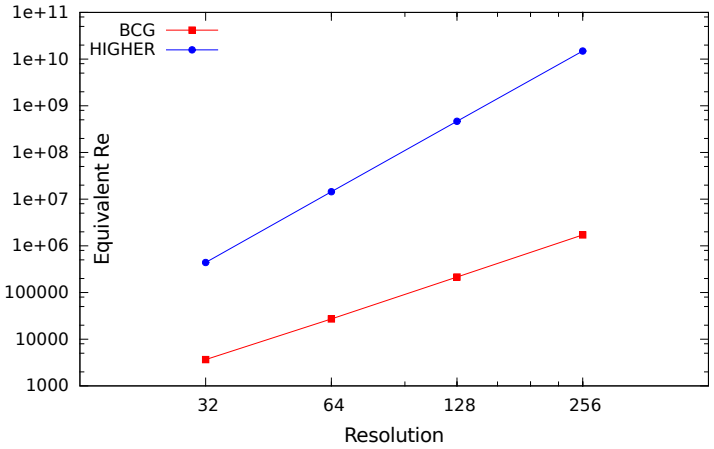


Figure 4.14: Taylor–Green vortex – Equivalent Reynolds Number (BCG vs higher-order).

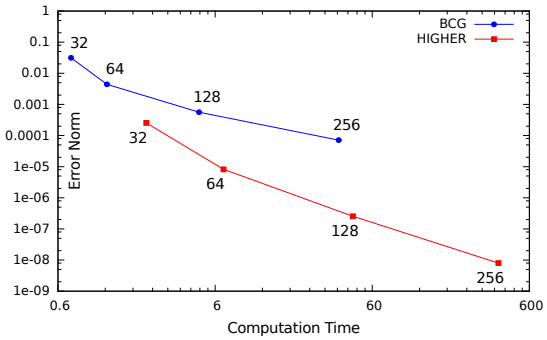


Figure 4.15: TG inviscid vortex – Error norm vs time.

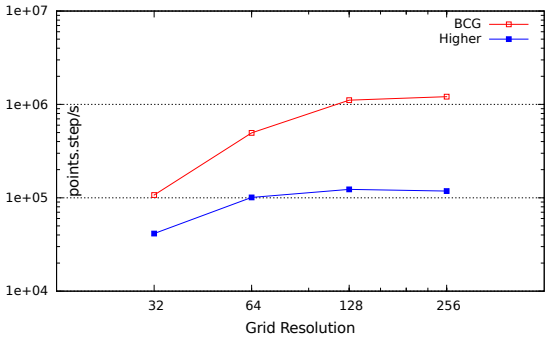


Figure 4.16: TG inviscid vortex – Speed vs resolution.

The fig. 4.15 displays the computation cost of the entire NS solver. On the y-axis we have the error norm and on the x-axis is the time taken for the code execution. The WENO curve lies left and down from the BCG curve, and hence is computationally superior. For instance, to get to a solution accuracy of 10^{-4}, for the BCG scheme we would require a grid size of 256×256 which will have a computation time of roughly 37 seconds, whereas the same solution accuracy level can be achieved using a WENO scheme on a 32×32 grid size with a computation time of roughly 2.5 seconds. This means our higher-order code is 15 times faster than the BCG scheme at error level 10^{-4}. The gap between the two graphs widen as we move to lower accuracy of error values, meaning a higher speedup for the new scheme.

The figure 4.16 shows how expensive each step of the calculation is for both the BCG scheme and the WENO scheme. On the y-axis we have the number of grid points times the number of time-marching steps computed per second, and on the x axis we have the grid resolution. The BCG scheme reaches an asymptotic limit of 10^6 points.steps/s, while the higher-order scheme reaches a limit of approximately 10^5 points.steps/s. This should be fairly obvious, given the computationally expensive stencil calculations carried out for the higher-order scheme.

We now turn our attention to a slightly more complex case, where the Taylor–Green vortices are being advected by a background advection current.

4.6 Taylor–Green vortex with uniform background flow

In this section we add a uniform backflow to the initial condition so the Taylor–Green vortices are advected along with the backflow. This is done to complicate the case from a steady flow to a uniformly advecting one, so we can study the computations of the convection term, time-marching scheme and the projection operator together. The initial condition in the cell-centered form is given by eq. 4.44, and the equivalent volume averaged form can also be written as was done in eq. 4.41. The initial KE and pressure field are plotted in fig. 4.17 & 4.18 respectively.

$$\begin{aligned} u_x(t=0) &= 0.5 - cos(2\pi x) \times sin(2\pi y); \\ u_y(t=0) &= 0.5 + sin(2\pi x) \times cos(2\pi y); \\ p(t=0) &= -\frac{cos(4\pi x) + cos(4\pi y)}{4} \end{aligned} \tag{4.44}$$

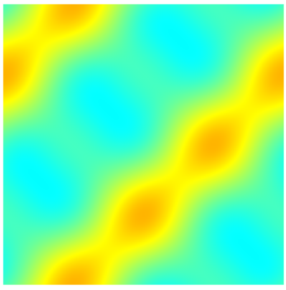


Figure 4.17: *Taylor–Green Vortex Initial kinetic Energy field*

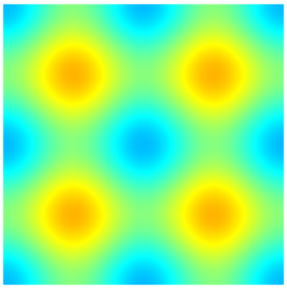


Figure 4.18: *Taylor–Green Vortex Initial Pressure field*

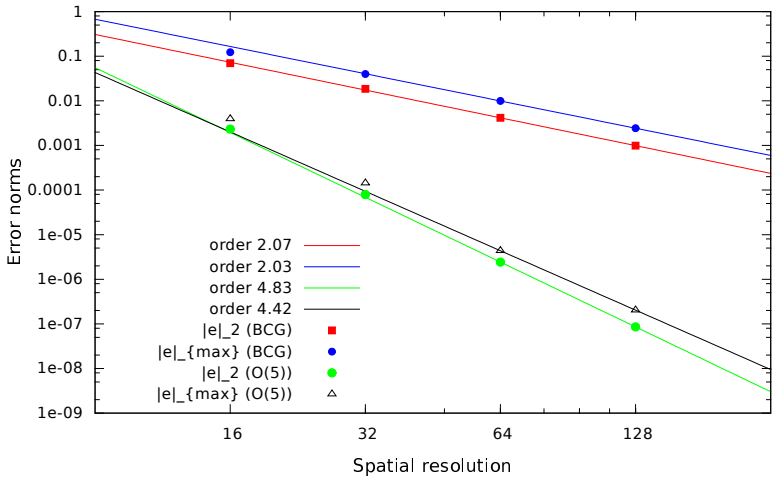


Figure 4.19: TG vortex in uniform flow – Error convergence (BCG vs higher-order).

The error convergence study is carried out and it is observed from fig. 4.19 that the BCG scheme performance falls from $\mathcal{O}(3)$ to $\mathcal{O}(2)$. This can be attributed to the flow changing from a steady

to an unsteady flow, and the effects of time-marching scheme coming into play, which is an $\mathcal{O}(2)$ method. In the case of the higher-order scheme, the error convergence order falls down slightly from $\mathcal{O}(5)$. This drop can be attributed to the Tolerance limit on the Poisson solver residual operator used for the projection algorithm. The current tolerance is set at 10^{-8}. This can be further reduced to increase the error convergence of the solver, however this leads to a penalty on the computation cost, as the number of iterations increase.

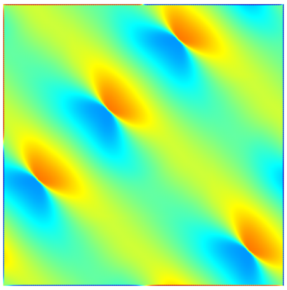


Figure 4.20: *TG vortex in uniform flow (BCG scheme) - Error on KE field at $t = 1$, Grid resolution 128×128, Max (Red) $= 2.5 \times 10^{-3}$.*

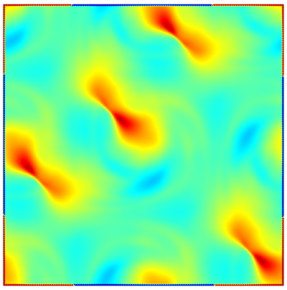


Figure 4.21: *TG vortex in uniform flow (WENO scheme) - Error on KE field at $t = 1$, Grid resolution 128×128, Max(Red) $= 2 \times 10^{-7}$.*

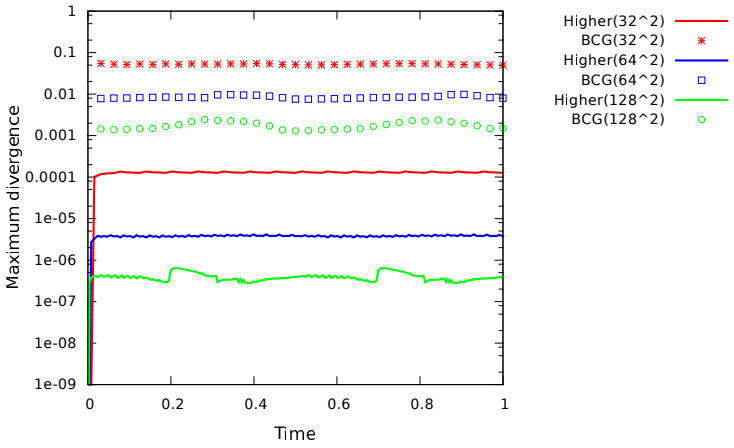


Figure 4.22: *TG vortex in uniform flow - Time evolution of the Maximum divergence of the centered velocity field*

The domain distribution of the error on the KE field is plotted in fig. 4.20 & 4.21 for the BCG & WENO schemes respectively. The maximum divergence timeseries (fig. 4.22) has some noise for higher resolutions, but is well bounded and convergent. The computation costs for the advecting test case is similar to the steady test case (fig. 4.23 & 4.24).

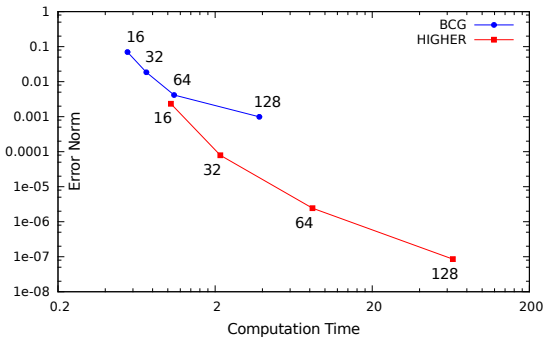


Figure 4.23: TG - Error Norm vs Time

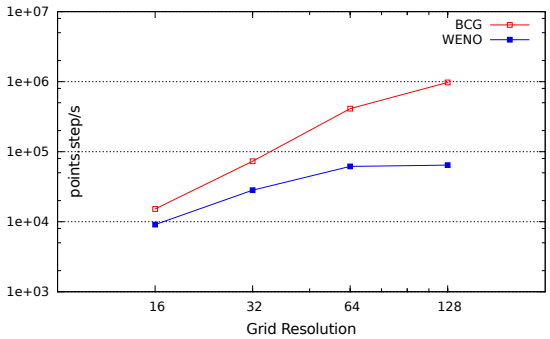
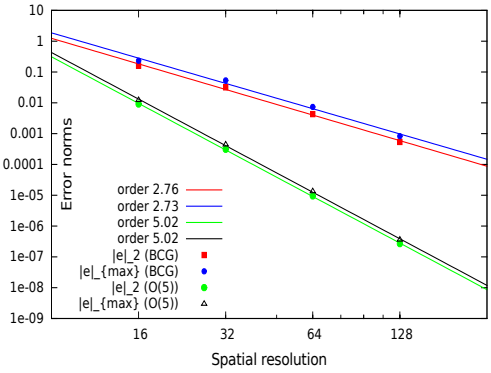


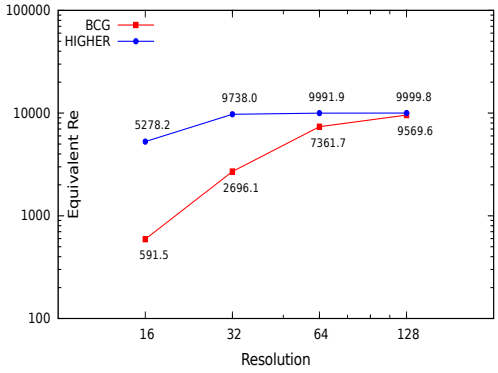
Figure 4.24: TG - Speed vs resolution

4.7 Taylor–Green vortex with viscosity

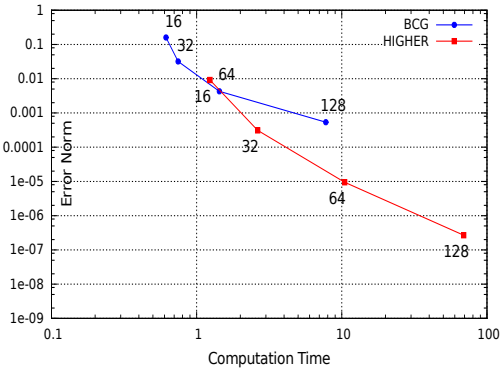
In this test case we solve for the stationary viscous Taylor–Green vortex with a uniform viscous coefficient $\nu = 10^{-4}$. We include the $\mathcal{O}(4)$ implicit viscosity solver with the WENO-5 advection schemes and the RK-4 time-marching scheme. Since the viscosity scheme is implicit, the timestep is only restricted by the CFL criterion for advection. The results are compared with the $\mathcal{O}(2)$ BCG scheme which implements the classical $\mathcal{O}(2)$ viscosity solver.



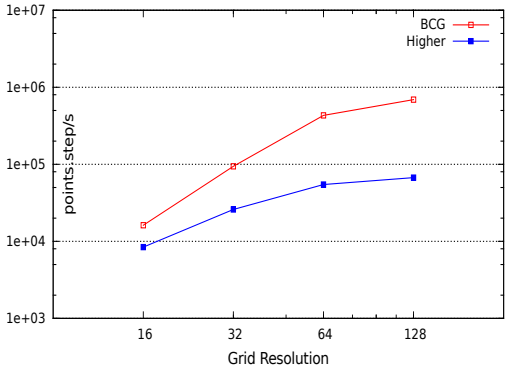
(a) Error norms vs resolution.



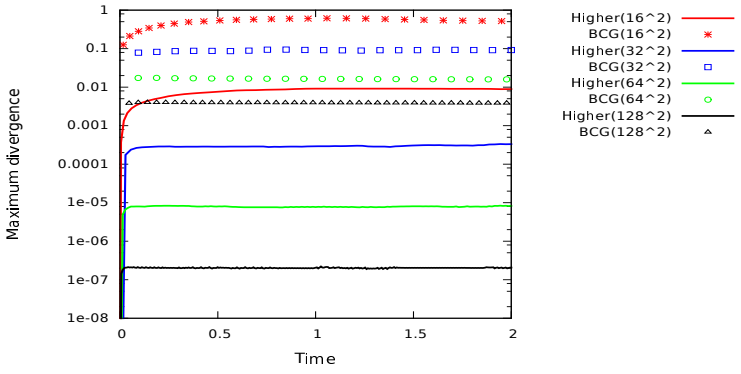
(b) Equivalent Reynolds number vs resolution.



(c) Error vs. computation Time.



(d) Computation speed vs resolution.

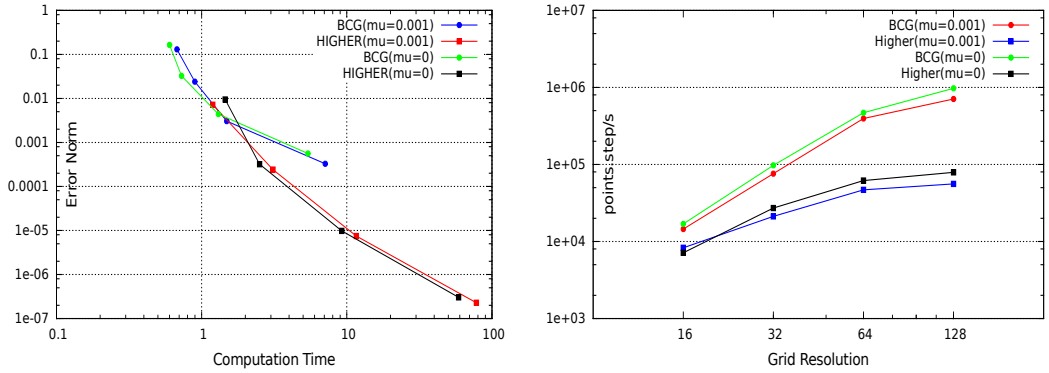


(e) Evolution of Max divergence of centered velocity field.

Figure 4.25: Viscous Taylor–Green vortex - ($\mu = 0.001$).

We observe from fig. 4.25a that the higher-order solver converges to $\mathcal{O}(5)$, which means the convection errors are dominant. Had the errors from the viscous term been dominant the scheme would have reduced to an $\mathcal{O}(4)$ scheme. From fig 4.25b we notice that the Reynolds number is predicted to an accuracy of $\approx 2.5\%$ for the higher-order scheme for a grid resolution of 32×32 grid, whereas the same resolution accuracy for the BCG scheme is $\approx 73\%$. The max divergence

of the centered velocity field is convergent and bounded (fig. 4.25e). The Error vs computation time plot of 4.25c and the Computation speed vs resolution plot of 4.25d are qualitatively very similar to the steady state and advection current test cases.



(a) Error vs Computing time

(b) Computing speed vs resolution

Figure 4.26: Taylor–Green vortex : Comparing performance of a viscous ($\nu = 10^{-3}$) and inviscid case

We compare computational performance of the inviscid and the viscous cases together in fig. 4.26. Here, we have run a simulation for a viscous coefficient of 10^{-3}, and we of course observe that the computational cost has increased. For instance, for a resolution of 128×128, in the case of higher-order scheme, the inviscid solver computes 1.2×10^5 points.steps/sec, while the viscous solver computes 5.6×10^4 points.steps/sec, which makes each time-marching step for the viscous solver 2.1 times more expensive to the inviscid solver. In case of the BCG scheme, for the same resolution, the inviscid method computes 9.8×10^5 points.steps/sec, while the viscous scheme computes 7.1×10^5 points.steps/sec, thus making each time-marching step for the viscous bcg solver 1.4 times expensive, compared to its inviscid counterpart. The decrease in computation-speed is for the viscous case is fully attributed to the additional Poisson equation which is solved at each time-step.

4.8 Conclusion & Future scope

In this chapter, we have been able to successfully bring together the schemes developed in Chapter 2 & 3 to build an $\mathcal{O}(5)$ Navier–Stokes solver. We have also built a new $\mathcal{O}(4)$ semi-implicit viscosity solver, and thereby completed the implementation of a higher-order Navier–Stokes viscous solver. The convergence of the solver has been demonstrated through three different test-cases for a classical Taylor–Green vortex. The net computational time required to reach a given error norm has been cut down by the introduction of the higher-order scheme. What is left to be done from here, is to implement non-trivial boundary conditions (Dirichlet/Neumann/Robin). Derivations of $\mathcal{O}(4)$ boundary condition implementations have been carried out in sub-section 3.3.6.

The full Navier–Stokes equations have not been implemented for adaptive meshes, as this requires further development. The higher-order prolongation function, is currently equipped to handle cell-centered variables (volume-averages). The Gaussian quadrature analysis of Chapter 2, can of course be extended to prolongate face-averaged values. However the problem in the Navier–Stokes solver arises out of the implementation of prolongation on the face velocity fields. This field has a crucial solenoidal property, which must be retained when coarsening / refining cells. The $\mathcal{O}(2)$ method implements this using a local projection method with an auxillary potential (see Popinet (2009)). A suitable higher-order solenoidal prolongation function needs to be built in order to implement adaptive grids for the Navier–Stokes solver.

Chapter 5

Explicit Saint-Venant Schemes

Contents

In this Chapter, we introduce an explicit $\mathcal{O}(2)$ solver for Basilisk implementations of the Saint-Venant equations. We propose implementing improvements to the spatial discretization of this schemes, and observe the effects this has on the dissipation and dispersion relations of the scheme using a sample test case.

5.1 Background

The Saint-Venant equations are a set of hyperbolic partial-differential equations introduced in the work of Saint-Venant (1871). These equations describe the flow below a free surface of a fluid, when specific conditions of shallow-water flows are met, which is, the horizontal length-scales of the flow are far larger than its vertical length-scales. The Saint-Venant equations are derived from the Navier–Stokes equations by depth integrating them. Under the conditions of shallow flow the scale of the vertical velocity of the fluid is small compared to the scale of the horizontal velocity. These equations are widely used for modeling flows in rivers and coastal regions.

Assuming a slowly varying bottom bathymetry $z(x)$ (where, x denotes the coordinates in the horizontal direction) , a water height described by $h(x,t)$ and the water velocity described by $u(x,t)$, the one-dimension Saint-Venant equation can be represented using the formulation given by eqs. 5.1 & 5.2. The two equations can be written in a familiar hyperbolic-equation format of 5.3.

$$\frac{\partial h}{\partial t} + \frac{\partial (h\mathbf{u})}{\partial x} = 0 \tag{5.1}$$

$$\frac{\partial (h\mathbf{u})}{\partial t} + \frac{\partial (h\mathbf{u}^2 + gh^2/2)}{\partial x} = -gh\frac{\partial z}{\partial x} \tag{5.2}$$

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{U}) = S \tag{5.3}$$

Where, $F(U) = (hu, hu^2 + gh^2/2)$, and $U = (h, hu)$. The semi-discrete form of the Saint-Venant equations can now be expressed using eq. 5.4. When solving the Saint-Venant equations

numerically it is extremely important that we are able to preserve the steady state lake at rest condition, which is given by eq. 5.5

$$\Delta x \frac{dU_i(t)}{dt} + F_{i+1/2} - F_{i-1/2} = S_i \tag{5.4}$$

$$h + z = cst \ , \ u = 0 \tag{5.5}$$

Numerically such schemes have been called as well-balanced schemes in the literature, which includes the works of LeVeque (1998), Jin (2001), Kurganov and Levy (2002), Gallouët et al. (2003) & Audusse et al. (2004). Another criterion for a well- designed Saint-Venant numerical scheme, is how it handles dry regions ($h = 0$) or near dry regions ($h \approx 0$). If handled trivially, in such regions, due to numerical oscillations, the height h may become negative, which will lead to a breakdown of the computations, since the eigen-value of the Jacobian of eq. 5.3 are $u \pm \sqrt{gh}$. The numerical schemes of Perthame and Simeoni (2001), Kurganov and Levy (2002) & Audusse et al. (2004) are equipped with positivity preserving property of h.

5.2 Basilisk $\mathcal{O}(2)$ Saint-Venant solver

Since the scheme of Audusse et al. (2004) is equipped with the well-balanced condition as well as the positivity preserving property, hence it was chosen to be generalized for implementation on adaptive-quadtrees in Gerris (see - Popinet (2011)), and the same has been implemented on Basilisk as an explicit $\mathcal{O}(2)$ Saint-Venant solver. This was the state of the art for Basilisk when I started my PhD. In the following subsection, I will describe the mathematical details of the Basilisk implementation of the Saint-Venant solver.

5.2.1 First-order well balanced method

In a basic first-order method the numerical fluxes are supposed to be computed using the cell centered values of $U_i(t)$, and fitting it in a flux function $\mathcal{F}$, which is basically an approximate solution of a Riemann-problem, mathematically expressed as $F_{i+1/2} = \mathcal{F}(U_i(t), U_{i+1}(t))$. The Riemann-problem provides stability to the solution. Some of the well known Riemann solvers are the Godunov, Roe, Kinetic solvers etc. However, cell centered evaluations of the source term will not be able to demonstrate the steady states of a lake at rest solution. Hence, what is done is to build first-order hydrostatic reconstruction for face values of U, viz the flux is now computed using the formulation $F_{i+1/2} = \mathcal{F}(U_{i+1/2-}, U_{i+1/2+})$, where these values are given by eq. 5.6, while the source term is discretized using eq. 5.7.

$$\begin{gathered} U_{i+1/2-} = \begin{pmatrix} h_{i+1/2-} \\ h_{i+1/2-}\, u_i \end{pmatrix} \ , \ U_{i+1/2+} = \begin{pmatrix} h_{i+1/2+} \\ h_{i+1/2+}\, u_{i+1} \end{pmatrix} \\ h_{i+1/2-} = max(0, h_i + z_i - z_{i+1/2}) \ , \ h_{i+1/2+} = max(0, h_{i+1} + z_{i+1} - z_{i+1/2}) \end{gathered} \tag{5.6}$$

$$S_i = \begin{pmatrix} 0 \\ \frac{g}{2} h_{i+1/2-}^2 - \frac{g}{2} h_{i+1/2+}^2 \end{pmatrix} \tag{5.7}$$

5.2.2 Second-order well balanced method

From a given first-order method, the extension to a second-order method is fairly simple. Here the fluxes are computed using limited reconstructed values on both sides of the interface, rather than using the cell centered values. These new values are obtained using three steps, viz. computation of gradients, linear interpolation and limitation procedure. In the second-order scheme, apart from reconstructing values of $\mathbf{U}_{i+1/2-}$ & $\mathbf{U}_{i+1/2+}$, we need new reconstructions for the bathymetry terms ($z_{i,r}$ & $z_{i+1,l}$) as well as a new cell-centered source term S_{ci} needs to be added to the right hand side of eq. 5.4 to maintain consistency, thus we end up solving eq. 5.8. The Basilisk scheme based on the work of Audusse et al. (2004) does so by first computing the left & right reconstructions : $\{h_{i,l}, h_{i,r}, \mathbf{u}_{i,l}, \mathbf{u}_{i,r}, z_{i,l}, z_{i,r}\}$ through limited gradient reconstructions, and then following the steps listed in eq. 5.9.

$$\Delta x \frac{d\mathbf{U}_i(t)}{dt} + F_{i+1/2} - F_{i-1/2} = S_i + S_{ci} \tag{5.8}$$

$$F_{i+1/2} = \mathcal{F}(U_{i+1/2-}, U_{i+1/2+}) \ , \ F_{i-1/2} = \mathcal{F}(U_{i-1/2-}, U_{i-1/2+})$$

$$U_{i+1/2-} = \begin{pmatrix} h_{i+1/2-} \\ h_{i+1/2-}u_{i,r} \end{pmatrix} \ , \ U_{i+1/2+} = \begin{pmatrix} h_{i+1/2+} \\ h_{i+1/2+}u_{i+1,l} \end{pmatrix}$$

$$\begin{aligned} h_{i+1/2-} &= max(0, h_{i,r} + z_{i,r} - z_{i+1/2}) \\ h_{i+1/2+} &= max(0, h_{i+1,l} + z_{i+1,l} - z_{i+1/2}) \\ z_{i+1/2} &= max(z_{i,r}, z_{i+1,l}) \end{aligned} \tag{5.9}$$

$$S_i = S_{i+1/2-} + S_{i-1/2+}$$

$$S_{i+1/2-} = \begin{pmatrix} 0 \\ \frac{g}{2}h_{i+1/2-}^2 - \frac{g}{2}h_{i,r}^2 \end{pmatrix} \ , \ S_{i-1/2+} = \begin{pmatrix} 0 \\ \frac{g}{2}h_{i,l}^2 - \frac{g}{2}h_{i-1/2+}^2 \end{pmatrix}$$

$$S_{ci} = \begin{pmatrix} 0 \\ g\frac{h_{i,l}+h_{i,r}}{2}(z_{i,l} - z_{i,r}) \end{pmatrix}$$

The second-order reconstruction preserves the mass conservation property of the finite volume method, while the limitation procedure ensures the non-negativity of the second-order reconstructed water heights. The second-order reconstruction preserves the lake at rest steady condition.

5.2.3 Riemann Solver

In basilisk, a Riemann solver based on Kurganov's work is used to traditionally calculate the face fluxes $F(U)_{i+1/2}$ and $F(U)_{i-1/2}$, using the left and right reconstructed heights i.e. $h_{i+1/2-}$ and $h_{i+1/2+}$ and left and right reconstructed velocities $u_{i+1/2-}$ and $u_{i+1/2+}$. The flux formulation can be summed up using the equations 5.10 & 5.11.

$$\begin{aligned} a^+ &= max(0 \ , \ u_{i+1/2+} + \sqrt{G * h_{i+1/2+}} \ , \ u_{i+1/2-} + \sqrt{G * h_{i+1/2-}}) \\ a^- &= min(0 \ , \ u_{i+1/2+} - \sqrt{G * h_{i+1/2+}} \ , \ u_{i+1/2-} - \sqrt{G * h_{i+1/2-}}) \\ q^+ &= h_{i+1/2+}u_{i+1/2+} \quad , \quad q^- = h_{i+1/2+}u_{i+1/2-} \end{aligned} \tag{5.10}$$

$$F(U)_{i+1/2} = \begin{pmatrix} \frac{a^+q^- - a^-q^+ + a^+a^-(h_{i+1/2+} - h_{i+1/2-})}{a^+ - a^-} \\ \frac{a^+(q^-u_{i+1/2-} + \frac{G}{2}h_{i+1/2-}^2) - a^-(q^+u_{i+1/2+} + \frac{G}{2}h_{i+1/2+}^2) + a^+a^-(q^+ - q^-)}{a^+ - a^-} \end{pmatrix} \tag{5.11}$$

5.2.4 Time marching scheme – predictor-corrector algorithm

The time integration method used in basilisk is a $\mathcal{O}(2)$ predictor-corrector algorithm. The predictor step is used to predict the updates to the conserved fields h & $h\mathbf{u}$, while the corrector step is used for updating the evolving fields h & $\mathbf{u}$, to get their new values at the next time step.

5.3 Test case - Linear surface gravity wave

Surface gravity waves are interfacial waves generated in a fluid medium, when disturbed from its equilibrium state, where the force of gravity or buoyancy tries to restore equilibrium. Wind generated waves on the ocean surface or tsunamis and tidal-waves all are examples of gravity-waves. For demonstrating the performance of the Basilisk $\mathcal{O}(2)$ code, a simple test case is chosen. A 1D shallow-water domain, with periodic boundary conditions and no bathymetry is used. As an initial condition a small amplitude periodic sinusoidal wave disturbance is imposed on the free surface. The domain with the water heights and initial sinusoidal-disturbance is shown in fig. 5.1.

The orders of magnitude for the various domain length-scales, are chosen to closely mimic the behavior of tsunami waves in deep sea. Tsunami waves have a relatively small wave amplitude in deep-sea (usually 0.3-0.4 metres), while their wavelengths are in the order of 100s of kms. The average offshore sea depth is considered to be in the range of 3–4 kms. The fluid depth is therefore 10^4 times the disturbance wave amplitude, and the wavelength of the disturbance wave is 10^2 times the fluid depth. In the absence of viscous dissipation, the solution should theoretically be a perpetual non-dissipative standing wave on the free surface with identical phase and group velocities given by the formulation $c_p = c_g = \sqrt{gh}$.

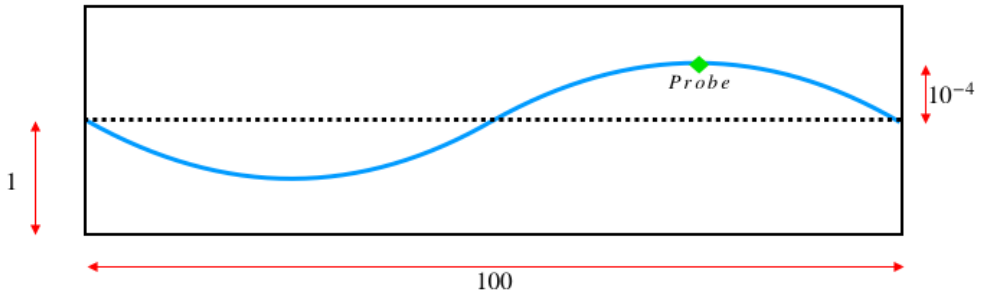
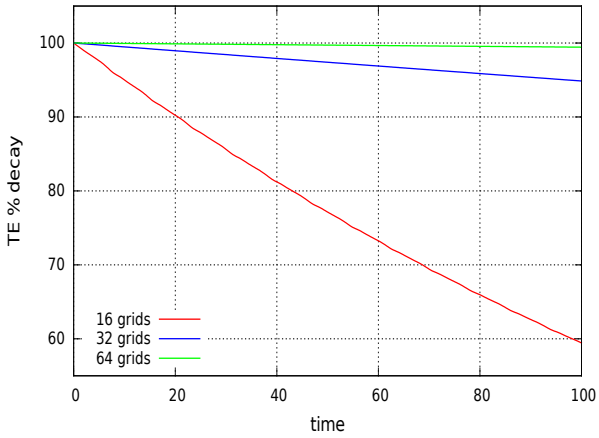


Figure 5.1: Schematic of linear gravity wave test case

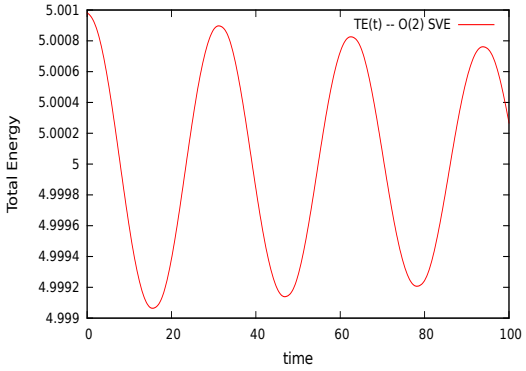
The primary idea behind this test case is to look at the numerical dissipation and numerical dispersion of the scheme implemented in basilisk based on the work of Audusse et al. (2004). The domain-averaged total energy of the disturbance wave is computed at t=0 in 5.12, and we track the percentage fall in this quantity for different grid resolutions in fig. 5.3a.

$$
\begin{aligned}
h_{undis} &= 1 \\
h_{initial} &= 1 + 10^{-4} sin(2\pi x/50) \\
\overline{TE}_{undis} &= 0.5 \times G \times h^2_{undis} = 5 \\
\overline{TE}_{initial} &= \frac{1}{100} \int_{-50}^{50} 0.5 \times G \times h^2_{initial} dx = 5.000000025 \\
\overline{TE}_{disturb}(t=0) &= TE_{initial} - TE_{undis} = 2.5 \times 10^{-8}
\end{aligned}
\tag{5.12}
$$

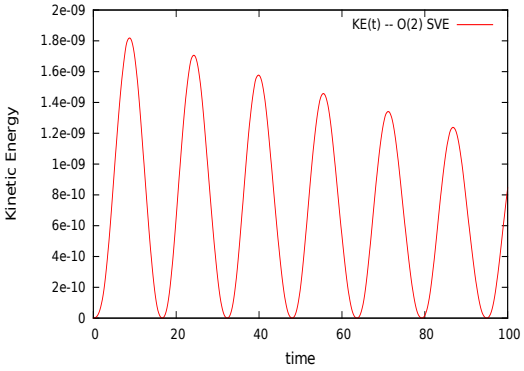
We take a particular probe point marked by a green marker at a crest point in fig. 5.1, and observe the evolution of the total energy and the kinetic energy at the particular probe point in fig. 5.3b & 5.3c for a resolution of 16 cells.



(a) Percentage decay in domain-integrated total energy (Grid-resolution :16, 32 & 64).



(b) Evolution of Total Energy with time at $x = x_{probe}$, results for 16 grid cells.



(c) Evolution of Kinetic Energy with time at $x = x_{probe}$, results for 16 grid cells.

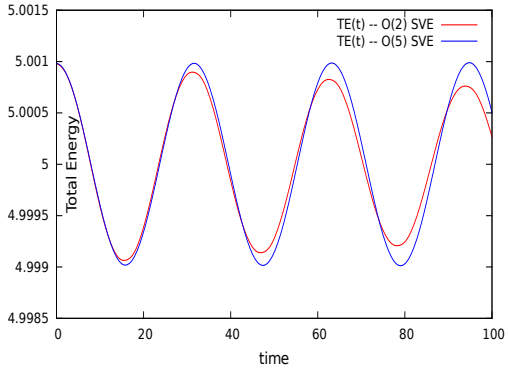
Figure 5.2: Gravity wave test case - $\mathcal{O}(2)$ Saint-Venant solver - Dissipation of Energy.

From all three figures we can observe visible dissipation in the energy of the disturbance wave, which reduces as we improve the grid resolution. However, when solving actual tsunami flows, where the length-scales are widely separated, (the wave propagation can take place over thousands of kms of ocean basin, whereas the impact on a coastline will depend on small details ranging over a couple of meters (eg : sea defenses)), it is important to use very coarse grids in the off-shore region calculations, and an $\mathcal{O}(2)$ scheme introduces numerical dissipations, thereby damping the tsunami-wave amplitudes. We will look at dispersion relations of this scheme in a later subsection.

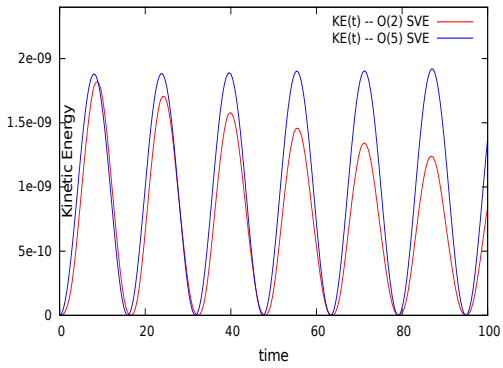
This provides a motivation to improve the general order of the Saint-Venant scheme. The WENO scheme developed in the previous chapter provides a possible solution, since the WENO scheme is a fifth order non-limited scheme in regions of smooth solutions, and reduces to a third order limiting scheme in regions of solution discontinuities. In the past, Xing and Shu (2006) have used WENO schemes with discrete Galerkin Runge–Kutta methods to arrive at a well balanced scheme for still water problems. The linear gravity wave case that we are solving for does not require an interpretation of the surface bathymetry term, and hence all we do in this exercise is change the face reconstruction procedure for $\{h_{i+1,l}, h_{i,r}, \mathbf{u}_{i+1,l}, \mathbf{u}_{i,r}\}$ from a $\mathcal{O}(2)$ limited derivative reconstruction to a $\mathcal{O}(5)$ WENO based reconstruction.

5.3.1 WENO-based explicit Saint-Venant solver

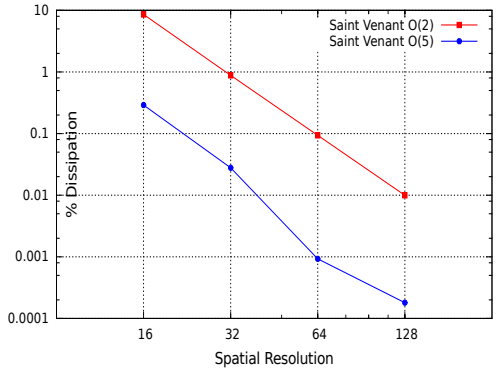
The scheme is implemented by using WENO reconstructions for the heights viz. $h_{i,l}$ and $h_{i,r}$, for the velocities viz. $u_{i,l}$ and $u_{i,r}$ and for the surface bathymetry $z_{i,l}$ and $z_{i,r}$. The flux reconstructions are based on a similar Kurganov Riemann solution for which the $+/-$ face constructions from the left and right constructions are similar to the one derived in Audusse et al. (2004), (eq. 5.9). The time marching scheme is still kept as the predictor-corrector scheme. The simulations have been run at a CFL number of 0.1.



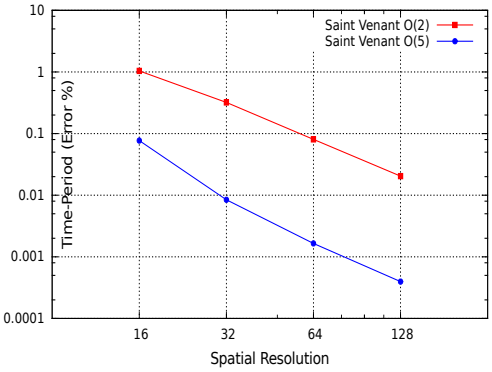
(a) TE vs time at $x = x_{probe}$, (grid-resolution :16).



(b) KE vs time at $x = x_{probe}$, (grid-resolution : 16).



(c) Dissipation error vs Resolution



(d) Dispersion error vs Resolution

Figure 5.3: Gravity wave test case - $\mathcal{O}(2)$ vs $\mathcal{O}(5)$ Saint-Venant solver. Details :-
(c) The dissipation error is calculated at $x = x_{probe}$, by computing the percentage decay in the amplitude of total disturbance energy over one time-period.
(d) The dispersion error is calculated by computing the numerical time period and computing its percentage deviation from the theoretically time period given by ($T = \lambda/\sqrt{G \times h}$).

The figures 5.3a & 5.3b, which are plotted for a resolution of 16 grid-cells, clearly demonstrates a superior performance of the $\mathcal{O}(5)$ WENO based Saint-Venant scheme over the Minmod limiter based $\mathcal{O}(2)$ Saint-Venant schemes. The numerical dissipation has reduced which clearly provides an advantage to the calculations of tidal waves or tsunamis at coarse resolutions. This has been quantified in fig. 5.3c, where two successive peak values of the disturbance total-energy are chosen from a continuous sampling at $x = x_{probe}$, and the decay is the peak value is reported as a percentage of the first peak value, thus giving the dissipation error percentage, which has been plotted for different grid resolutions for both the $\mathcal{O}(2)$ & $\mathcal{O}(5)$ scheme. The dispersion effects are quantified by looking at the time-period data. In a shallow water test case, the gravity waves have an analytical time-period given by the formulation $T = \lambda/\sqrt{G \times h}$. The numerical time-period is computed by looking at the time evolution data of the total-energy, and this is compared with the analytical computation. The percentage errors are plotted for both cases for varying grid resolutions, and it becomes clear that the WENO based scheme is a far better dispersion relation preserving scheme compared to the minmod limiter based scheme.

5.4 Conclusion

A test case simulation of an inviscid gravity wave, using Saint-Venant equations shows a drop in the dissipation as well as dispersion errors, when the interfacial flux terms are re-constructed using the WENO-5 scheme instead of using the $\mathcal{O}(2)$ minmod limiter scheme. The drop in dissipation provides significant gain in computation performance when simulating flows such as tsunamis, where the resolution in the deep sea can be kept very coarse, without dissipating the energy of the wave.

A numerical scheme for shallow-water flows, which has a lower dispersion effect has definite advantages, especially when there might be different gravity waves which have different phase velocities and different wavelengths operating inside the same system. We observe such behaviors in ocean flows. In conclusion, the WENO based advection schemes definitely prove promising for solutions to Saint-Venant equations when applied to tsunami calculations, and hence, further work is needed in this direction, especially deriving numerical discretizations for surface-bathymetry implementation in a way that maintains the lake at rest condition at higher-orders discretely.

Chapter 6

Conclusion & Perspectives

The work accomplished in this thesis manuscript is a contribution to the development and implementation of a higher-order finite-volume Navier–Stokes solver. This work was undertaken on Basilisk with the aim to implement higher-order methods for fluid dynamics equations. The succesful development of the new NS solver is a step in that direction. This work has been carried out by bringing together higher-order solvers for the different sub-components of the NS equations, each of which has either been developed from scratch or using existing schemes in the literature. The various sub-components of the solver include:

1. A convection solver based on WENO $\mathcal{O}(5)$ schemes.
2. A multigrid Poisson–Helmholtz solver based on a new $\mathcal{O}(4)$ discretization scheme.
3. A $\mathcal{O}(4)$ projection method for the NS solver, based on the multigrid Poisson–Helmholtz solver implementation.
4. RK2, RK4 & SSP-RK3 classical time-marching schemes.
5. A new $\mathcal{O}(4)$ semi-implicit solver for diffusion equations.
6. A novel $\mathcal{O}(5)$ prolongation function, for implementing higher-order wavelet-based adaptivity.

An additional output of this work has been the application of the WENO-5 schemes to the $\mathcal{O}(2)$ Saint-Venant solver implementation of Basilisk and a study of the corresponding changes in the performance of the description of the dispersion and dissipation of surface gravity waves. A higher-order solver, when applied to test cases, should demonstrate the theoretical order of convergence. Additionally, for it to replace an existing robust lower-order scheme, the higher-order solver must demonstrate the capability of reaching comparable error levels in shorter computation times. Throughout chapters 2, 3, 4 & 5, where we have developed or implemented different higher-order schemes as listed above, these two criteria have been met for the new higher-order solvers. A range of classical test cases have been applied to different solvers for this purpose. Another criterion which is important for a new solver is its robustness, or its capability of giving consistent accuracy and performance when applied to a range of different problems. This work tried to achieve a comprehensive coverage for all the solvers, using a broad range of test cases, possible within the bounds of symmetry/periodic boundary conditions: this technical limitation on the type of high-order boundary conditions will be lifted soon.

Perspectives

There are certain regions of this work which have future scopes for improvement. They are listed below.

1. A higher-order Discrete–Galerkin Runge–Kutta scheme could be implemented in Basilisk, and its relative performance on adaptive octrees could be studied vis-a-vis the adaptive WENO-5 implementation of this work. This development could then be applied to the solution of the Saint-Venant equations following the work of Xing and Shu (2006), and could find application in the tsunami solver of Popinet (2011).

2. The formulations for the $\mathcal{O}(4)$ implementation of non-trivial boundary conditions for the Poisson–Helmholtz solver have been derived in this work (3.3.6). They can be implemented as soon as Basilisk is equipped with boundary condition implementations on two ghost cells. Thus more complicated / practical cases can be run on the higher-order solver, which can further enhance the study of its robustness. The same can be said about the convection solver, where the boundary conditions can be implemented by following an exactly similar procedure as described for the Poisson–Helmholtz solver.

3. The Navier–Stokes solver is still not an optimized adaptive solver, since the higher-order prolongation function is not equipped with a solenoidal face field prolongation operator. For now adaptivity can be implemented for the NS solver by recomputing the face velocity $\mathbf{u}_f$ using an additional projection step after each time the grid is coarsened/refined, but this will negatively impact on the solver performance. What is needed instead is an equivalent $\mathcal{O}(4)$ operator for the $\mathcal{O}(2)$ implementation discussed in Popinet (2009).

4. The higher-order scheme for the Saint-Venant equations would benefit from significant extra work, since this research only touched upon the discretization of the flux terms. This includes the discretization of the source terms to higher orders to preserve the well-balanced property. A good starting point for this could be the work of Xing and Shu (2011). As already mentioned in point-1, a Runge–Kutta Discrete–Galerkin method is also equally appealing.

Bibliography

Alexandre Joel Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of computational physics*, 2(1):12–26, 1967.

Roger Temam. Sur l'approximation de la solution des équations de navier-stokes par la méthode des pas fractionnaires (ii). *Archive for Rational Mechanics and Analysis*, 33(5):377–385, 1969.

Marsha J Berger and Phillip Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of computational Physics*, 82(1):64–84, 1989.

William John Coirier. *An adaptively-refined, Cartesian cell-based scheme for the Euler and Navier-Stokes equations*, volume 106754. Citeseer, 1994.

Louis H Howell and John B Bell. An adaptive mesh projection method for viscous incompressible flow. *SIAM Journal on Scientific Computing*, 18(4):996–1013, 1997.

Alexei M Khokhlov. Fully threaded tree algorithms for adaptive refinement fluid dynamics simulations. *Journal of Computational Physics*, 143(2):519–543, 1998.

Stéphane Popinet. Gerris: a tree-based adaptive solver for the incompressible euler equations in complex geometries. *Journal of Computational Physics*, 190(2):572–600, 2003.

Randall J LeVeque. Numerical methods for conservation laws. *Lectures in Mathematics ETH Zürich*, 1992.

Thomas Y Hou and Philippe G LeFloch. Why nonconservative schemes converge to wrong solutions: error analysis. *Mathematics of computation*, 62(206):497–530, 1994.

Peter Lax and Burton Wendroff. Systems of conservation laws. *Communications on Pure and Applied mathematics*, 13(2):217–237, 1960.

Sergei Konstantinovich Godunov. A difference method for numerical calculation of discontinuous solutions of the equations of hydrodynamics. *Matematicheskii Sbornik*, 89(3):271–306, 1959.

Edwige Godlewski and Pierre-Arnaud Raviart. *Numerical approximation of hyperbolic systems of conservation laws*, volume 118. Springer Science & Business Media, 2013.

Richard Courant, Kurt Friedrichs, and Hans Lewy. On the partial difference equations of mathematical physics. *IBM journal of Research and Development*, 11(2):215–234, 1967.

Chi-Wang Shu. Total-variation-diminishing time discretizations. *SIAM Journal on Scientific and Statistical Computing*, 9(6):1073–1084, 1988.

Chi-Wang Shu and Stanley Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes, ii. In *Upwind and High-Resolution Schemes*, pages 328–374. Springer, 1989.

Sigal Gottlieb and Chi-Wang Shu. Total variation diminishing runge-kutta schemes. *Mathematics of computation of the American Mathematical Society*, 67(221):73–85, 1998.

Xu-Dong Liu, Stanley Osher, and Tony Chan. Weighted essentially non-oscillatory schemes. *Journal of computational physics*, 115(1):200–212, 1994.

John B Bell, Phillip Colella, and Harland M Glaz. A second-order projection method for the incompressible navier-stokes equations. *Journal of Computational Physics*, 85(2):257–283, 1989.

Phiilip Colella. A multidimensional second order godunov scheme for conservation laws. *J. Comput. Phys*, 87:171–200, 1990.

Bram Vanleer. Multidimensional explicit difference schemes for hyperbolic conservation laws. *Proc. of the sixth int'l. symposium on Computing methods in applied sciences and engineering, VI*, pages 493–497, 1983.

Ami Harten, Bjorn Engquist, Stanley Osher, and Sukumar R Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, iii. In *Upwind and high-resolution schemes*, pages 218–290. Springer, 1987.

Guang-Shan Jiang and Chi-Wang Shu. Efficient implementation of weighted eno schemes. *Journal of computational physics*, 126(1):202–228, 1996.

Philip L Roe. Approximate riemann solvers, parameter vectors, and difference schemes. *Journal of computational physics*, 43(2):357–372, 1981.

Bram Van Leer. Towards the ultimate conservative difference scheme. v. a second-order sequel to godunov's method. *Journal of computational Physics*, 32(1):101–136, 1979.

Ami Harten. High resolution schemes for hyperbolic conservation laws. *Journal of computational physics*, 49(3):357–393, 1983.

Phillip Colella and Paul R Woodward. The piecewise parabolic method (ppm) for gas-dynamical simulations. *Journal of computational physics*, 54(1):174–201, 1984.

Chi-Wang Shu and Stanley Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *Journal of computational physics*, 77(2):439–471, 1988.

Jing Shi, Changqing Hu, and Chi-Wang Shu. A technique of treating negative weights in weno schemes. *Journal of Computational Physics*, 175(1):108–127, 2002.

Haim Nessyahu and Eitan Tadmor. Non-oscillatory central differencing for hyperbolic conservation laws. *Journal of computational physics*, 87(2):408–463, 1990.

Doron Levy, Gabriella Puppo, and Giovanni Russo. Central weno schemes for hyperbolic systems of conservation laws. *ESAIM: Mathematical Modelling and Numerical Analysis*, 33(3):547–571, 1999.

Jianxian Qiu and Chi-Wang Shu. On the construction, comparison, and local characteristic decomposition for high-order central weno schemes. *Journal of Computational Physics*, 183 (1):187–209, 2002.

Bernardo Cockburn and Chi-Wang Shu. The runge–kutta discontinuous galerkin method for conservation laws v: multidimensional systems. *Journal of Computational Physics*, 141(2): 199–224, 1998.

Bernardo Cockburn and Chi-Wang Shu. Runge–kutta discontinuous galerkin methods for convection-dominated problems. *Journal of scientific computing*, 16(3):173–261, 2001.

Senka Vukovic and Luka Sopta. Eno and weno schemes with the exact conservation property for one-dimensional shallow water equations. *Journal of Computational Physics*, 179(2):593–621, 2002.

Yulong Xing and Chi-Wang Shu. High order finite difference weno schemes with the exact conservation property for the shallow water equations. *Journal of Computational Physics*, 208 (1):206–227, 2005.

Yulong Xing and Chi-Wang Shu. A new approach of high order well-balanced finite volume weno schemes and discontinuous galerkin methods for a class of hyperbolic systems with source terms. *Comput. Phys*, 1(1):100–134, 2006.

Sebastian Noelle, Yulong Xing, and Chi-Wang Shu. High-order well-balanced finite volume weno schemes for shallow water equation with moving water. *Journal of Computational Physics*, 226(1):29–58, 2007.

Stephane G Mallat. Multiresolution approximations and wavelet orthonormal bases of l^2. *Transactions of the American mathematical society*, 315(1):69–87, 1989a.

Stephane G Mallat. Multifrequency channel decompositions of images and wavelet models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(12):2091–2110, 1989b.

Stephane G Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE transactions on pattern analysis and machine intelligence*, 11(7):674–693, 1989c.

Yves Meyer. *Wavelets and operators*, volume 1. Cambridge university press, 1992.

Ingrid Daubechies. Orthonormal bases of compactly supported wavelets. *Communications on pure and applied mathematics*, 41(7):909–996, 1988.

Ingrid Daubechies. *Ten lectures on wavelets*, volume 61. Siam, 1992.

Wim Sweldens. The lifting scheme: A construction of second generation wavelets. *SIAM journal on mathematical analysis*, 29(2):511–546, 1998.

Keith W Morton. *Numerical solution of convection-diffusion problems*. Chapman & Hall, 1996.

Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation*, 31(138):333–390, 1977.

Youcef Saad and Martin H Schultz. Conjugate gradient-like algorithms for solving nonsymmetric linear systems. *Mathematics of Computation*, 44(170):417–424, 1985.

Murli M Gupta, Jules Kouatchou, and Jun Zhang. Comparison of second-and fourth-order discretizations for multigrid poisson solvers. *Journal of Computational Physics*, 132(2):226–232, 1997.

Scott R Fulton, Paul E Ciesielski, and Wayne H Schubert. Multigrid methods for elliptic problems: A review. *Monthly Weather Review*, 114(5):943–959, 1986.

Lothar Collatz. Integral and functional equations. In *The Numerical Treatment of Differential Equations*, pages 467–535. Springer, 1960.

Michael Barad and Phillip Colella. A fourth-order accurate local refinement method for poisson's equation. *Journal of Computational Physics*, 209(1):1–18, 2005.

Qinghai Zhang, Hans Johansen, and Phillip Colella. A fourth-order accurate finite-volume method with structured adaptive mesh refinement for solving the advection-diffusion equation. *SIAM Journal on Scientific Computing*, 34(2):B179–B201, 2012.

Francis H Harlow and J Eddie Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The physics of fluids*, 8(12):2182–2189, 1965.

Andrzej Krzywicki and Olga Aleksandrovna Ladyzhenskaya. The method of nets for nonstationary navier–stokes equations. *Trudy Matematicheskogo Instituta imeni VA Steklova*, 92:93–99, 1966.

John Kim and Parviz Moin. Application of a fractional-step method to incompressible navier-stokes equations. *Journal of computational physics*, 59(2):308–323, 1985.

Geoffrey Ingram Taylor and Albert Edward Green. Mechanism of the production of small eddies from large ones. *Proc. R. Soc. Lond. A*, 158(895):499–521, 1937.

Stéphane Popinet. An accurate adaptive solver for surface-tension-driven interfacial flows. *Journal of Computational Physics*, 228(16):5838–5866, 2009.

AJC de Saint-Venant. Theorie du mouvement non permanent des eaux, avec application aux crues des rivieres et a l'introduction de marees dans leurs lits. *Comptes rendus des seances de l'Academie des Sciences*, 36:174–154, 1871.

Randall J LeVeque. Balancing source terms and flux gradients in high-resolution godunov methods: the quasi-steady wave-propagation algorithm. *Journal of computational physics*, 146(1): 346–365, 1998.

Shi Jin. A steady-state capturing method for hyperbolic systems with geometrical source terms. *ESAIM: Mathematical Modelling and Numerical Analysis*, 35(4):631–645, 2001.

Alexander Kurganov and Doron Levy. Central-upwind schemes for the saint-venant system. *ESAIM: Mathematical Modelling and Numerical Analysis*, 36(3):397–425, 2002.

Thierry Gallouët, Jean-Marc Hérard, and Nicolas Seguin. Some approximate godunov schemes to compute shallow-water equations with topography. *Computers & Fluids*, 32(4):479–513, 2003.

Emmanuel Audusse, François Bouchut, Marie-Odile Bristeau, Rupert Klein, and Benoı t Perthame. A fast and stable well-balanced scheme with hydrostatic reconstruction for shallow water flows. *SIAM Journal on Scientific Computing*, 25(6):2050–2065, 2004.

Benoît Perthame and Chiara Simeoni. A kinetic scheme for the saint-venant system with a source term. *Calcolo*, 38(4):201–231, 2001.

Stéphane Popinet. Quadtree-adaptive tsunami modelling. *Ocean Dynamics*, 61(9):1261–1285, 2011.

Yulong Xing and Chi-Wang Shu. High-order finite volume weno schemes for the shallow water equations with dry states. *Advances in Water Resources*, 34(8):1026–1038, 2011.