Smoothness-Increasing Accuracy-Conserving (SIAC) Line Filtering: Effective Rotation for Multidimensional Fields

Julia Docampo Sánchez

University of East Anglia

School of Mathematics

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in Applied Mathematics

Norwich, November 2016

Abstract

Over the past few decades there has been a strong effort towards the development of Smoothness-Increasing Accuracy-Conserving (SIAC) filters for Discontinuous Galerkin (DG) methods, designed to increase the smoothness and improve the convergence rate of the DG solution through this post-processor. The applications of these filters in multidimension have traditionally employed a tensor product kernel, allowing a natural extension of the theory developed for one-dimensional problems. In addition, the tensor product has always been done along the Cartesian axis, resulting in a filter whose support has fixed shape and orientation. This thesis has challenged these assumptions, leading to the investigation of rotated filters: tensor product filters with variable orientation. Combining this approach with previous experiments on lower-dimension filtering, a new and computationally efficient subfamily for post-processing multidimensional data has been developed: SIAC Line filters. These filters transform the integral of the convolution into a line integral. Hence, the computational advantages are immediate: the simulation times become significantly shorter and the complexity of the algorithm design reduces to a one-dimensional problem.

In the thesis, a solid theoretical background for SIAC Line filters has been established. Theoretical error estimates have been developed, showing how Line filtering preserves the properties of traditional tensor product filtering, including smoothness recovery and improvement in the convergence rate. Furthermore, different numerical experiments were performed, exhibiting how these filters achieve the same accuracy at significantly lower computational costs. This affords great advantages towards the applications of these filters during flow visualization; one important limiting factor of a tensor product structure is that the filter grows in support as the field dimension increases, becoming computationally expensive. SIAC Line filters have proven efficiency in computational performance, thus overcoming the limitations presented by the tensor product filter. The experiments carried out on streamline visualization suggest that these filters are a promising tool in scientific visualisation.

Contents

Abstract **I**

List of Figures **III**

List of Tables **VI**

Acknowledgements **X**

1 Introduction **1**

1.1 Contributions . . . 3

2 Background **5**

2.1 DG Schemes . . . 5

2.2 SIAC Filters . . . 12

2.3 General SIAC Filters . . . 13

2.3.1 Discussion . . . 16

2.4 Divided Differences & Translation Invariance . . . 18

2.5 Summary . . . 19

3 Tensor Product Rotated Filters **20**

3.1 Mathematical Formulation . . . 21

3.1.1 Rotated Convolution . . . 22

3.2 Global Analysis: Superconvergence & Errors . . . 25

3.3 Local Behaviour . . . 26

3.4 Discussion . . . 36

4 SIAC Line Filters **38**

4.1 Proving Superconvergence for SIAC Filters . . . 39

4.2 SIAC Line Kernels . . . 41

4.2.1 Univariate B-Splines Along Lines in $\mathbb{R}^2$ . . . 41

4.2.2 Differentiation and Divided Differences . . . 43

4.3 Error Estimates for SIAC Line Filters . . . 47

4.4 Numerical Results . . . 49

4.4.1 Recovering Smoothness . . . 50

4.4.2 Accuracy Enhancement . . . 55

4.5 Discussion . . . 68

5 **Computing SIAC Filters** **69**
5.1 Finding the Kernel Coefficients 70
5.2 Implementation of SIAC Filters 72
5.2.1 The Intersection Algorithm 72
5.2.2 The Kernel Integral Delimiters 74
5.2.3 Finding all the Integral Regions 75
5.2.4 Numerical Integration over Arbitrary Regions 78
5.2.5 Implementation of the SIAC Line Filter 81
5.3 Computational Study 83
5.4 Discussion 83

6 **SIAC Filters and Streamline Visualisation** **88**
6.1 Streamlines and ODE solvers 89
6.2 Symmetric SIAC Filters 91
6.2.1 Line Kernels 91
6.2.2 Line Kernels vs Tensor Product Kernels 100
6.3 Boundary Filters 105
6.3.1 Boundary Line Filters 105
6.3.2 Filtering Along the Streamline 108
6.4 Closest Point Approach 117
6.5 Discussion 117

7 **Conclusions and Future Research** **122**
7.1 Future Resarch 124

List of Figures

2.1 Point-wise errors (log) profiles of the DG approximation and its divided differences 11

2.2 B-Splines (left) for different degrees and two symmetric SIAC kernels (centre and right). 13

2.3 Three different types of boundary filters. 15

2.4 Point-wise errors (log) before and after filtering the DG solution. 17

2.5 Two meshes highlighting the superelement that also forms a uniform mesh. 19

3.1 Support of a Cartesian axis aligned filter projected onto a uniform quadrilateral mesh. 21

3.2 Comparison of the support of the Cartesian axis aligned filter with a rotated filter aligned with the streamline direction. 21

3.3 Illustration of the basis vectors for the two coordinate systems. 24

3.4 Structure of the 2D tensor product filters using a Cartesian axis aligned kernel and a $\pi/4$-rotated kernel. 24

3.5 Kernel scalings using $H = h$ and $H = \sqrt{2}h$. The central image is not translation invariant. 26

3.6 A $\pi/6$ rotated filter with a scaling $H = 1/\cos(\pi/6)$ showing the invariant space. 27

3.7 Contour-line error plots (log) before and after filtering using a $\pi/4$ tensor product. 28

3.8 Contour-line error plots (log) before and after filtering using a $\pi/6$ tensor product filter. 29

3.9 Contour-line error plots (log) before and after filtering for different SIAC filters using $H = h$. 32

3.10 Surface error plots (log) and 1D slices of the error after filtering for the element centre along the entire field. 33

3.11 Footprints of two kernels with same error profiles corresponding to a $\theta-$rotated filter and the supplementary $\theta' = \pi - \theta$ 33

3.12 Pointwise error profiles as a function of the scaling and the rotation angle after postprocessing inside a particular element. 34

3.13 Local L^2 and L^∞ errors for a particular element as a function of the rotation angle and scaling. 35

3.14 Footprint of the mesh of two kernels with translation invariance property showing an alternative rotation with non orthogonal inner axis. 37

4.1 Illustration of an univariate B-Spline support along a line in $\mathbb{R}^2$. 42
4.2 Point-wise error profile slices along a horizontal and diagonal direction applying several Line Filters on the DG solution with initial condition $u_0(x, y) = \sin x$. 51
4.3 Point-wise error profile slices along several directions applying Line Filters on the DG solution with initial condition $u_0(x, y) = \sin x \cos y$. . . 53
4.4 Point-wise error profile slices along several directions applying Line Filters on the DG solution with initial condition $u_0(x, y) = \sin(x + y)$. . . 54
4.5 Contour line error plots (log) before and after applying Line Filters on the L^2 projection of the function $u(x, y) = \sin(x + y)$. 58
4.6 Contour line error plots (log) before and after applying the Tensor Proudct filter and the Line Filter on the DG solution to the problem $u_t + u_x + u_y = 0$. 60
4.7 Contour line error plots (log) before and after applying Line Filters on the DG solution with initial condition $u(x, y) = \sin(x + 3y)$. 63
4.8 Contour line error plots (log) before and after applying different line filters oriented using the mesh and flow direction respectively for the DG solution to the problem $u_t + 1.3u_x + 0.8u_y = 0$. 64
4.9 Contour line error plots (log) before and after applying Line Filters on the DG solution using a rectangular mesh. 66

5.1 Footprint of a rotated filter. 73
5.2 Footprint of an axis aligned filter highlighting the integration regions. . 73
5.3 Example of a kernel consisting of three B-Splines in each direction. . . 74
5.4 Illustration of the Ray Casting Algorithm *even-odd* rule. 76
5.5 The four possible intersection types of two segments on the 2D plane. . 77
5.6 Segment intersection using orientations. 78
5.7 Illustration of the Convex polygon construction. 79
5.8 Quadrature points in the standard regions. 81
5.9 Integration regions applying different filters over an uniform mesh. . . . 84
5.10 Computational times required to post-process one point applying different Tensor Product Filters and a Line Filter. 85
5.11 Integration regions applying a Tensor Product Filter and a Line Filter over a nonuniform mesh. 85
5.12 Integration regions applying a Tensor Product Filter and a Line Filter over a uniform triangular mesh. 86

6.1 Streamlines along CF1 before and after applying different symmetric Line Filters. 93

6.2 Streamlines along CF2 before and after applying different symmetric Line Filters. 94
6.3 Unfiltered and filtered streamlines with seed near a critical point computed using several ODE solvers. 96
6.4 Unfiltered and filtered streamlines using the RK2 solvert using $\mathbb{P}^2$ polynomials. 97
6.5 nfiltered and filtered streamlines using the RK3 solver and $\mathbb{P}^2$ polynomials. 97
6.6 Streamlines after applying higher order filters. 98
6.7 Streamlines using the $K_{\Gamma}^{(5,3)}$ filter along the $\pi/4$ and $3\pi/4$ directions. . 99
6.8 Comparison of the performance between the Cartesian axis Tensor Product Filter and the $\pi/4$-Line Filter. 101
6.9 Streamlines before and after applying the 2D Filter compared to the higher order line filter. 103
6.10 Streamlines before and after applying boundary Line Filters along the $\theta = \pi/4$ and $\theta = 3\pi/4$ directions. 106
6.11 Streamlines before and after applying boundary Line Filters along the $\theta = 5\pi/4$ and $\theta = 7\pi/4$ directions. 107
6.12 Location of the quadrature points using a linear reconstruction compared to their exact location on the curve. 108
6.13 Streamlines along CF1 before and after filtering along the streamlines. 113
6.14 Streamlines along CF2 before and after filtering along the streamlines. 114
6.15 Filtered streamlines comparing different types of boundary filters aligned with the flow. 115
6.16 Filtered streamlines comparing filtering along the streamline to boundary filtering using the mesh for the orientation. 116
6.17 Streamlines adding the time dimension highlighting the delay on the filtered curve. 118
6.18 Streamlines with different stop condition. 118

List of Tables

2.1 L^2 and L$^\infty$ errors and orders for the DG approximation and its divided differences . . . 10

2.2 Global L^2 errors and orders after applying three different kernels to the DG solution. . . . 16

3.1 L^2 and L^∞ results before and after applying a $\pi/4$-rotated filter over an uniform mesh. . . . 27

3.2 L^2 and L^∞ results before and after applying a $\pi/6$-rotated filter. 30

3.3 L^2 errors and orders comparing the $\pi/4$-rotated filter and the Cartesian axis filter. . . . 31

3.4 L^2 errors and orders before and after post-processing the applying filters with scaling $H = h$. . . . 32

4.1 L^2 errors and orders before and after applying several Line Filters to the L^2 projection of the function $u(x, y) = \sin(x)$. . . . 52

4.2 L^2 error and order before and after applying several Line Filters on the L^2 projection of the function $u(x, y) = \sin(x + y)$. . . . 59

4.3 L^2 errors and orders before and after applying several Line Filters on the DG solution to the problem $u_t + u_x + u_y = 0$. . . . 61

4.4 L^2 errors and orders before and after applying several Line Filters on the DG solution with initial condition $u(x, y) = \sin(x + 3y)$. . . . 62

4.5 L^2 errors and orders before and after post-processing taking the $3\pi/4$ direction and two additional orientations corresponding to the flow direction and its tangent respectively. . . . 65

4.6 L^2 error and order before and after line filtering the DG solution over a rectangular mesh. . . . 67

5.1 Summary of the number of operations required to compute the filtering convolution. . . . 84

5.2 Summary of the number of operations required to compute the filtering convolution for several mesh types and filters. . . . 86

6.1 Error analysis of the symmetric line filters during streamline computations. 95

6.2 Error analysis of the line and tensor product filters during streamline computations. . . . 102

6.3 Computational times for the Tensor Product filter and two $\pi/4$-Line filters of order 2 and 3 respectively. 104
6.4 Error analysis of the boundary filters during streamline computations. . 112
6.5 Computational times comparing Hermite and linear interpolation. . . . 112
6.6 Different techniques for estimating the error during streamline visualisation. 119

Acknowledgements

First and foremost, I would like to thank my supervisor Jennifer Ryan, for all she has done and is still doing for me but on top of everything, for always have believed in me. She has continuously inspired me and I look up to her, as a mathematician and as a person. Her intuition and passion for research has made me really enjoy developing this project together. Without her patience and support, I would have never been able to develop and finish this thesis. Thank you Jennifer, for always encouraging me to carry on. I am also immensely grateful to Mike Kirby. I had the pleasure to work close to him and visit him at SCI in two occasions and he has constantly guided me throughout my PhD. I strongly admire his mathematical intuition, enthusiasm and attitude towards any kind of problem. His knowledge and discussions helped enormously to overcome all the obstacles that I encountered during these three years.

I would also like to thank Mahsa Mirzargar, especially for always showing interest in my research. She was an excellent mentor during my visits to Utah and it was a pleasure to listen and follow her advice. Thanks to our research group: Thea Vuik, Daniel Frean, Xiong Meng and Xiaozhou Li, I really enjoyed our meetings. I wish to mention the Nektar++ team and thank Chris Cantwell and David Moxey, whom I met a while ago in Utah and since then, they have always provided support when I was struggling to adapt my code to the software. I should also thank Lawrence E. Coates, who put me in contact with Jennifer in the first place. I was very lucky to meet him.

I am very grateful for the good environment of our PhD office and for all the great times that we shared together, mathematical and not so mathematical. Many thanks to Davide for always listening with interest my research doubts and frustrations and to my dear Natasha, with whom I've done uncountable pub trips under the pretext "we are looking for inspiration". I wish to thank Tim Southon, who has always been keen on financing any kind of PhD students social event.

This thesis is dedicated to all my family. They mean everything to me. My brother, a true scientist, and my sister, who couldn't care less about this degree, let alone mathematics, but is the one who will celebrate most that I've finished. My mother, who admires and enjoys the crazy life that I live more than anyone does, constantly reminds us how far we can go and thanks to whom I never feel alone. My father, who always knows what is best for us and for his love for mathematics. He is the reason why I started this journey many years ago. Gracias a Sali también, que se que está orgullosa de mí y esperando que algún día vuelva a casa. I would also like to thank my housemates and all my friends. Explicitly, Antonis and Marco, who would be devastated if I didn't spell their names in this page. You guys have been very inspiring. Thanks for keeping me sane.

Finally, I wish to acknowledge the financial support provided by the Air Force Office of Scientific Research (AFOSR), Computational Mathematics program (Program Manager: Dr. Jean-Luc Cambier), under grant number FA 8655-13-1-3017.

Chapter 1

Introduction

Flow visualisation through particle tracking methods such as streamlines and streaklines is a common technique used to provide insight into fluid dynamics. Among the many techniques used for Computational Fluid Dynamics, Discontinuous Galerkin (DG) methods are one family of numerical schemes that allow for generating data for flow visualisation. They are robust, high order methods which can handle complicated geometries as well as effectively solve solutions containing shocks [14].

Field lines (streamlines, streaklines, pathlines, etc.) are mathematically described using Ordinary Differential Equations (ODEs) and there are many solvers such as the Runge-Kutta schemes, designed to find these curves numerically. Visualising DG solutions can be challenging; the numerical solution has low levels of continuity and most ODE solvers assume smooth field conditions. For discontinuous fields where smoothness can no longer be assumed, in order to obtain accurate field lines, adaptive methods are usually employed. Unfortunately, these methods add computational intensity since near critical regions, computing a new point usually requires a "pre-stage" that locates the discontinuity (*e.g.*, predictor-corrector methods) and effectively steps over it. Alternatively, one can introduce a filter to increase the levels of continuity and subsequently compute field lines through a simpler ODE solver. This thesis investigates a particular class of post-processor, *Smoothness-Increasing Accuracy-Conserving* (SIAC) filters, and its applications to flow visualisation for solutions obtained by DG methods.

DG schemes, like Finite Element (FEM) and Finite Volume (FVM) Methods, use a variational form to solve Partial Differential Equations (PDEs). However, unlike FEM that require global continuity, a DG solution is continuous only inside the elements. The solution across the element interface is controlled through a numerical flux that is only weakly continuous; as a result, the error exhibits high frequency oscillations. SIAC Filtering [38, 40, 41] is a post-processing technique employed to reduce the error oscillations and recover smoothness in the solution and its derivatives [36, 51, 53, 57]. It consists of convolving a B-Spline kernel at a particular point with the DG solution at final time. The filters were originally designed for accuracy enhancement of FEMs [2, 44] and later applied to DG [13]. The post-processor extracts the hidden "superconvergence" of these methods; for linear hyperbolic problems, the filtered solution is of order $2k+1$, where k denotes the degree of the polynomial space used for the DG approximation which is order $k+1$ convergent for *special* meshes [10,50] and of $k+\frac{1}{2}$ order for arbitrary meshes [11,30]. Hence, in addition to increasing the smoothness, for smooth initial data and linear problems, the filtered solution is generally more accurate than the DG solution.

The original filter used a symmetric kernel whose support was centred around the post-processing point and expanded equally in every direction. Today, there are several versions of these filters attempting to address issues related to domain boundaries and near-shock regions. Since beyond the computational domain there is no information, the symmetric kernel can not be implemented near the boundaries. Furthermore, for solutions containing shocks, taking information near the shock may produce an undesirable smooth region. Hence, alternative kernel versions were introduced, giving rise to one-sided [56] and position-dependent [36,63] SIAC filters, and more recently, the non-uniform knot based PSIAC filters [45]. These filters include a shifting parameter in the kernel that translates the support towards one direction. Therefore, points near the domain boundary can be post-processed by pushing the support towards the interior of the domain and one can filter points belonging to shock regions by translating the support away from the discontinuity.

There has been ongoing work on the application of SIAC filters for DG solutions to improve the flow conditions where streamlines are subsequently computed. The authors of [61] implemented a multidimensional filter, generated a whole whole new smooth field and then computed streamlines. They observed that when the field contained high discontinuous jumps, filtering resulted in a more efficient technique than applying an adaptive method. However, multidimensional filtering implies solving the integral of the convolution in several variables. In practise, the long computational times associated to these filters limits their applications to real-world problems. This issue was first tackled by [64]. They presented a numerical experiment along 2D fields applying a type of one dimensional filter along the streamline curves, saving computational costs and reducing the complexity of the filter implementation. However, the theoretical and numerical investigation into the effectiveness of this technique on typical test problems was not carried out.

This thesis attempts to build a solid bridge between the theoretical work on the filters and the applications during flow visualisation for optimal accuracy and smoothness enhancement. Previous theoretical work on post-processing with SIAC filters has mainly concentrated on extracting superconvergence. However, the theoretical error estimates give information on the convergence order but can not ensure error minimisation. Rather than seeking superconvergence, the purpose of this research is to answer questions such as: "given a DG field and a streamline seed, which type of filter should be applied at each point in order to obtain the most accurate streamline?". The foundations for proving superconvergence assume only smooth initial data and link the filter directly to the underlying mesh, restricting the choices on the area of the domain from which information is extracted. For example, the traditional multidimensional filter is built as a tensor product of univariate filters along each Cartesian axis and hence the support orientation is fixed. From a visualisation perspective, it is natural to question if orienting the filter with the flow direction and changing the support size plays a role in improving the quality of the filtered solution. Therefore, a new type of filter is presented: the rotated SIAC filters. These filters are no longer Cartesian coordinate aligned and have variable orientation. Furthermore, based on the "filtering along streamlines" approach in [64], a subfamily of these filters is derived: the *SIAC Line Filters*. This is a new and computationally efficient technique for post-processing multidimensional fields using lower dimensional filters. This family of filters transforms the 2D integral of the convolution into a line integral. Hence, from a computational point of view, the advantages are immediate. Furthermore, theoretical error estimates are given showing that it is possible to extract superconvergence for such filters. In addition, the post-processed solution is not only smoother but generally much more accurate. The low computational costs associated to these filters makes them a very attractive tool for the scientific community.

1.1 Contributions

This thesis has contributed towards the investigation of SIAC filters in view of improving the vector field conditions during flow visualisation. The research has lead to the discovery of a subfamily of these filters which whilst preserving the properties of traditional SIAC filtering, reduce significantly the computational costs. The SIAC Line filters open up new horizons for this type of post-processing technique with promising applications across the whole scientific community. The main contributions are:

- **Introducing rotations for tensor product SIAC filtering.** A robust formulation for tensor product SIAC filters has been developed which introducing a rotation in the kernel, allows the filter support to expand in different directions. This has extended the concept of multidimensional filtering that until now, always used a tensor product construction along the Cartesian axis. The mathematical formulation is given together with a range of numerical results studying the potential of rotated filters for error reduction and smoothness recovery from DG solutions. This is discussed in Chapter 3.

- **Development of the SIAC Line filters.** A subclass of rotated filters was derived which reduces the filtering convolution to a one-dimensional problem by post-processing along a line, thus avoiding the tensor product structure. This is achieved by transforming the filtering convolution into a line integral. Theoretical error estimates are given, proving $2k+1$ order for the filtered solution when applied to linear hyperbolic problems over uniform meshes. In addition, several numerical results were performed showing that through this low dimensional filtering, it is possible to recover smoothness as well as reduce the error from the DG solution. This is discussed in detail in Chapter 4 and these contributions have been reported in the submitted journal article: "*Multi-dimensional filtering:*

Reducing the dimension through rotation", Julia Docampo Sánchez, Jennifer K. Ryan, Mahsa Mirzargar and Robert M. Kirby, SIAM Journal of Scientific Computing (SISC), submitted in 2016.

- **Implementation of General SIAC filters.** A robust algorithm has been developed in order to implement line and tensor product filters over general meshes. A detailed discussion on the different challenges associated to the computations of SIAC filters is given as well as full details on how to derive the implementation. This includes a powerful algorithm designed to find, compute and sort intersection points between two overlapping structures which in this case are identified with the element interfaces of the DG mesh and the SIAC kernel breaks. This is discussed in Chapter 5.

- **Applications of the SIAC Line filters to streamline visualisation.** The SIAC Line filters were tested during streamline computations over DG fields containing singularities. The experiments showed how these filters successfully improve the quality of the visualisation, producing highly accurate streamlines that without filtering, diverge from the exact solution. Furthermore, the experiments revealed that the Line filter can match (and sometimes improve) the results compared to traditional tensor product filter. This is discussed in Chapter 6.

Chapter 2

Background

The theory of SIAC filtering for DG methods relies on the divided differences of the numerical solution. Using a piecewise polynomial basis of degree k, the numerical solution is typically of order $k+1$ under the L^2 norm in both the approximation and divided differences for linear hyperbolic equations [60]. However, DG solutions have "hidden" superconvergence. In [13] it was proven that the DG solution has $2k+1$ convergence in the negative-order norm for the approximation and the divided differences. SIAC filters exploit this fact and can achieve $2k+1$ order in the L^2 norm for the actual solution. In order to understand how one can extract superconvergence, this chapter begins by introducing the DG scheme together with theoretical error estimates.

2.1 DG Schemes

DG schemes were introduced by Reed and Hill, who in 1973 implemented them to solve the neutron transport equation [49]. Although originally designed for linear hyperbolic equations, these methods extended to diffusion and elliptic problems and today are used to solve non-linear combined problems such as the incompressible and compressible Navier-Stokes equations [1,33]. DG schemes for hyperbolic conservation laws have been studied in depth by [9,14]. However, for the development of theoretical error estimates, understanding the properties of the solution is necessary. Therefore, the theoretical analysis is typically done for the linear advection equation. This equation provides a simple model for exploring numerical schemes for hyperbolic problems, including initial discontinuous data or time developing shocks. Here, an overview of the scheme is given together with theoretical error estimates as well as a numerical example highlighting two important features: the superconvergence property and the oscillatory behaviour in the error profile.

Consider the linear hyperbolic problem:

$$\begin{cases} u_t + \sum_{i=1}^{d} A_i u_{x_i} + A_0 u = 0, \quad (\mathbf{x}, t) \in \Omega \times [0,T], \quad \Omega \text{ bounded}, \\ u(\mathbf{x}, 0) = u_0, \end{cases} \tag{2.1}$$

where $A_i,\ i = 1\dots,d$ are constant, $\mathbf{x} = (x_1, x_2, \dots, x_d)$ and u represents the advection of the conserved quantity. Furthermore, assume periodic boundary conditions. The numerical solution using a DG scheme is found using the Method-Of-Lines (MOL) approach, which in this case implies a spatial discretization but not temporal. Hence, the first step is to choose a suitable tessellation $\mathcal{T}(\Omega) = \sum e$ of the domain Ω and a piece-wise polynomial approximation space:

$$V_h^k = \left\{v \in L^2(\Omega) : v \in \mathbb{P}^k(e),\ \forall e \in \mathcal{T}(\Omega)\right\}.$$

Then, the DG solution is obtained using the variational form of Equation (2.1). It is the unique function $u_h \in V_h^k$ satisfying

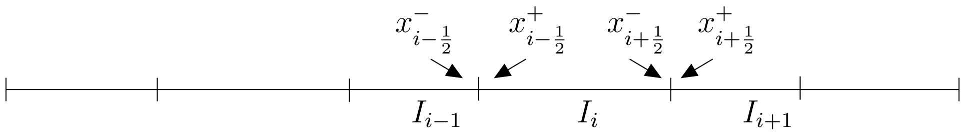
$$\int_e (u_h)_t v dx - \sum_{i=1}^{d} \left(\int_e A_i u_h(x,t) v_{x_i} dx \right) + \int_e A_0 u_h v dx + \sum_{i=1}^{d} \int_{\partial e} \widehat{A_i u_h} \cdot \mathbf{n} v dS = 0 \quad (2.2)$$

for all $v \in V_h^k$ and for every element of the tessellation. The term $\widehat{A_i u_h}$ refers to the numerical flux, the function enforcing weak continuity across the element interfaces, which is typically taken to be the upwind flux. In the following, this discretization is demonstrated for the linear advection equation.

Example 2.1.1. *One dimensional DG Scheme.*

$$\text{Consider the scalar problem} \begin{cases} u_t + u_x = 0, & x \in \Omega \quad t \in [0,T], \\ u_0(x) = u(x,0). \end{cases} \quad (2.3)$$

Define the mesh elements by $I_i = (x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}),\quad \Omega = \bigcup_{i=1}^{N} I_i$.



Imposing the upwind flux,

$$\hat{u}_{h_{i+\frac{1}{2}}} = u_h(x^-_{i+\frac{1}{2}}, t),$$

the DG formulation given in equation (2.2) becomes:

$$\sum_{i=1}^{N} \left[\frac{\partial}{\partial t} \int_{I_i} u(x,t) v_h dx = \int_{I_i} u(x,t) \frac{\partial v_h}{\partial x} dx - \hat{u}_{h_{i+\frac{1}{2}}} v^-_{h_{i+\frac{1}{2}}} + \hat{u}_{h_{i-\frac{1}{2}}} v^+_{h_{i-\frac{1}{2}}} \right]. \quad (2.4)$$

$$\text{Choose the Legendre basis:} \begin{cases} \phi_0(\xi) &= 1, \\ \phi_1(\xi) &= \xi, \\ \phi_{\ell+1}(\xi) &= \dfrac{2\ell+1}{\ell+1}\xi\phi_\ell(\xi) - \dfrac{\ell}{\ell+1}\phi_{\ell-1}(\xi), \quad \ell \geq 1, \end{cases}$$

to be the piece-wise polynomial approximation space V_h^k. The following two ideas will be used to carry out the approximation:

1. Legendre polynomials are orthogonal with respect L^2-norm for $-1 \leq x \leq 1$. Moreover

$$\phi_\ell(\pm 1) = (\pm 1)^\ell, \tag{2.5}$$

$$\int_{I_i} \phi_m(x)\phi_\ell(x)dx = \begin{cases} 0, & \text{if } \ell \neq m, \\ \frac{2}{2j+1}, & \text{if } \ell = m. \end{cases} \tag{2.6}$$

2. The change of variables:

$$\xi = \frac{2}{h_i}(x - x_i), \quad d\xi = \frac{2}{h_i}dx,$$

maps each DG element to the interval $[-1, 1]$.

$$\text{Let} \quad v_h(x) = \phi_\ell(\xi) \quad \text{and} \quad u_h(x,t) = \sum_{i=1}^{N}\sum_{\ell=0}^{k} u_i^{(\ell)}(t)\phi_\ell(\xi). \tag{2.7}$$

Imposing the properties of Legendre polynomials (equations (2.5) and (2.6)) in equation (2.4) and rearranging gives:

$$\sum_{i=1}^{N}\sum_{\ell=0}^{k}\left[\frac{\partial u_i^{(\ell)}(t)}{\partial t} = \frac{h}{2}\frac{2\ell+1}{2}\left(-\hat{u}_{h_{i-\frac{1}{2}}}(-1)^\ell + \hat{u}_{h_{i+\frac{1}{2}}} + \int_{I_i}\phi_\ell(\xi)\sum_{m=0}^{k}\frac{\partial\phi_m(x)}{\partial x}(x)dx\right)\right]. \tag{2.8}$$

The solution to this equation is found implementing a time marching scheme such as the SSP Runge-Kutta methods [21].

The theorem presented next presents superconvergent error estimates for the DG approximation for linear hyperbolic problems. Before, it is necessary to define the *negative order norm*:

$$\|v\|_{-\ell,\Omega} = \sup_{\phi\in\mathcal{C}_0^\infty(\Omega)}\frac{(v,\phi)_\Omega}{\|\phi\|_{\ell,\Omega}}, \quad \|\phi\|_{\ell,\Omega} = \left(\sum_{|\alpha|\leq\ell}\|D^\alpha v\|_\Omega^2\right)^{\frac{1}{2}} \text{ and } \ell > 0. \tag{2.9}$$

Here, $\|\cdot\|_{\ell,\Omega}$ is the norm associated to the Sobolev space $H^\ell(\Omega) = W_2^\ell(\Omega)$ and D^α denotes the differential operator. Furthermore, the notation ∂^α will be used to define the *(scaled) divided difference*:

$$\partial_h^\alpha = \partial_{h,1}^{\alpha_1}\partial_{h,2}^{\alpha_2}\cdots\partial_{h,d}^{\alpha_d}, \quad \partial_{h,j}f(x) = \frac{1}{h}\left(f(x+\frac{h}{2}e_j) - f(x-\frac{h}{2}e_j)\right), \tag{2.10}$$

$$\partial_{h,j}^{\alpha_j}f = \partial_{h,j}(\partial_{h,j}^{\alpha_j-1}f), \quad \alpha_j > 1,\ j = 1,\ldots,d. \tag{2.11}$$

Theorem 2.1.1. *(cf. Theorem 3.3 in [13]). Let $\Omega_0 \subset\subset \Omega$ and u be the solution to problem* (2.1) *with periodic boundary conditions. Assume that $u_0 \in L^2_{per}(\Omega, H^s(D\tilde{\Omega}))$, where $H^s(D\tilde{\Omega})$ is certain Sobolev space and*

$$L^2_{per}(\Omega) = \left\{f \in L^2_{loc}(\Omega) :\ f(x+\alpha) = f(x),\ \forall x \in \Omega,\ \alpha \in \mathbb{Z}^d\right\},$$

i.e., functions that are translation invariant by integer shifts. Under certain conditions, the DG approximation u_h to problem (2.1) *satisfies the following negative order norm error estimate:*

$$\|u(T) - u_h(T))\|_{-(k+1),\Omega_0} \leq Ch^{2k+1}. \tag{2.12}$$

Note 2.1.1. *This result is a particular case of the original theorem using the values from Table 3.4 in [13] corresponding to DG approximations.*

Finally, the error estimates for the divided differences of the DG solution are given below.

Theorem 2.1.2. *(cf. Theorems 3.3 & 3.4 in [13]). Let u and u_h be the exact and DG solutions respectively to problem* (2.1) *with periodic boundary conditions. For a uniform mesh, the following error estimates hold:*

$$\|\partial_h^\alpha(u(T) - u_h(T))\|_{0,\Omega} \leq Ch^{k+1} \tag{2.13}$$

in the L^2-norm and in the negative order norm:

$$\|\partial_h^\alpha(u(T) - u_h(T))\|_{-(k+1),\Omega} \leq Ch^{2k+1}. \tag{2.14}$$

Here k denotes the polynomial order used for the DG approximation and $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_d)$ a multi-index.

Note 2.1.2. *The divided differences are scaled by the mesh element size and the approximation space is translation invariant by shifts of $x \mapsto x \pm \frac{h}{2}$. Hence, the divided difference of the solution satisfies the PDE (taking $u = \partial_h u$ and $u_0 = \partial_h u_0$) and the same error estimates than those for the DG approximation hold.*

Finally, the following Lemma is introduced which allows us to switch between the L^2 and the negative-order norms.

Lemma 2.1.1. *(Bramble and Schatz [2]). Let $\Omega_0 \subset\subset \Omega_1 \subset\subset \Omega$, Ω bounded domain in $\mathbb{R}^d$ and s be an arbitrary but fixed non-negative integer. Then, for $u \in H^s(\Omega_1)$, there is a constant C such that*

$$||u||_{0,\Omega_0} \leq C \sum_{|\alpha|\leq s} \|D^\alpha u\|_{-s,\Omega_1}. \tag{2.15}$$

Now a numerical result is given for the scalar problem (2.3).

Example 2.1.2. *Let u_h be the DG approximation to the problem:*

$$\begin{cases} u_t + u_x = 0 \quad (t,x) \in (0,T) \times [0,1] \\ u_0(x) = \sin(2\pi x), \end{cases} \tag{2.16}$$

with final time $T = 2$ and using an uniform mesh. Table 2.1 shows the global errors and orders illustrating how both the DG solution and its divided differences attain $k+1$ convergence (both in the L^2 and L^∞ norms) when using $\mathbb{P}^k$ polynomials. For the highest degree ($k = 3$) and finest mesh, both the convergence order and error are destroyed but this is due to round off errors arising from using double precision. Figure 2.1 shows another important feature of these schemes: the spurious oscillations in the error profile resulting from the weak continuity at the element interface.

The underlying mechanism of SIAC filters transforms the differential operator D^α in Lemma 2.1.1 into a divided differences operator. This allows for using the negative order estimate of Theorem 2.1.2 for the filtered solution, giving $2k+1$ accuracy in the L^2 norm [13]. Furthermore, these filters reduce the oscillations in the error. In the next section, details on this post-processor are given together with numerical examples showing superconvergence, smoothness recovery and error reduction.

N	$\boldsymbol{u_h}$				$\boldsymbol{\partial_h u_h}$				$\boldsymbol{\partial_h^2 u_h}$				$\boldsymbol{\partial_h^3 u_h}$			
	L^2 error	Order	L^∞ error	Order	L^2 error	Order	L^∞ error	Order	L^2 error	Order	L^∞ error	Order	L^2 error	Order	L^∞ error	Order
$\mathbb{P}^1$																
20	4.6e-03	-	1.1e-02	-	2.8e-02	-	5.0e-02	-	-	-	-	-	-	-	-	-
40	1.1e-03	2.08	3.2e-03	1.82	6.4e-03	2.10	1.2e-02	2.01	-	-	-	-	-	-	-	-
80	2.7e-04	2.02	8.5e-04	1.92	1.6e-03	2.03	3.4e-03	1.88	-	-	-	-	-	-	-	-
160	6.6e-05	2.01	2.2e-04	1.96	3.9e-04	2.01	8.8e-04	1.94	-	-	-	-	-	-	-	-
320	1.7e-05	2.00	5.5e-05	1.98	9.7e-05	2.00	2.2e-04	1.97	-	-	-	-	-	-	-	-
$\mathbb{P}^2$																
20	1.1e-04	-	3.7e-04	-	5.2e-04	-	1.2e-03	-	4.2e-03	-	1.4e-02	-	-	-	-	-
40	1.3e-05	3.00	4.6e-05	2.99	6.5e-05	3.00	1.5e-04	2.99	5.3e-04	2.99	1.8e-03	2.98	-	-	-	-
80	1.7e-06	3.00	5.8e-06	3.00	8.2e-06	3.00	1.9e-05	3.00	6.6e-05	3.00	2.3e-04	3.00	-	-	-	-
160	2.1e-07	3.00	7.2e-07	3.00	1.0e-06	3.00	2.3e-06	3.00	8.2e-06	3.00	2.9e-05	3.00	-	-	-	-
320	2.6e-08	3.00	9.0e-08	3.00	1.3e-07	3.00	2.9e-07	3.00	1.0e-06	3.00	3.6e-06	3.00	-	-	-	-
$\mathbb{P}^3$																
20	2.1e-06	-	6.0e-06	-	9.5e-06	-	1.7e-05	-	8.1e-05	-	2.4e-04	-	3.7e-04	-	6.5e-04	-
40	1.3e-07	4.00	3.8e-07	3.99	5.9e-07	4.00	1.0e-06	3.99	5.1e-06	3.99	1.5e-05	3.98	2.3e-05	3.99	4.1e-05	3.98
80	8.1e-09	4.00	2.4e-08	4.00	3.7e-08	4.00	6.6e-08	3.99	3.2e-07	4.00	9.4e-07	3.99	1.5e-06	4.00	2.6e-06	3.99
160	5.0e-10	4.00	1.5e-09	4.00	2.3e-09	4.00	4.1e-09	3.99	2.0e-08	4.00	6.2e-08	3.93	2.2e-07	2.72	1.1e-06	1.28
320	3.2e-11	3.99	9.4e-11	3.99	1.5e-10	3.98	2.7e-10	3.93	4.0e-09	2.33	2.1e-08	1.58	2.1e-06	-3.24	9.7e-06	-3.18

Table 2.1: L^2 and L^∞ errors and orders for the DG approximation and its divided differences for the solution to Problem (2.16).

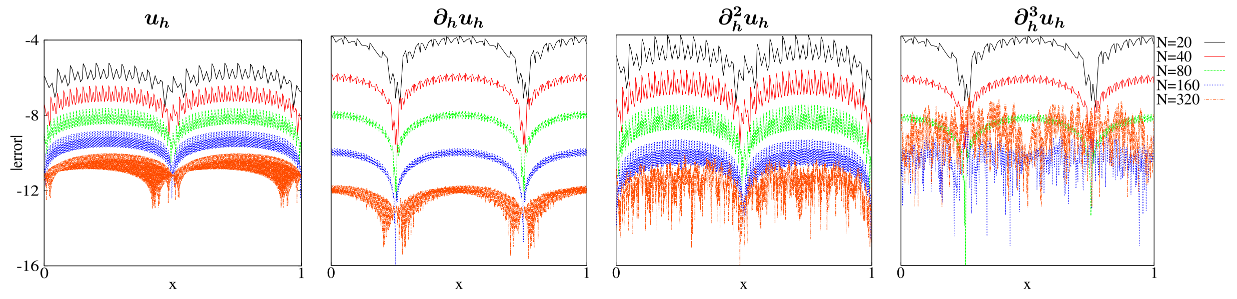


Figure 2.1: Point-wise errors (log) profiles of the DG approximation and its divided differences for the solution to Problem (2.16) using several meshes and $\mathbb{P}^3$ polynomial basis.

2.2 SIAC Filters

SIAC filtering finds its foundations on a class of post-processor originally developed for Finite Element Methods by Bramble and Schatz [2]. They showed how to recover smoothness and even increase the order by convolving the solution with a filter acting as a local averaging operator. Later, Cockburn, Luskin, Shu and Süli [13] applied them to DG schemes for hyperbolic problems and today they are known as Smoothness-Increasing-Accuracy-Conserving (SIAC) filters [52]. A very detailed description of the properties of these filters, implementation details and applications to different mesh types can be found in [39, 40, 42, 43, 52].

Let $r = 2k$ and $\ell = k + 1$, where k denotes the degree of the DG approximation. The post-processor is a continuous convolution:

$$u_h^\star(x,T) = \int_{-\infty}^{\infty} K_H^{(r+1,\ell)}(x-y)u_h(y,T)\,dy, \quad x \in \Omega, \tag{2.17}$$

where u_h denotes the DG solution at final time and the kernel (symmetric) is a linear combination of central B-Splines:

$$K^{(r+1,\ell)}(\eta) = \sum_{\gamma=0}^{r} c_\gamma \psi^{(\ell)}\left(\eta - \left(\frac{r}{2} - \gamma\right)\right). \tag{2.18}$$

Here, γ denotes the B-Splines centres. The kernel subindex H in equation (2.17) acts as a scaling factor , *i.e.*, $K_H(x-y) = \frac{1}{H}K\left(\frac{x-y}{H}\right) = \frac{1}{H}K(\eta)$. To give an idea of the filter size, for uniform meshes, the usual scaling choice is $H = h$, where h denotes the element size used for the DG approximation. The superindexes $(r+1,\ell)$ indicate the number of B-Splines employed to build the kernel $(r+1)$ and the spline order (ℓ). Basis Splines (B-Splines) are local functions providing maximum approximation order with minimum support. They are computationally very attractive since they can be calculated using recurrence formulas. Define a knot sequence by $\boldsymbol{t} = (t_i)$, made of non-decreasing real numbers. The normalised i^{th} B-Spline is given by:

$$B_{i,1,\boldsymbol{t}}(x) = \begin{cases} 1, & t_i \le x < t_{i+1}, \\ 0, & \text{otherwise} \end{cases} \quad \text{if} \quad \ell = 1, \tag{2.19}$$

$$B_{i,\ell,\boldsymbol{t}}(x) = \frac{x - t_j}{t_{i+\ell-t_i}}B_{i,\ell-1}(x) + \frac{t_{j+\ell} - x}{t_{i+\ell-t_i}}B_{i+1,\ell-1}(x), \quad \text{if} \quad \ell > 1. \tag{2.20}$$

The central B-Splines are the particular case consisting of the uniform knot sequence:

$$\boldsymbol{t} = -\frac{\ell}{2}, -\frac{\ell-2}{2}, \dots, \frac{\ell-2}{2}, \frac{\ell}{2}. \tag{2.21}$$

Here, they will be identified with the letter $\psi^{(\ell)} = B_{0,\ell,t}(x)$. These splines can be defined using the following recurrence:

$$\psi^{(1)}(x) = \chi_{[-1/2,1/2)}(x), \tag{2.22}$$

$$\psi^{(\ell)}(x) = \psi^{(\ell-1)} \star \psi^{(1)}(x), \quad \ell > 1 \tag{2.23}$$

$$= \frac{1}{\ell-1}\left(\left(\frac{\ell}{2} + x\right)\psi^{(\ell-1)}\left(x + \frac{1}{2}\right) + \left(\frac{\ell}{2} - x\right)\psi^{(\ell-1)}\left(x - \frac{1}{2}\right)\right). \tag{2.24}$$

Moreover, they have the following property for the derivatives:

$$d^\alpha \psi^{(\ell)} = \partial^\alpha \psi^{(\ell-\alpha)}, \quad \partial^\alpha = \alpha^{th} \text{ divided difference } (\partial^\alpha_{h=1} \text{ in eq. (2.10)}). \tag{2.25}$$

A complete description on these spline functions can be found in [17] and [59].

Note 2.2.1. *The letter ℓ used to denote the B-Splines order will appear frequently in this thesis and will always correspond to $\ell = k+1$ unless otherwise specified. Furthermore, the notation $r = 2k$ will be commonly used to specify the number of B-Splines used to build the kernel.*

Finally, the kernel coefficients, c_γ, dictate each of the B-Spline weights and are determined by imposing the polynomial reproduction property:

$$K^{(r+1,\ell)} \star x^p = x^p, \ p = 0, \dots, r. \tag{2.26}$$

The problem of computing these kernel coefficients is discussed later in chapter 5, which is dedicated to the implementation of SIAC filters and the computational challenges associated to them. Figure 2.2 shows the B-Spline functions for several degrees together with two kernels corresponding to equation (2.18) using $k = 1, 2$ respectively.

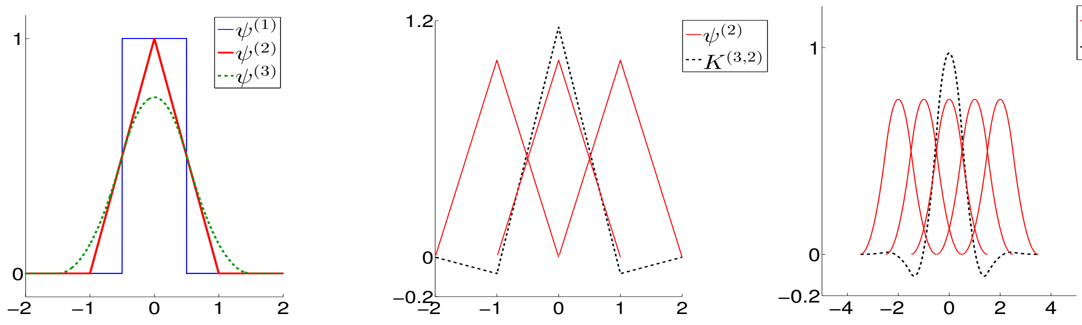


Figure 2.2: B-Splines (left) for different degrees and two symmetric SIAC kernels (centre and right).

The following Theorem provides error estimates for the post-processed DG solution.

Theorem 2.2.1. *(Cockburn, Luskin, Shu, and Süli [13]) Under the same conditions in Theorem 2.1.2 and if $\Omega_0 + 2supp\left(K_h^{(2k+1,k+1)}\right) \subset\subset \Omega_1 \subset \Omega$, then for $H = h$ (h mesh size):*

$$\left\| u - K_h^{(2k+1,k+1)} \star u_h \right\|_{0,\Omega_0} \leq Ch^{2k+1}. \tag{2.27}$$

Note 2.2.2. *Using Property* (2.25) *together with Lemma 2.1.1 allows to transform the differential operator into a divided difference operator and subsequently applying Theorem 2.1.2 over the filtered solution, giving the error estimate in Theorem 2.2.1.*

2.3 General SIAC Filters

Here, an outline of the development of SIAC filters is given. The kernel presented in equation (2.18) is symmetric in the sense that the support is centred around the post-processing point and expands equally in all directions. This can be appreciated in the kernels from Figure 2.2, which are centred at zero. The aforementioned position-dependent SIAC filters include a shifting parameter in the B-Splines, translating the kernel support towards one direction when necessary. These type of filters were first applied in [56], and are called RS filters. Unfortunately, they noticed that near the boundaries, where the filter support is most shifted, accuracy dropped. To overcome this limitation, [63] introduced the SRV filter. This variation of the RS filter consists of doubling the amount of splines employed on the position-dependent kernel and introduce a function that allows a smooth transition towards the original symmetric kernel as soon as the latter one can be implemented. In [28], superconvergent error estimates were developed for the SRV filter for multidimensional problems, proving global $2k+1$ accuracy under the L^2 norm and order $\min\{2k+1, 2k+2-\frac{d}{2}\}$ (d being the field dimension) under the L^∞ norm. However, superconvergence is achieved at the expense of increasing computational costs. Furthermore, increasing the number of splines affects the magnitude of the error since the constant in the error in Theorem 2.2.1,

$$\|u - u^\star\|_0 \leq Ch^{2k+1},$$

includes a factor that depends on the kernel coefficients (see the proof in [13]):

$$C_1 = \sum_{\gamma=0}^{r} |c_\gamma|,$$

which grow as the number of splines increase. A detailed discussion on the effects of this constant can be found in [35, Ch. 2]. In order to overcome this limitations, [55] developed the *new efficient position-dependent* SIAC filters. Rather than doubling the amount of splines near the boundaries, these filters introduce a general spline. Such spline is added at the filter support boundaries but does not increase the support size. Notice that the SRV filter has $2k$ extra splines, hence the support is wider. This new filter is globally $2k+1$ superconvergent for the linear case (*i.e.*, $k=1$) and of order $k+1$ if $k>1$, *i.e.*, it does not destroy the DG accuracy. Furthermore, the numerical results show that the filtered solution has generally lower error than the original DG and actually can outperform the SRV filter (in terms of the error). For more details on the filters performance and comparison between each, see [35].

The shift in the kernel support is achieved by translating the centre of the B-Splines towards one direction. Let $(r+1, \ell)$ be the number of B-Splines and their degree respectively. The position-dependent kernel is given by the formula:

$$K^{(r+1,\ell)}(\eta) = \sum_{\gamma=0}^{r} c_\gamma \psi^{(\ell)}(\eta - x_\gamma(\lambda)) + c_{r+1} b^{(\ell)}(\underbrace{x - H\eta}_{=y}), \quad x_\gamma(\lambda) = -\frac{r}{2} + \gamma + \lambda, \quad (2.28)$$

where η is the scaled variable, *i.e.* $\eta = \frac{x-y}{H}$, and the shift function depends on the location of the post-processing point x:

$$\lambda(x) = \begin{cases} \min\left\{0, -\dfrac{r+\ell}{2} + \dfrac{\overline{x} - x_L}{H}\right\}, & x \in \left[x_L, \dfrac{x_L + x_R}{2}\right) \\ \max\left\{0, \dfrac{r+\ell}{2} + \dfrac{\overline{x} - x_R}{H}\right\}, & x \in \left(\dfrac{x_L + x_R}{2}, x_R\right]. \end{cases} \tag{2.29}$$

Here, x_L and x_R denote the domain boundaries respectively. The central B-Splines, $\psi^{(\ell)}(\cdot)$, are calculated in the usual way (see equations (2.22)-(2.24)) and the general spline, $b^{(\ell)}(\cdot)$, is given by:

$$b^{(\ell)}(y) = \begin{cases} \left(y - \left(\dfrac{y - x_L}{h} - 1\right)\right)^{\ell-1} & \text{if} \quad \dfrac{y - x_L}{h} - 1 \leq y \leq \dfrac{y - x_L}{h}, \\ \left(\left(\dfrac{y - x_R}{h} + 1 - y\right)\right)^{\ell-1} & \text{if} \quad \dfrac{y - x_R}{h} \leq y \leq \dfrac{y - x_R}{h} + 1, \\ 0 \quad \text{otherwise.} \end{cases} \tag{2.30}$$

Note 2.3.1. *The RS and SRV kernels can be written using equation* (2.28) *excluding the last term, corresponding to the general spline* $b^{(\ell)}$.

Note 2.3.2. *The kernel given in equation* (2.28) *will be identified as XLi kernel (filter) throughout this thesis.*

Figure 2.3 shows an example of three boundary filters using the RS, SRV and the new efficient kernels respectively.

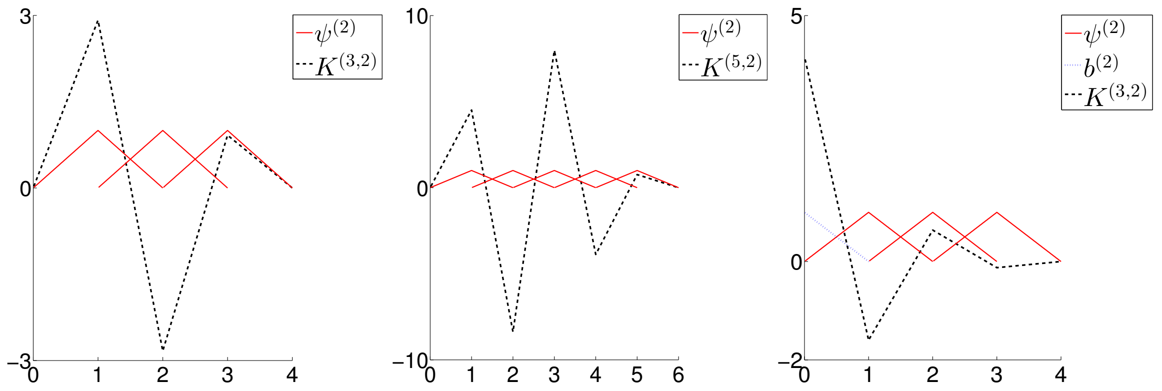


Figure 2.3: Three different types of boundary filters using 3 B-Splines (left), 5 B-Splines (centre) and the XLi kernel using 3 B-Splines plus a general spline ($b^{(2)}$).

Figure 2.4 and Table 2.2 show the numerical results after filtering the DG solution to Problem (2.16). Three different types of filters were applied; a symmetric filter, implemented by assuming periodic boundary conditions, a position-dependent filter using the kernel given in equation (2.28) and a boundary filter. The latter one was obtained through equation (2.28) imposing $\lambda(x) = \frac{r+\ell}{2}$, *i.e., every point was assumed to be a boundary point.* The global results in Table 2.2 show how for the linear approximation, both the symmetric and position-dependent filters give the same errors and orders. However, as the polynomial order increases, using a symmetric filter along the entire field results in better post-processing: the global and point-wise errors are most

	Before post-processing		After post-processing					
			Symmetric		Postition Dependent		Boundary	
N	L^2 error	Order	L^2 error	Order	L^2 error	Order	L^2 error	Order
$\mathbb{P}^1$								
20	4.6e-03	NA	2.0e-03	NA	2.0e-03	NA	3.4e-03	NA
40	1.1e-03	2.1	2.4e-04	3.0	2.4e-04	3.0	6.7e-04	2.3
80	2.7e-04	2.0	3.0e-05	3.0	3.0e-05	3.0	1.6e-04	2.1
160	6.6e-05	2.0	3.8e-06	3.0	3.8e-06	3.0	3.9e-05	2.0
$\mathbb{P}^2$								
20	1.1e-04	NA	4.1e-06	NA	1.2e-05	NA	9.0e-05	NA
40	1.3e-05	3.0	1.0e-07	5.4	5.5e-07	4.5	9.4e-06	3.3
80	1.7e-06	3.0	3.1e-09	5.0	4.8e-08	3.5	1.2e-06	3.0
160	2.1e-07	3.0	1.6e-10	4.3	4.2e-09	3.5	1.5e-07	3.0
$\mathbb{P}^3$								
20	2.1e-06	NA	7.3e-08	NA	2.3e-06	NA	8.2e-06	NA
40	1.3e-07	4.0	6.5e-10	6.8	4.2e-09	9.1	7.8e-08	6.7
80	8.1e-09	4.0	4.7e-11	3.8	4.8e-11	6.4	4.4e-09	4.1
160	5.0e-10	4.0	5.8e-12	3.0	5.8e-12	3.0	2.8e-10	4.0

Table 2.2: Global L^2 errors and orders after applying three different kernels to the DG solution to Problem (2.16).

reduced and the filter successfully eliminates the oscillations. If periodic boundary conditions cannot be assumed, the position-dependent filter should be implemented. The error plots (central images) show how the oscillations are fully eliminated towards the domain centre where the filter is essentially symmetric and are significantly reduced towards the boundary. Furthermore, there is a clear error reduction compared to the DG solution, both globally and locally. The boundary filter on the other hand, is neither able to raise the DG convergence order or reduce the error. Furthermore, the error plots in Figure 2.4 show how this filter is unable to recover smoothness.

2.3.1 Discussion

SIAC filters were designed both for smoothness recovery and superconvergence extraction. In addition, the filtered solution is generally more accurate than the original one, provided the DG solution is well resolved. This can be observed both in the pointwise error profile plots in Figure 2.4 and in the global results from Table 2.2. The numerical experiments suggest that greatest error reduction occurs when applying a symmetric kernel except for the $\mathbb{P}^1$ case; the position-dependent filter using the kernel from equation (2.28) (including the general B-Spline $b^{(\ell)}$), gives exactly the same results as applying a fully symmetric kernel. On the other hand, for higher polynomial degrees, when the filter is no longer symmetric, the position-dependent kernel reduces the oscillations but does not remove them completely compared to the fully symmetric kernel. The boundary filter does not raise the convergence order nor recover smooth-

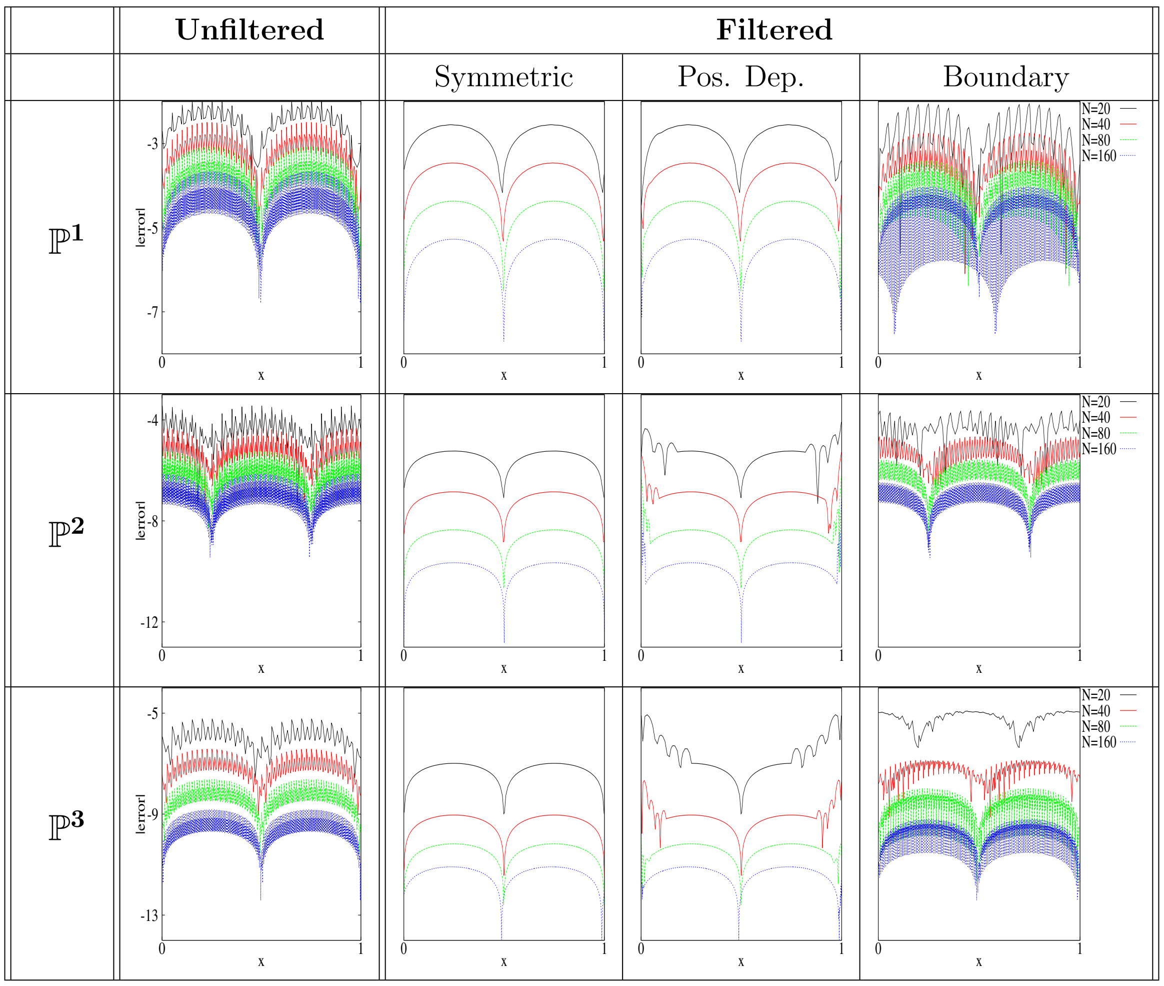


Figure 2.4: Point-wise errors (log) before and after filtering the DG solution to Problem (2.16) for several meshes and polynomial degrees using three different kernel types.

ness. Furthermore, the global errors shown in Table 2.2 for this filter indicate that although the new solution has a lower error than the original one, the difference is almost negligible. However, these filters were designed to improve the errors at the boundary and not globally. In the last column of Figure 2.4, it can be seen that near the boundaries, the filter successfully reduces the magnitude of the errors from the original DG solution. This case was included because it relates back to streamline visualisation; the one-dimensional filter from [64] consisted of a boundary filter implemented along the streamline curve. The idea behind this type of post-processing is that the filter support expands downstream along the streamline from the evaluating point. In this way, the filter uses information from the “true” curve, corresponding to previously computed streamline points. Since for streamline visualisation, error reduction is more important than superconvergence, Figure 2.4 gives a better insight into the filter performance. The error profile of the boundary and symmetric filters indicate that the kernel should stay as symmetric as possible. This seems to be in disagreement with the “filtering along the streamlines” approach. However, this numerical example corresponds to a one-dimensional field so it can not be concluded that a boundary filter should be avoided for multidimensional fields, where for example, the flow direction may become relevant.

2.4 Divided Differences & Translation Invariance

DG methods have superconvergent properties under the negative-order norm. SIAC filters use the "hidden" information under this norm to attain the same order of accuracy but in the L^2 norm. However, theoretical error estimates of SIAC filters are strongly linked not only to the DG solution itself, but to its divided differences as well. Lemma 2.1.1 provides a L^2 bound of the DG approximation in terms of its derivatives and a negative-order norm. The post-processor makes use of the B-Spline properties to transform the differential operator into a divided difference. Theorem 2.1.2 shows how for linear hyperbolic problems, the divided differences have $2k+1$ order of accuracy under the negative-order norm. The superconvergent error estimates from Theorem 2.2.1 are obtained by combining Lemma 2.1.1 and Theorem 2.1.2 together with the kernel divided differences properties.

The kernel scaling plays a major role for allowing switching from the derivatives to the divided difference operator. Recall that the (scaled) B-Splines are defined by

$$\psi_H^{(\ell)}(x) = \frac{1}{H}\psi^{(\ell)}\left(\frac{x}{H}\right),$$

and differentiation of this B-Spline gives the scaled divided difference:

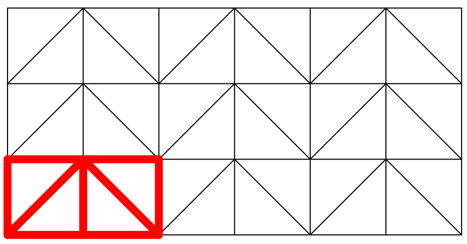
$$\frac{d^\alpha \psi_H^{(\ell)}(x)}{dx^\alpha} = \partial_H^\alpha \psi_H^{(\ell-\alpha)}(x).$$

For uniform meshes, the scaling $H = mh$ where $m \in \mathbb{Z}^+$ and h is the DG mesh size, the error estimate for the divided differences,

$$\|\partial_H^\alpha(u-u_h)\|_{-(k+1),\Omega} \leq CH^{2k+1},$$

still holds because it preserves the mesh translation invariance. For general nonuniform meshes, the theoretical estimates for the divided differences give only $k+1-\alpha$ order and determining an optimal scaling becomes very complicated. This is studied in detail in [35, Ch. 4]. In [32], a numerical study on the translation invariance property was carried out from a geometrical perspective. Several mesh types were proposed consisting of elements that when assembled together in certain groups, produce a translation invariant space in terms of the superelements. Examples of such meshes are the Chevron or the Union Jack meshes shown in Figure 2.5. Their numerical results consistently showed that selecting the scaling using the characteristic length of the superelement lead to optimal results.

Another important limitation for the development of superconvergent theoretical estimates is the nature of the hyperbolic problem. For linear hyperbolic equations, the same negative-order estimates for the DG solution hold for the divided differences. However, this does not hold for variable coefficient or non-linear hyperbolic equations. In this case, it is necessary to develop error estimates on the divided differences in both norms. In [29], superconvergent error estimates for the DG solution were extended to non-linear hyperbolic conservation laws. The authors proved how the DG solution can


(a) Chevron Mesh

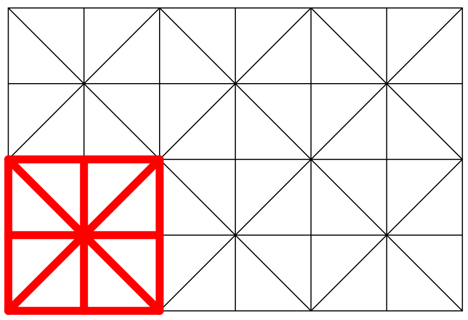

(b) Union-Jack Mesh

Figure 2.5: Two meshes highlighting the superelement that also forms a uniform mesh.

attain $2k + m$ order in the negative order norm, where $0 \leq m \leq 1$ and depends on the numerical flux. However, they did not extend the proof for the divided differences of the solution. On the other hand, they provided numerical results showing $2k + 1$ order for the filtered solution. Recently, [37] proved for one-dimensional non-linear scalar hyperbolic problems, superconvergence of SIAC filters for uniform meshes of at least $\frac{3}{2}k + 1$ order in the L^2 norm. For the first time, theoretical error estimates of the divided differences were developed, giving $2k + \frac{3}{2} - \frac{\alpha}{2}$ order in the negative-order norm. Furthermore, the authors show that is possible to prove $2k+1$ superconvergence for variable coefficient hyperbolic equations. Unfortunately, the proofs do not extend naturally to multidimensional problems so error estimates for such cases remain still unproven.

2.5 Summary

Combining SIAC filters with DG approximations results in smooth fields with high order of accuracy. This can be exploited during vector field visualisation, enhancing the quality of the field data used, for example, by a streamline solver. The symmetric filter seems optimal in terms of smoothness recovery, superconvergence extraction and error reduction. The numerical results show that position-dependent filters are globally superconvergent but on a local basis, they are still outperformed by symmetric filters both in terms of error reduction and smoothness recovery. Although theoretical error estimates address order of accuracy, applications of these filters should focus on smoothness recovery and seek error reduction on top of superconvergence. In addition, on a local basis, boundary filters should match the performance of the symmetric filter in terms of error reduction in order to apply them during flow visualisation.

Chapter 3

Tensor Product Rotated Filters

Multidimensional SIAC filters have traditionally been implemented using tensor product filters along each of the Cartesian axes. This configuration, a natural extension of the one dimensional case, allows for developing theoretical error estimates both for uniform and nonuniform meshes [13]. The foundations for proving superconvergence assumes only smooth initial data and links the filter directly to the underlying mesh. For uniform meshes, provided the kernel scaling is of the form $H = mh$, where $m \in \mathbb{Z}$ and h is the mesh size, it is possible to show $2k+1$ accuracy when using $\mathbb{P}^k$ polynomials for the DG approximation. However, as soon as the mesh uniformity assumption drops, finding a suitable scaling becomes complicated. A detailed theoretical discussion on the kernel scaling and nonuniform meshes can be found in [35, Ch. 4] and [15] provides a numerical study.

In this Chapter, a variant of the multidimensional tensor product filter is presented. It consists of introducing a rotation in the kernel, hence the name rotated filters. This idea comes from practical applications of SIAC filters. In terms of robustness, the Cartesian axis aligned tensor product set-up is restrictive. There is only one possible choice for the kernel support: a box aligned with the Cartesian axis as shown in Figure 3.1. This presents limitations for the choices on the area of the domain from which information is extracted. The question arises whether changing the direction in which information was filtered could improve the results. For example, if the post-processor is used during flow visualization, it is reasonable to ask whether keeping the kernel aligned with the mesh is more relevant than aligning the kernel with the flow direction. Figure 3.2 illustrates an example of an alternative support configuration. In this figure, the filter orientation is obtained from two consecutive streamline points. In this way, the filter is taking information from the vector field as close to the curve as possible.

Previous work on post-processing with SIAC filters has mainly concentrated on extracting superconvergence. Theorem 2.2.1 gives information on the convergence order but does not say anything about the value of the constant in front of the error:

$$\|u - K_h^{(2k+1,k+1)} \star u\|_{0,\Omega_0} \leq Ch^{2k+1}.$$

Thus, *ensuring superconvergence does not necessarily imply ensuring error minimization.* This will be stressed later in the numerical experiments. Furthermore, these estimates assume a filter with constant scaling and a tensor product structure aligned with the domain axis. On the other hand, the concept of a rotated SIAC filter presented here, suggests a filter with variable orientation and size. In addition, rather than seeking global superconvergence, the rotated SIAC filter seeks local optimal accuracy. Unfortunately, there is no theory behind the local performance of the filter, even for the original Cartesian aligned one. Therefore, before attempting to develop theoretical error estimates, the filter performance is first tested numerically. The numerical experiments are not conclusive but give insight into the behaviour of the filters. This Chapter begins by introducing and discussing the technical details of the rotated filter, both mathematical and computational. Then a qualitative study is done based on numerical experiments, testing combinations of rotations and scalings on a local and global basis.

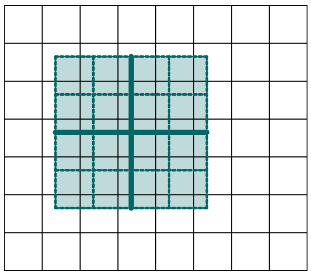


Figure 3.1: Support of a Cartesian axis aligned filter projected onto a uniform quadrilateral mesh.

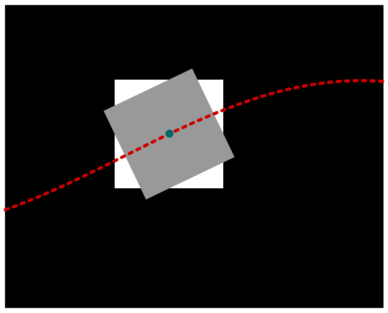


Figure 3.2: Comparison of the support of the Cartesian axis aligned filter (white box) with a rotated filter (gray box) aligned with the streamline direction (red dashed curve).

3.1 Mathematical Formulation

Before introducing the formula for the convolution of the rotated filter, a short review of basis and coordinate systems is given.

Definition 3.1.1. *(Basis, coordinate systems and change of basis matrices [48, Ch. 6]).*

- *Let $\mathcal{B} = \{u_1, u_2, \ldots, u_n\}$ be a subset of a vector space V. $\mathcal{B}$ is a* ***basis*** *for V if it satisfies the following properties:*

1. $\mathcal{B}$ spans V

2. $\{u_1, u_2, \ldots, u_n\}$ are linearly independent.

Any vector $\mathbf{v} \in V$ can be written as $\mathbf{v} = c_1u_1 + c_2u_2 + \ldots + c_nu_n$ and the values $\{c_1,\ c_2, \ldots,\ c_n\}$ denote its coordinates in the basis $\mathcal{B}$ which can be given as a coordinate vector:

$$[v]_{\mathcal{B}} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}.$$

Theorem 3.1.1. *Let $\mathcal{B}_1 = \{u_1, u_2, \ldots, u_n\}$ and $\mathcal{B}_2 = \{v_1, v_2, \ldots, v_n\}$ be two bases for a vector space V. Then, the* ***change-of-basis matrix*** *from $\mathcal{B}_1$ to $\mathcal{B}_2$, $P_{\mathcal{B}_2 \leftarrow \mathcal{B}_1} [\mathbf{x}]_{\mathcal{B}_1}$ is defined by:*

$$P_{\mathcal{B}_2 \leftarrow \mathcal{B}_1} = \left[[u_1]_{\mathcal{B}_2} \, [u_2]_{\mathcal{B}_2} \cdots [u_n]_{\mathcal{B}_2} \right] \tag{3.1}$$

and it satisfies:

a) $P_{\mathcal{B}_2 \leftarrow \mathcal{B}_1} [\mathbf{x}]_{\mathcal{B}_1} = [\mathbf{x}]_{\mathcal{B}_2} \quad \forall \mathbf{x} \in V$.

b) $P_{\mathcal{B}_2 \leftarrow \mathcal{B}_1}$ is unique and invertible. Furthermore, $(P_{\mathcal{B}_2 \leftarrow \mathcal{B}_1})^{-1} = P_{\mathcal{B}_1 \leftarrow \mathcal{B}_2}$.

The proof of this theorem can be found in [48, Ch. 6.3].

3.1.1 Rotated Convolution

The original 2D filter is defined by:

$$u^\star(\overline{x}, \overline{y}) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} K_{H_x}^{(r+1,\ell)}(\overline{x} - x) K_{H_y}^{(r+1,\ell)}(\overline{y} - y) u_h(x, y) dx dy \tag{3.2}$$

and each (symmetric) kernel is defined in the usual way:

$$K_H^{(r+1,\ell)}(\eta) = \sum_{\gamma=0}^{r} c_\gamma \psi^{(\ell)} \left(\eta - \left(-\frac{r}{2} + \gamma \right) \right), \tag{3.3}$$

where γ are the B-Splines nodes.

Consider now a general 2D filter whose support expands along the axis

$$\vec{k_x} = (\cos\theta_x, \sin\theta_x), \qquad \vec{k_y} = (\cos\theta_y, \sin\theta_y).$$

Remark 3.1.1. *The original 2D kernel can be seen as a particular case using the axis $\vec{k_x} = (1, 0)$ and $\vec{k_y} = (0, 1)$.*

Let $\mathcal{B}_1 := \{\mathbf{e}_1 = (1, 0), \mathbf{e}_2 = (0, 1)\}$, *i.e.*, the Cartesian reference system, and consider a second basis $\mathcal{B}_2 := \left\{\vec{k_x}, \vec{k_y}\right\}$.

Remark 3.1.2. *The set $\mathcal{B}_2 = \left\{\vec{k_x}, \vec{k_y}\right\}$ is a basis of $\mathbb{R}^2$ provided $\theta_x \neq \theta_y$.*

The following equations express the kernel axis in the original reference system $\mathcal{B}_1$:

$$\vec{k_x} = \cos\theta_x \cdot \mathbf{e}_1 + \sin\theta_x \cdot \mathbf{e}_2, \tag{3.4}$$

$$\vec{k_y} = \cos\theta_y \cdot \mathbf{e}_1 + \sin\theta_y \cdot \mathbf{e}_2. \tag{3.5}$$

The **change-of-basis matrices** for the rotated filter coordinate system are obtained from these relations:

$$P_{\mathcal{B}_2\leftarrow\mathcal{B}_1}\begin{pmatrix}\cos\theta_x & \cos\theta_y\\ \sin\theta_x & \sin\theta_y\end{pmatrix} \text{ and } P_{\mathcal{B}_1\leftarrow\mathcal{B}_2} = \frac{1}{det\left(P_{\mathcal{B}_2\leftarrow\mathcal{B}_1}\right)}\begin{pmatrix}\sin\theta_y & -\cos\theta_y\\ -\sin\theta_x & \cos\theta_x\end{pmatrix}. \tag{3.6}$$

This gives the following relations:

$$X = (x_1, x_2)_{\mathcal{B}_1} = P_{\mathcal{B}_1\leftarrow\mathcal{B}_2}\cdot X' \Leftrightarrow X' = (x_1', x_2')_{\mathcal{B}_2} = P_{\mathcal{B}_2\leftarrow\mathcal{B}_1}\cdot X.$$

Note 3.1.1. *For orthogonal rotations, let $\theta_x = \theta$. Since*

$$\theta_y = \theta + \pi/2 \Rightarrow \begin{cases}\cos\theta_y = -\sin\theta\\ \sin\theta_y = \cos\theta,\end{cases} \tag{3.7}$$

the change of basis matrices reduces to

$$P_{\mathcal{B}_2\leftarrow\mathcal{B}_1} = \begin{pmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{pmatrix} \quad \textit{and} \quad P_{\mathcal{B}_1\leftarrow\mathcal{B}_2} = \begin{pmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{pmatrix}. \tag{3.8}$$

The rotated filter consists of rewriting the convolution in the new basis:

$$u^\star(\overline{x},\overline{y}) = \frac{1}{H_xH_y}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} K_x\left(\frac{\overline{x}-x'}{H_x}\right)K_y\left(\frac{\overline{y}-y'}{H_y}\right)u(x',y')J(x',y')dx'dy' \tag{3.9}$$

where

$$\begin{pmatrix}x'\\ y'\end{pmatrix} = P_{\mathcal{B}_2\leftarrow\mathcal{B}_1}\begin{pmatrix}x\\ y\end{pmatrix}, \qquad \text{and } J(x',y') = det(P_{\mathcal{B}_2\leftarrow\mathcal{B}_1}). \tag{3.10}$$

Note 3.1.2. *For orthogonal systems, the Jacobian $J(x',y')$ reduces to*

$$det(P_{\mathcal{B}_2\leftarrow\mathcal{B}_1}) = \cos^2\theta + \sin^2\theta = 1.$$

Remark 3.1.3. *This definition is consistent with the original post-processor. This can be seen by taking $\theta = 0$. In this case, $x = x'$ ($y = y'$), giving equation* (3.2).

Figure 3.4 illustrates the structure of two tensor product filters. The original DG mesh is aligned with the Cartesian axis. The right picture shows the effect of the rotation in the kernel and each of the B-Spline supports.

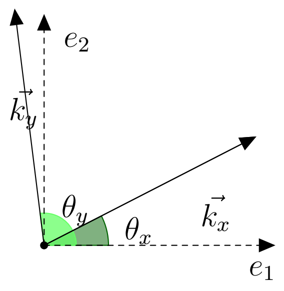


Figure 3.3: Illustration of the basis vectors for the two coordinate systems.

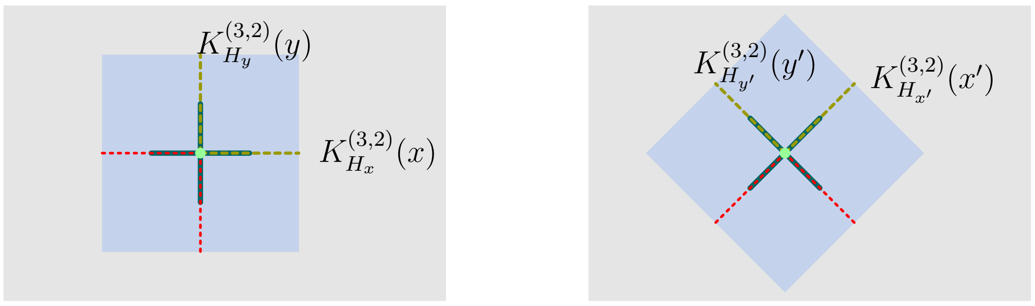


Figure 3.4: Structure of the 2D tensor product filters using a Cartesian axis aligned kernel (left) and a $\pi/4$-rotated kernel (right). The smaller squares (dark blue) highlight the total filter support. The segments (solid, dotted and dashed) across the filters support along the directions K_x, K_y represent each of the B-Spline support and the two dots (light blue) denote the evaluation point.

3.2 Global Analysis: Superconvergence & Errors

One important question to ask about these new filters is if they preserve superconvergence. Thus, a numerical study was carried out over the 2D linear advection equation:

$$\begin{cases} u_t + u_x + u_y = 0, \quad (x,y) \in [0,2\pi]^2,\ t \in [0,T] \\ u_0(x,y) = \sin(x+y). \end{cases} \tag{3.11}$$

The unfiltered solution was obtained with a DG scheme using an upwind flux over a uniform quadrilateral mesh. In order to quantify the effect of the rotation, the first study was done over the L^2 projection of the function

$$u_0(x,y) = \sin(x+y), \qquad (x,y) \in [0,2\pi]^2. \tag{3.12}$$

This provides a simplified setup for the filter since the effects of the numerical flux on the solution are omitted.

Three particular rotations: $\theta = 0,\ \pi/6,\ \pi/4$ were considered. Furthermore, three types of scalings were used corresponding to the formula $H = \mu h$, with h being the DG mesh size and varying μ. The value $\mu = 1$ corresponds to the scaling for which it is proven that the Cartesian axis aligned filter achieves superconvergence [12]. This scaling is chosen to ensure mesh translation invariance by shifts of size H as shown in Figure 3.5. Thus, for the $\pi/4$-rotation, the case $H = \sqrt{2}h$ was introduced. The $\pi/6$ rotation cannot produce a translation invariant space but it is possible to preserve invariance for each direction by using shifts of $\mu = 1/\cos(\pi/6)$ as shown in Figure 3.6. Finally, the value $\mu = 0.9$ was included in the study to support the discussion on superconvergence and minimum error.

Figure 3.7 shows the contour lines of the point-wise error along the entire field before and after applying a $\pi/4$-rotated filter. In Figure 3.8 the same error profile is shown but in this case for a $\pi/6$-rotated filter. These plots highlight how relevant is the scaling both in terms of smoothness recovery and error reduction; regarding smoothness, observe that for the lowest polynomial degree ($\mathbb{P}^1$), the solution is only smooth when the scaling is chosen according to the discussion given in the previous paragraph. This prevails still for the case $\mathbb{P}^2$ in Figure 3.7, corresponding to a $\pi/4$ rotated kernel and it is due to the large difference between the scalings. Concerning mesh translation invariance, this rotation requires a larger scaling than the $\pi/6$ rotation. Notice that the difference between the values $\mu = 0.9,\ 1$ and $\mu = \sqrt{2}$, corresponding to the $\pi/4$ rotation is much larger than the difference between $\mu = 1$ and $\mu = 1/(\cos(\pi/6) \approx 1.15$. On the other hand, in terms of error reduction, larger scalings seemed less effective. Notice how both filters achieved lower errors when applying a scaling equal to the mesh size. Nevertheless, regardless the support size, all the filtered solutions reduced the error from the original DG solution.

Table 3.1 shows the global L^2 and L^∞ errors from Figure 3.7. In terms of superconvergence, the $\mathbb{P}^1$ case shows that only the scaling $H = \sqrt{2}h$ allows the rotated filter to extract the expected $2k+2$ order; when post-processing the L^2 projection of a function, the filtered solution is of order $(2k+1)+1$. A similar behaviour can be observed

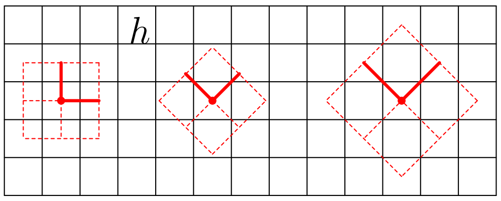


Figure 3.5: Kernel scalings using $H = h$ (left and centre) and $H = \sqrt{2}h$ (right). The central image is not translation invariant.

for the $\pi/6$-rotation for the scaling $\mu = 1/\cos(\pi/6)$ as shown in Table 3.2, although the difference here is less clear. The mesh resolution is not high enough to allow the scaling effects to be more visible. Table 3.3, which will be discussed later, also includes a 80×80 element mesh for the $\mathbb{P}^1$ approximation. For such case, it is clear how the $\pi/4$-rotated filter has a better global performance when applying the scaling $\mu = \sqrt{2}$. With the available computational resources, the studies on these filters were limited to relatively coarse meshes. The global errors shown in this table suggest that the scaling $H = h$ has a better performance. On the other hand, for a $\pi/4$-rotated filter, as the mesh is being refined (see Table 3.3), the value switches towards $H = \sqrt{2}h$. In [32], the filter was tested using different scalings for several mesh types. It was concluded that although asymptotically, the translation invariant scaling gives the best results, over coarser meshes, a smaller scaling will result in greater error reduction. On the other hand, in general, larger scalings tend to worsen the error. This is consistent with the results presented here.

Finally, Table 3.3 compares the L^2-errors of the $\pi/4$-rotated filter with the Cartesian axis aligned filter. Observe that, in general, using the scaling $\mu = 0.9$ outperforms (in terms of the error) the other choices for both filters, aligned and rotated. Figure 3.9 compares the three rotations using a scaling $H = h$. Notice that for the value $\theta = 0$, this scaling effectively reduces the oscillations. On the other hand, as mentioned earlier, such value is not suitable for the other rotations since oscillations in the filtered error still persist. Table 3.4 shows the global L^2 errors and orders corresponding to these plots. The values in this table show that the case $\theta = 0$ gives consistently the lowest errors. Therefore, although the rotated filters recover smoothness and improve the unfiltered error, overall, the axis aligned filter has a better performance. This filter not only produces a smooth solution too but it is the one that achieves greatest error reduction.

3.3 Local Behaviour

The previous experiments focused on the global behaviour of the filters in order to examine whether superconvergence was preserved when rotating the kernel. The following experiments were carried out in order to understand whether there is a best rotation according to the point location relative to the element to which it belongs.

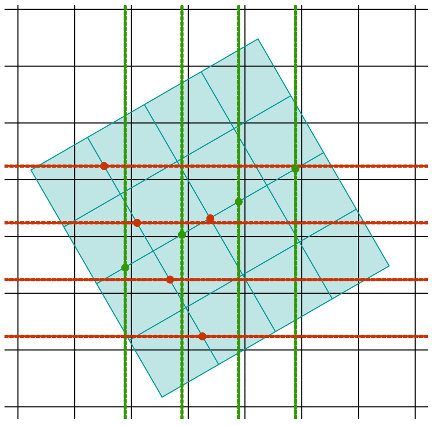


Figure 3.6: A $\pi/6$ rotated filter with a scaling $H = 1/\cos(\pi/6)$ showing the invariant space.

	Unfiltered				Scaling	Filtered			
					$H = \mu h$	$\theta = \pi/4$			
N	L^2-Error	Order	L^∞-Error	Order	μ	L^2-Error	Order	L^∞-Error	Order
					$\mathbb{P}^1$				
20	3.7e-03	-	1.3e-02	-	1 $\sqrt{2}$	3.5e-04 1.2e-03	- -	6.6e-04 1.7e-03	- -
40	9.2e-04	2.00	3.3e-03	1.99	1 $\sqrt{2}$	3.1e-05 7.8e-05	3.50 3.96	7.5e-05 1.1e-04	3.14 3.96
					$\mathbb{P}^2$				
20	9.8e-05	-	3.2e-04	-	1 $\sqrt{2}$	1.8e-05 1.3e-04	- -	2.5e-05 1.9e-04	- -
40	1.2e-05	3.00	4.1e-05	2.99	1 $\sqrt{2}$	2.8e-07 2.2e-06	5.95 5.92	4.3e-07 3.1e-06	5.87 5.92
					$\mathbb{P}^3$				
20	1.9e-06	-	4.9e-06	-	1 $\sqrt{2}$	1.1e-06 1.6e-05	- -	1.5e-06 2.3e-05	- -
40	1.2e-07	4.00	3.1e-07	3.99	1 $\sqrt{2}$	4.4e-09 6.9e-08	7.94 7.87	6.2e-09 9.7e-08	7.92 7.87

Table 3.1: L^2 and L^∞ results before and after applying a $\pi/4$-rotated filter on the L^2-projection of $u_0(x, y) = \sin(x + y)$ on $\Omega = [0, 2\pi]^2$ over an uniform mesh of size h. Results are shown using two different scalings, $H = h$ and $H = \sqrt{2}h$.

Post-Processing Cell Centres

Before performing a local analysis limiting the filter to a single element, the entire field was post-processed choosing a particular set of points: the element centres. This is a situation where the original post-processor (Cartesian axis aligned) is expected to be optimal. This is because the mesh is made of uniform quadrilaterals and the evaluating point is at the centre, using the scaling $H = h$ (h being the mesh size) provides the filter with a support that has a completely symmetric footprint.

Consider the L^2 projection of the initial condition of Problem (3.11), *ie*, the DG solution at initial time. Figure 3.10 shows the point-wise error profiles after applying

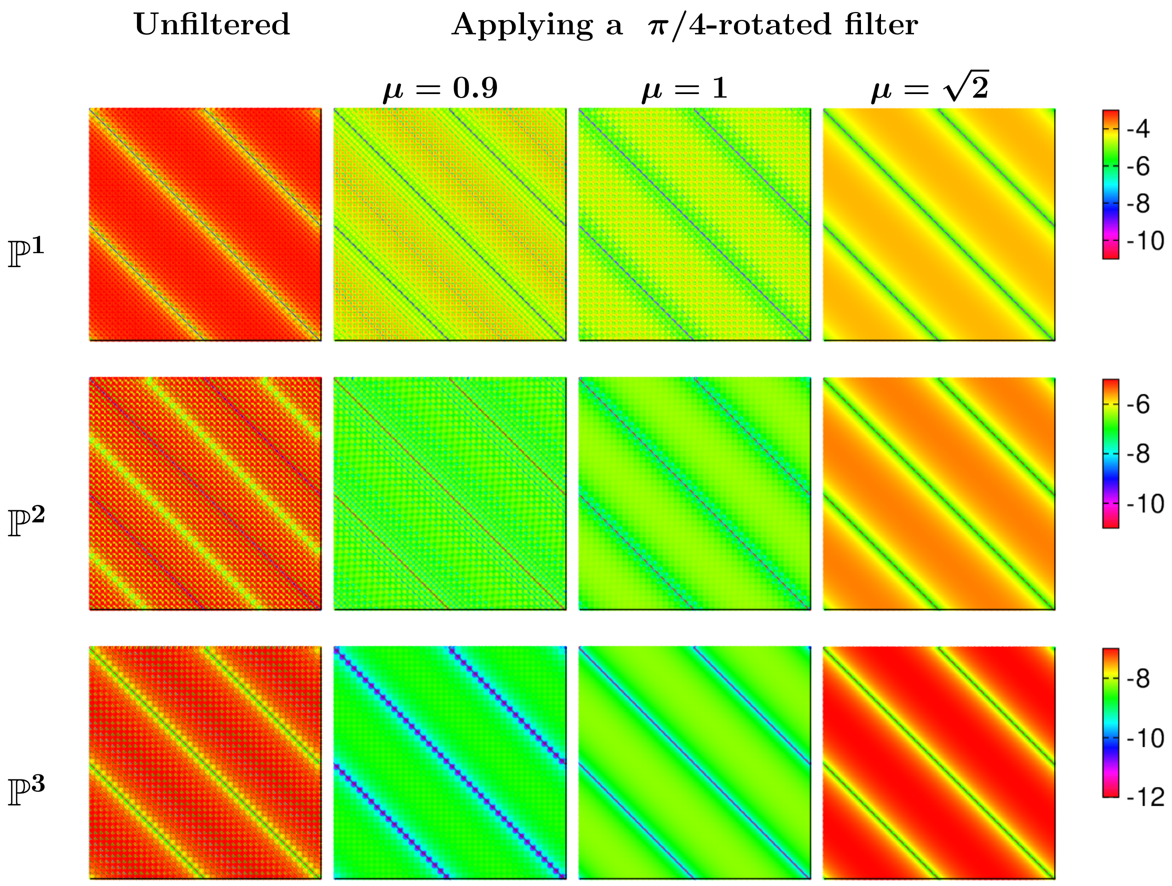


Figure 3.7: Contour-line error plots (log) before and after filtering the L^2-projection of $u_0(x,y) = \sin(x+y)$ in $\Omega = [0, 2\pi]^2$ using a $\pi/4$ tensor product filter with $H = \mu h$, h mesh size.

the rotated filter using ten rotations:

$$\theta = \frac{k}{10}\frac{\pi}{2}, \; k = 0, \dots, 9$$

and two scalings.

Note 3.3.1. *The filter support is square so it is not necessary to take $\theta \geq \pi/2$. In fact, pairs of complementary angles (see Figure 3.11) give exactly the same error and therefore in the right plots of Figure 3.10, only six error curves are shown. This can be attributed to the symmetry in the function* $\sin(x+y)$*.*

In this scenario, the zero rotation is consistently the best orientation. The magnitude of the error increases with the rotation angle, finding its maximum at $\theta = \pi/4$. Another important remark is that changing the scaling seems to have a similar impact on all rotations. Observe the $\mathbb{P}^2$ case of Figure 3.10. In this case, the filter has a smaller support since $H = 0.8h$ and the error curves show similar behaviour as that for $H = h$ case in terms of the relation between the magnitude of the error and the rotation angle.

Filtering Inside a Particular Element

In the previous study, the axis aligned filter showed optimal performance (in terms of error reduction) compared to the rotated filters. Overall, the behaviour of the

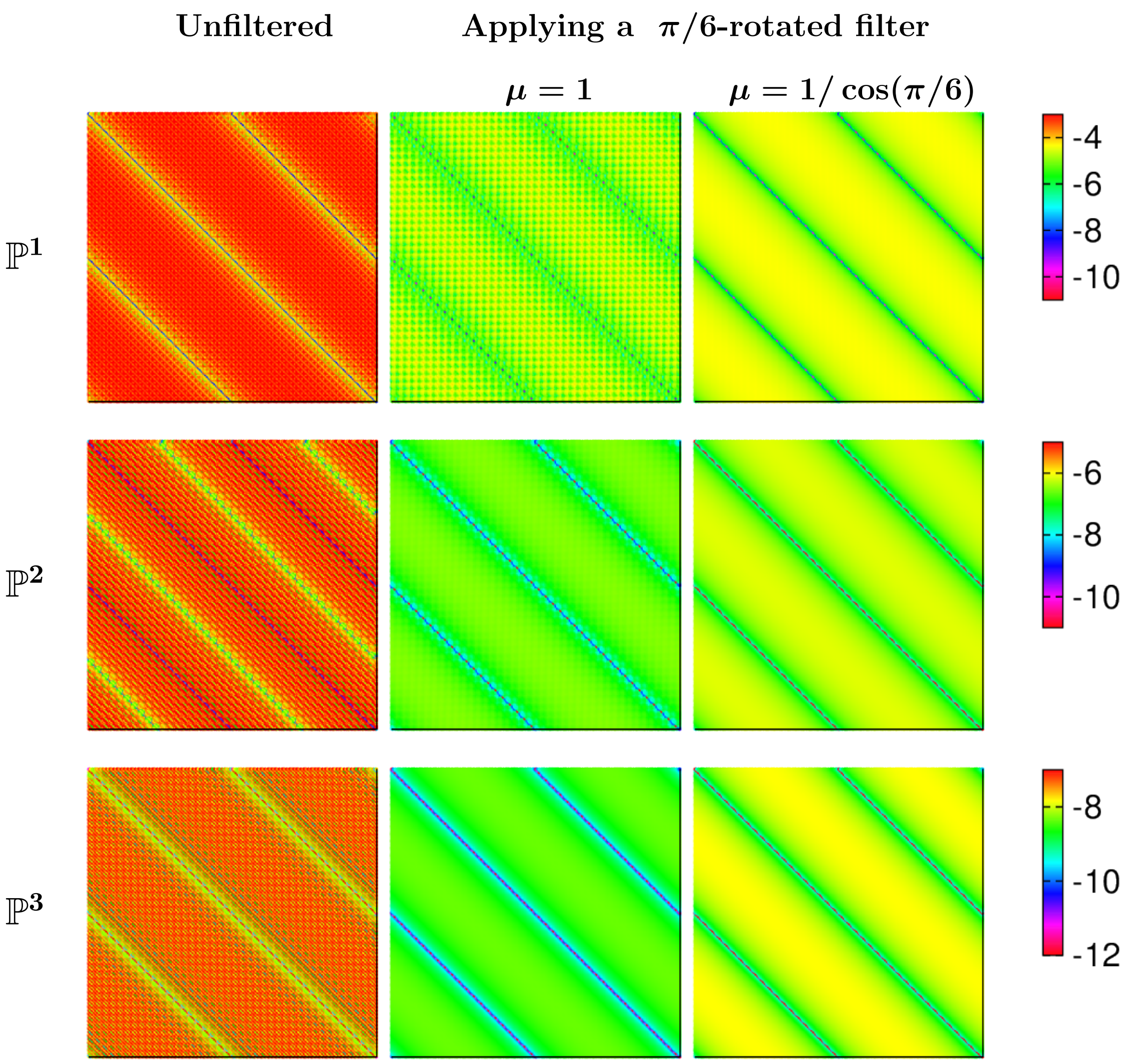


Figure 3.8: Contour-line error plots (log) before and after filtering the L^2-projection of $u_0(x, y) = \sin(x+y)$ in $\Omega = [0, 2\pi]^2$ using a $\pi/6$ tensor product filter with $H = \mu h$, h mesh size.

filter with different orientations relative to each other was consistent throughout the entire field. In the next experiment, the filters were tested on a particular element. The reason for not studying all the elements was because it would require very long computational times. In order to analyse the filter behaviour, one needs to implement several polynomial degrees and mesh refinements. For example, using five scalings and five rotations implies needing 25 simulations. For a 20×20 mesh using nine quadrature points per element, it requires post-processing $25 \times 20 \times 20 \times 9 = 9 \cdot 10^4$ points. Therefore, it was assumed that the filter behaviour at a particular element could be representative (to some extent) of its behaviour on all elements.

Problem (3.11) was studied in two different ways; first, the filters were applied to the L^2 projection of the initial condition. Then, the problem was solved for time $T = 12$ using a Runge-Kutta (RK4) DG scheme with upwind flux. This introduces the effect of the weak continuity at the element interface. Figure 3.12 shows the error profiles for three different polynomial degrees using rotated filters of varying angle $\theta \in [0,\ 2\pi)$ and

	Unfiltered				Scaling	Filtered			
					$H = \mu h$	$\theta = \pi/6$			
N	L^2-Error	Order	L^∞-Error	Order	μ	L^2-Error	Order	L^∞-Error	Order
$\mathbb{P}^1$									
20	3.7e-03	-	1.3e-02	-	1	3.0e-04	-	5.7e-04	-
					$1/\cos(\pi/6)$	5.0e-04	-	7.2e-04	-
40	9.2e-04	2.00	3.3e-03	1.99	1	2.3e-05	3.74	6.0e-05	3.24
					$1/\cos(\pi/6)$	3.2e-05	3.98	4.6e-05	3.98
$\mathbb{P}^2$									
20	9.8e-05	-	3.2e-04	-	1	1.4e-05	-	2.1e-05	-
					$1/\cos(\pi/6)$	3.3e-05	-	4.7e-05	-
40	1.2e-05	3.00	4.1e-05	2.99	1	2.3e-07	5.96	3.4e-07	5.90
					$1/\cos(\pi/6)$	5.4e-07	5.95	7.6e-07	5.95
$\mathbb{P}^3$									
20	1.9e-06	-	4.9e-06	-	1	8.1e-07	-	1.2e-06	-
					$1/\cos(\pi/6)$	2.5e-06	-	3.6e-06	-
40	1.2e-07	4.00	3.1e-07	3.99	1	3.3e-09	7.94	4.7e-09	7.93
					$1/\cos(\pi/6)$	1.0e-08	7.92	1.5e-08	7.92

Table 3.2: L^2 and L^∞ results before and after applying a $\pi/6$-rotated filter on the L^2-projection of $u_0(x, y) = \sin(x + y)$ on $\Omega = [0, 2\pi]^2$ over an uniform mesh of size h. Results are shown using two different scalings, $H = h$ and $H = \sqrt{2}h$.

scaling $0.8 \leq \mu \leq 1.4$. The left set of figures were plotted as a function of the scaling and the right figures as a function of the rotation angle. The sample corresponds to 25 Quadrature points inside the element with centre $C \approx (1.2, 1.75)$ and $u_h(x, y) \approx 0.1$. Notice that as the polynomial degree increases, the scaling that minimises the error tends to $\mu \approx 1$. Furthermore, the angle plots (right) show that for the $\mathbb{P}^3$ case, the zero rotation is almost consistently the one that minimises the error. In fact, these plots suggest the same as Figure 3.10: the error increases as the rotation angle increases. Figure 3.13 shows the element L^2 and L^∞ norms for both the L^2 projection and the DG solution ($T = 12$). Comparing the two types of solutions (L^2 projection vs *DG*), it seems that the differences between the performance of the filters are smaller once the solution is evolved in time. In the L^2 projection error plots, increasing the scaling aids the rotated filters in reducing the error. For example, the $\pi/4$ rotation has a lower error when applying larger scalings. However, for the DG solution, the error behaves similar to the cell centre experiment (see Figure 3.10). Changing the scaling affects all the rotations in a similar way and the error increases as the rotation angle increases. The plots suggest that the optimal scaling for error reduction corresponds to the value $\mu = 1$. In addition, for this value, the difference in the magnitude of the error between the rotations $\theta = 0,\ \pi/4$ is very small, except for the highest polynomial degree. However, the zero rotation consistently minimises the error.

	Unfiltered		Scaling	Filtered			
			$H = \mu h$	$\theta = 0$		$\theta = \pi/4$	
N	L^2-Error	Order	μ	L^2-Error	Order	L^2-Error	Order
				$\mathbb{P}^1$			
20	3.7e-03	-	0.9	1.5e-04	-	3.5e-04	-
			1	1.9e-04	-	3.5e-04	-
			$\sqrt{2}$	6.7e-04	-	1.2e-03	-
40	9.2e-04	2.00	0.9	1.5e-05	3.28	6.4e-05	2.44
			1	1.2e-05	3.99	3.1e-05	3.50
			$\sqrt{2}$	6.7e-05	3.31	7.8e-05	3.96
80	2.3e-04	2.00	0.9	3.6e-06	2.37	1.6e-05	2.05
			1	7.3e-07	4.00	5.6-06	2.47
			$\sqrt{2}$	1.4e-05	2.28	4.9e-06	3.99
				$\mathbb{P}^2$			
20	9.8e-05	-	0.9	9.5e-06	-	2.4e-06	-
			1	1.8e-05	-	4.5e-06	-
			$\sqrt{2}$	1.3e-04	-	3.5e-05	-
40	1.2e-05	3.00	0.9	4.0e-08	5.90	2.4e-07	5.32
			1	7.1e-08	5.98	2.8e-07	5.95
			$\sqrt{2}$	5.8e-07	5.92	2.2e-06	5.92
				$\mathbb{P}^3$			
20	1.9e-06	-	0.9	6.0e-08	-	4.7e-07	-
			1	1.4e-07	-	1.1e-06	-
			$\sqrt{2}$	2.1e-06	-	1.6e-05	-
40	1.2e-07	4.00	0.9	2.4e-10	7.97	1.9e-09	7.93
			1	5.5e-10	7.97	4.4e-09	7.94
			$\sqrt{2}$	8.7e-09	7.94	6.9e-08	7.87

Table 3.3: L^2 errors and orders comparing the $\pi/4$-rotated filter and the Cartesian axis filter for the L^2-projection of $u_0(x, y) = \sin(x + y)$ on $\Omega = [0, 2\pi]^2$ over an uniform mesh of size h.

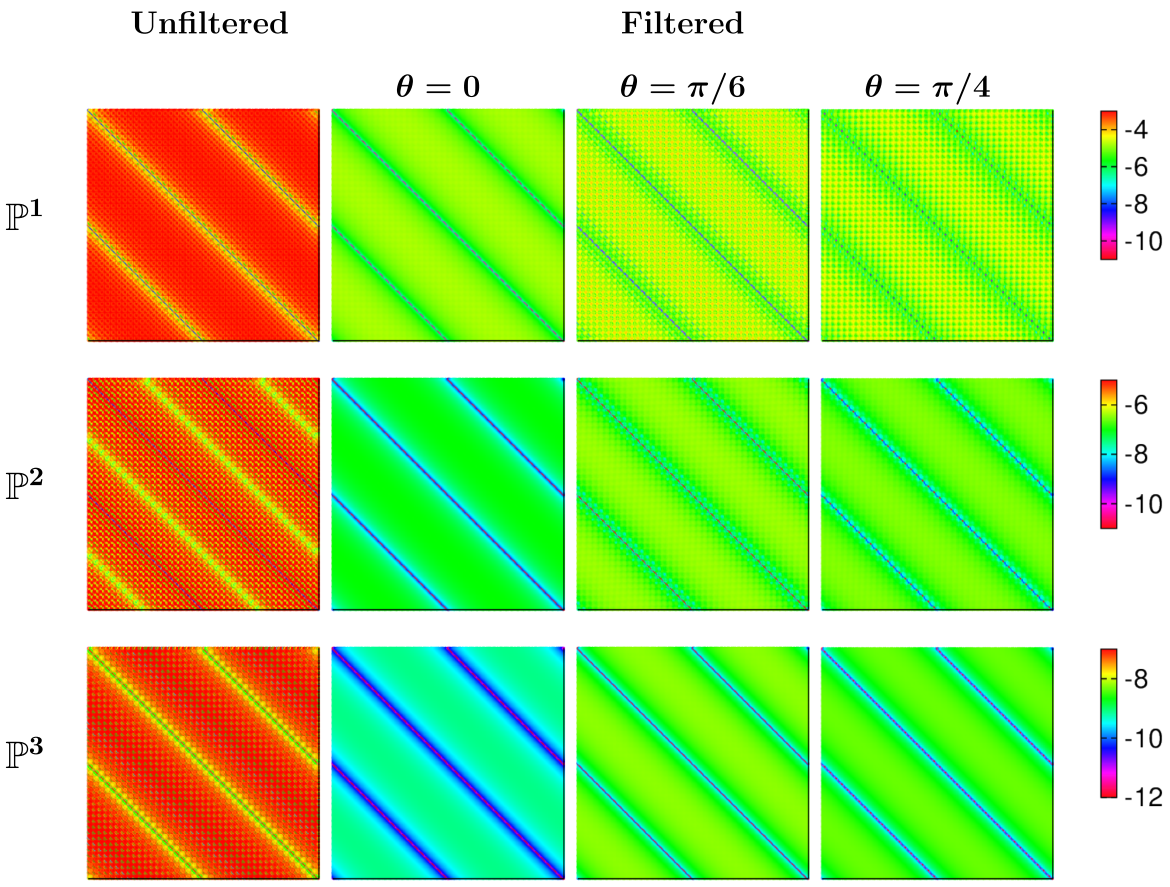


Figure 3.9: Contour-line error plots (log) before and after filtering the L^2-projection of $u_0(x, y) = \sin(x + y)$ in $\Omega = [0, 2\pi]^2$ for different SIAC filters using $H = h$, h mesh size.

	Unfiltered		Filtered					
			$\theta = 0$		$\theta = \pi/6$		$\theta = \pi/4$	
N	L^2-Error	Order	L^2-Error	Order	L^2-Error	Order	L^2-Error	Order
$\mathbb{P}^1$								
20	3.7e-03	-	1.9e-04	-	3.0e-04	-	3.5e-04	-
40	9.2e-04	2.00	1.2e-05	3.99	2.3e-05	3.74	3.1e-05	3.50
$\mathbb{P}^2$								
20	9.8e-05	-	4.5e-06	-	1.4e-05	-	1.8e-05	-
40	1.2e-05	3.00	7.1e-08	5.98	2.3e-07	5.96	2.8e-07	5.95
$\mathbb{P}^3$								
20	1.9e-06	-	1.4e-07	-	8.1e-07	-	1.1e-06	-
40	1.2e-07	4.00	5.5e-10	7.97	3.3e-09	7.94	4.4e-09	7.94

Table 3.4: L^2 errors and orders before and after post-processing the L^2-projection of $u_0(x, y) = \sin(x + y)$ on $\Omega = [0, 2\pi]^2$ applying filters with scaling $H = h$, h mesh size.

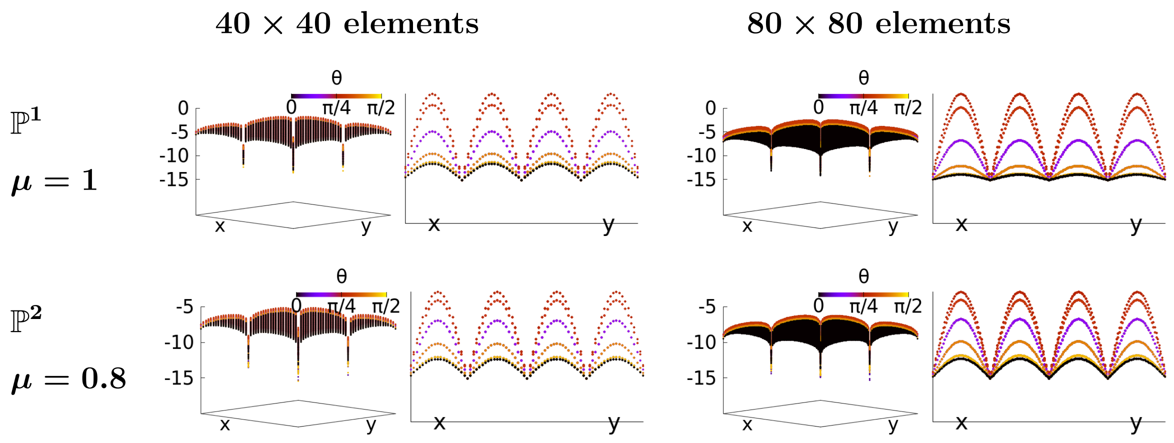


Figure 3.10: Surface error plots (log) and 1D slices of the error after filtering the L^2 projection of $u(x, y) = \sin(x + y)$ for the element centre along the entire field $\Omega = [0, 2\pi]^2$. The scalings correspond to $H = \mu h$ and ten rotations were considered.

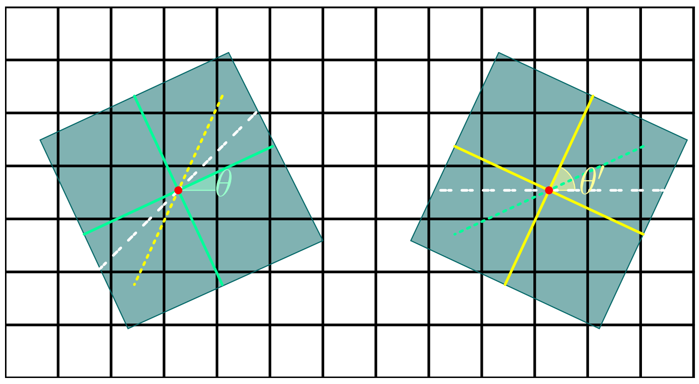


Figure 3.11: Footprints of two kernels with same error profiles corresponding to a $\theta-$rotated filter and the supplementary $\theta' = \pi - \theta$. The rotation can also be seen as $\theta' = \pi/2 - \theta$ since the kernels have square support.

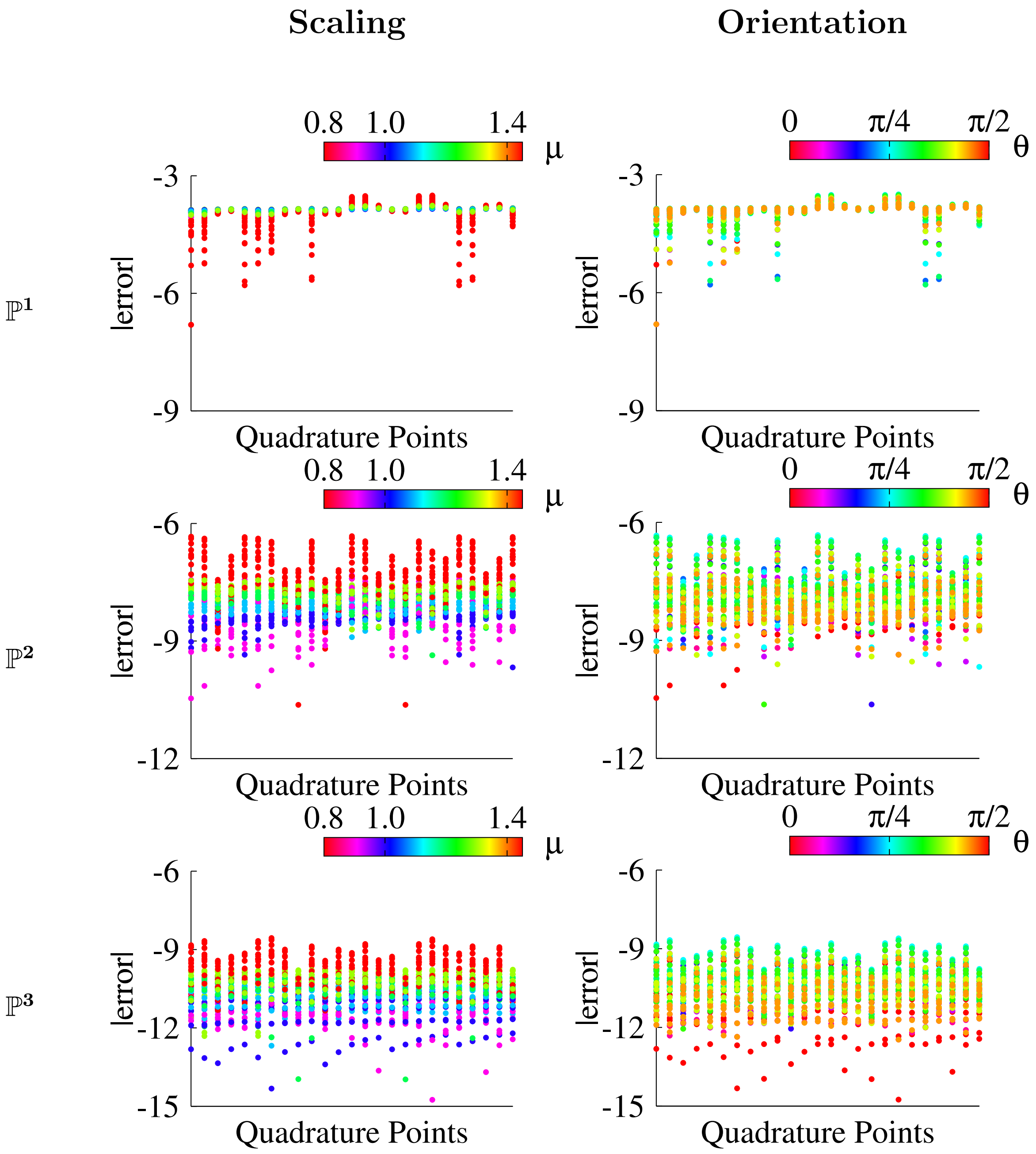


Figure 3.12: Pointwise error profiles as a function of the scaling (left) and the rotation angle (right) after postprocessing the L^2 projection of $u(x, y) = \sin(x + y)$ inside a particular element from a 80×80 mesh.

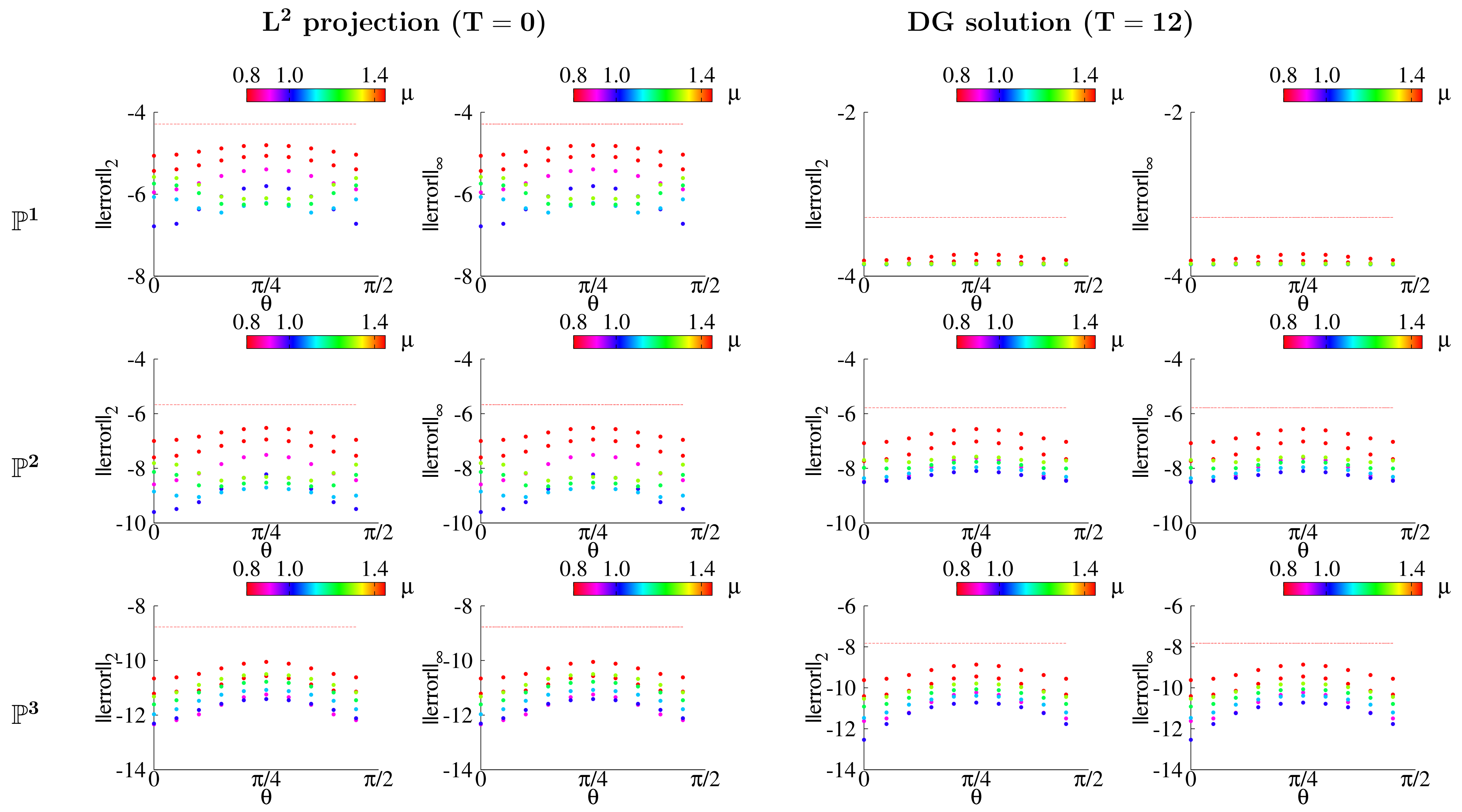


Figure 3.13: Local L^2 and L^∞ errors for a particular element as a function of the rotation angle and scaling after postprocessing the solution to Problem (3.11) using 80×80 elements. The horizontal dashed lines denote the unfiltered errors.

3.4 Discussion

The previous numerical results suggest that the rotated filters preserve the properties of SIAC filtering in terms of superconvergence and smoothness recovery. On the other hand, it was observed that keeping the filter aligned with the Cartesian axis resulted in greater error reduction. Maximum error reduction is the most desirable property when applying the filters for engineering problems. Hence, from these experiments, it was concluded that rotated tensor product filters are not a good alternative over the original axis aligned filter.

This chapter served as an introduction to tensor product rotated filtering. As a first approach, the problem model considered in the experiments consisted of a simple linear hyperbolic equation:

$$u_t + u_x + u_y = 0.$$

In the future, studies on these filters should extend to non-linear problems. Furthermore, the numerical examples done for these filters were done over a uniform quadrilateral mesh. Therefore, the question is whether the same behaviour holds for general meshes. For non-uniform meshes, it is possible that there is a less clear "best scaling-orientation" for maximum error reduction. In [36], optimal accuracy is discussed for non-uniform meshes. The authors show the difficulties arising from matching the theoretical optimal scaling to the one observed from running numerical tests. Since for non-uniform meshes it is not even clear for the Cartesian aligned filter which is the best scaling, the study for the rotated filters was limited to uniform quadrilateral meshes. The applications to non-uniform meshes are left for future work. The important information obtained from rotating the filter was that post-processing in different directions still allows for increasing the convergence order and reducing the error of the filtered solution. This motivated the development of the SIAC Line filters presented in the next chapter.

Finally, it is left as an open problem what would happen if the filter shape changed. Notice that although a rotation was introduced, throughout the experiments, the filter preserved a orthogonal inner axis. In Section 3.1, the formulation of the rotated filtering convolution was done for a general basis, not necessarily requiring orthogonal vectors. Figure 3.14 illustrates the idea of building a 2D $\pi/4$-rotated filter with non-orthogonal inner axis. This filter also has the mesh translation invariance property and uses a smaller support compared to the rotated filter applied in Section 3.2 using $\{\theta = \pi/4, \mu = \sqrt{2}\}$.

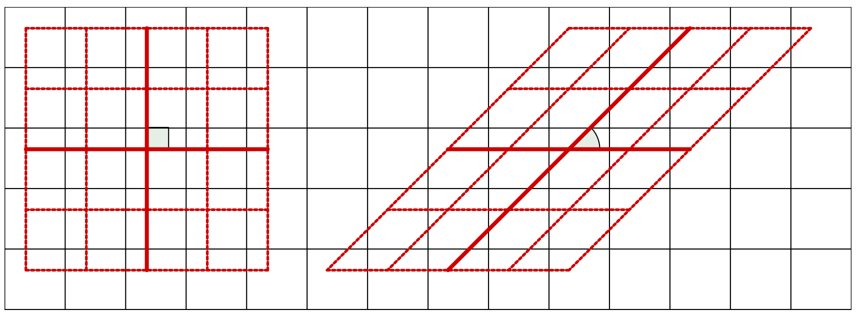


Figure 3.14: Footprint of the mesh of two kernels with translation invariance property showing an alternative rotation with non orthogonal inner axis (right).

Chapter 4

SIAC Line Filters

There are different challenges that SIAC filters should overcome in order to become a practical tool during flow visualisation. From an engineering perspective, the filter should be robust, require relatively low computational intensity and have short simulation times. One important limiting factor on the applications of tensor product SIAC filters is the long computational times for higher degree filters. In addition, as discussed in Chapter 3, applying the filter to general meshes not only adds computational complexity to the algorithm design but also intensifies the process of finding and sorting integrable regions.

The numerical experiments for the tensor product rotated filters suggested that introducing a rotation does not destroy the properties that characterise SIAC filters; provided the appropriate scaling is used, post-processing with these filters produces a smooth solution with same order of accuracy than when keeping the kernel aligned with the Cartesian axis. The question is then whether it is necessary to have a tensor product structure. The idea of lower dimension SIAC filters for multidimensional domains was first introduced in [64]. They showed the potential of this technique with an empirical study on streamlines, implementing a one-sided filter along the curve using arc-length parametrization. Here, that idea is developed mathematically leading to **SIAC Line filters**: rotated SIAC filters with support expanding only along a segment inside the 2D domain. This family of filters transforms the 2D integral of the convolution into a line integral. Therefore, from a computational point of view, the advantages are immediate. Not only the support size is enormously reduced and hence the computational times but also sorting the integrable regions becomes a much less intense and simple task. Furthermore, it is possible to prove superconvergence for these filters and provide similar error estimates than for the original tensor product filter. This chapter begins by reviewing the theory for proving superconvergence. The SIAC Line filters are then formally introduced together with Theorem 4.3.1 which provides theoretical error estimates for these filters for linear hyperbolic problems. Then, several numerical experiments are presented to study the performance of these filters.

4.1 Proving Superconvergence for SIAC Filters

In [13], it was proven that the DG approximation has $2k+1$ convergence in the negative-order norm for the approximation and the divided differences. SIAC filters exploit this fact and can achieve $2k+1$ order in the L^2 norm for the actual solution. In order to illustrate the important components for proving the same properties for the rotated filter, the proof of Theorem 2.2.1 is discussed.

Theorem 4.1.1. *(Cockburn, Luskin, Shu, and Süli [13].) Under the same conditions in Theorem 2.1.2 and if* $\Omega_0 + 2supp\left(K_h^{(2k+1,k+1)}\right) \subset\subset \Omega_1 \subset \Omega$, *then for* $H = h$ *(h being the DG mesh size):*

$$\left\|u - K_h^{(2k+1,k+1)} \star u_h\right\|_{0,\Omega_0} \leq Ch^{2k+1}. \tag{4.1}$$

Proof. The full proof of this Theorem can be found in [13] and here only a sketch is given.

Begin by splitting the error:

$$\left\|u - K_h^{(2k+1,k+1)} \star u_h\right\|_{0,\Omega_0} \leq \underbrace{\left\|u - K_h^{(2k+1,k+1)} \star u\right\|_{0,\Omega_0}}_{\Theta_1} + \underbrace{\left\|K_h^{(2k+1,k+1)} \star (u - u_h)\right\|_{0,\Omega_0}}_{\Theta_2}.$$

The term Θ_1 is bounded using the polynomial reproduction property. Let $T^{2k}u(y,\cdot)$ be the Taylor expansion of degree $2k$ of u around y and denote by $\mathcal{R}^{2k+1} = u(\cdot) - T^{2k}u(y,\cdot)$ the residual. Then

$$u(x) - K_h^{(2k+1,k+1)} \star u(x) = \mathcal{R}^{2k+1}u(y,x) - \int_{\text{supp } K_h} K_h^{(2k+1,k+1)}(y-x)\mathcal{R}^{2k+1}u(y,x)dx.$$

Let $z = \frac{y-x}{h}$, then

$$u(x) - K_h^{(2k+1,k+1)} \star u(x) = \mathcal{R}^{2k+1}u(y,x) - \int_{\text{supp } K} K^{(2k+1,k+1)}(z)\mathcal{R}^{2k+1}u(y,x-hz)dz$$

and if $x = y$:

$$u(x) - K_h^{(2k+1,k+1)} \star u(x) = -\int_{\text{supp } K} K^{(2k+1,k+1)}(z)\mathcal{R}^{2k+1}u(x,x-hz)dz.$$

Hence,

$$\Theta_1 \leq \left\|K^{(2k+1,k+1)}\right\|_{L^1(\mathbb{R}^d)} \cdot \sup_{z\in\text{supp } K} \left\|R^{2k+1}u(\cdot,\cdot - hz)\right\|_{0,\Omega_0} \tag{4.2}$$

$$\leq \left\|K^{(2k+1,k+1)}\right\|_{L^1(\text{supp}K)} \cdot \frac{h^{2k+1}}{(2k+1)!}|u|_{2k+1,\Omega_0+h\cdot\text{supp}K} \tag{4.3}$$

Since $\|\psi^{(m)}(\cdot)\|_{L^1(\mathbb{R}^d)} = 1$, the kernel term in the equation above is bounded by its coefficients:

$$\left\|K^{(2k+1,k+1)}\right\|_{L^1(\text{supp}K} \leq \sum_{\gamma=0}^{2k} |c_\gamma|\psi^{(k+1)}\left(\cdot - (\frac{r}{2} - \gamma)\right) = \sum_{\gamma=0}^{2k} |c_\gamma| = C_0.$$

This allows to give the following estimate:

$$\left\| u - K_h^{(2k+1,k+1)} \star u \right\|_{0,\Omega_0} \leq \frac{h^{2k+1}}{(2k+1)!} C_0 |u|_{2k+1,\Omega_0 + h\cdot \mathrm{supp}K} \leq C_1 h^{2k+1}. \tag{4.4}$$

Apply now Lemma 2.1.1 to the second term:

$$\Theta_2 \leq C_1 \sum_{|\alpha|\leq k+1} \left\| D^\alpha \left(K_h^{(2k+1,k+1)} \star (u - u_h) \right) \right\|_{-(k+1),\Omega_1}. \tag{4.5}$$

Using the central B-Splines derivative property:

$$D^\alpha \psi_h^{(k+1)} = \partial_h^\alpha \psi^{(k+1-\alpha)}, \tag{4.6}$$

and since the convolution is a linear operator, *i.e.*,

$$D^\alpha \psi_h^{(k+1)} \star u = \psi^{(k+1-\alpha)} \star \partial_h^\alpha u,$$

the following equation holds:

$$\begin{aligned} \left(D^\alpha K_h^{(2k+1,k+1)} \star (u - u_h) \right) &= \left(D^\alpha K_h^{(2k+1,k+1)} \right) \star (u - u_h) \\ &= K_h^{(2k+1,k+1;\alpha)} \star \partial_h^\alpha (u - u_h). \end{aligned}$$

Imposing this in equation (4.5) gives

$$\begin{aligned} \Theta_2 &\leq C_1 \sum_{|\alpha|\leq k+1} \left\| D^\alpha K_h^{(2k+1,k+1)} \star (u - u_h) \right\|_{-(k+1),\Omega_1} && (4.7) \\ &\leq C_1 \sum_{|\alpha|\leq k+1} \left\| K_h^{(2k+1,k+1)} \star \partial_h^\alpha (u - u_h) \right\|_{-(k+1),\Omega_1} && (4.8) \\ &\leq C_1 \sum_{|\alpha|\leq k+1} \left\| K_h^{(2k+1,k+1)} \right\|_{L^1(\mathbb{R})} \left\| \partial_h^\alpha (u - u_h) \right\|_{-(k+1),\Omega_1} && (4.9) \\ &\leq C_1 C_2 \sum_{|\alpha|\leq k+1} \left\| \partial_h^\alpha (u - u_h) \right\|_{-(k+1),\Omega_1}. && (4.10) \end{aligned}$$

Finally, using Theorem 2.1.2, one can conclude that:

$$\left\| K_h^{(2k+1,k+1)} \star (u - u_h) \right\|_{0,\Omega_0} \leq C h^{2k+1}. \tag{4.11}$$

□

Remark 4.1.1. *The polynomial reproduction property implies that convolving the exact solution with the filter produces an error of order $\mathcal{O}(h^{2k+1})$, with $2k$ being maximum polynomial degree of reproduction. This is controlled by the number of B-Splines used during kernel construction.*

Remark 4.1.2. *The divided differences play a key role for bounding the error component corresponding to the filtered DG approximation. The $2k+1$ accuracy is achieved by virtue of Theorem 2.1.2 using the B-Spline derivative property given in equation (4.6).*

4.2 SIAC Line Kernels

The proof of Theorem 4.1.1 highlights the important role played by the divided differences for proving superconvergence of the filtered solution; the proof relies on the ability of the kernel to transfer the derivatives to the DG approximation as divided differences and then apply Theorem 2.1.2. With an axis aligned kernel, a tensor product construction,

$$K_H^{(2k+1,k+1)}(x,y) = K_{H_x}^{(2k+1,k+1)}(x) \otimes K_{H_y}^{(2k+1,k+1)}(y),$$

is necessary in order to compute the multi-dimensional derivatives:

$$D^\alpha K_H^{(2k+1,k+1)}(x,y) = \frac{d^{\alpha_1}}{dx} K_{H_x}^{(2k+1,k+1)}(x) \frac{d^{\alpha_2}}{dy} K_{H_y}^{(2k+1,k+1)}(y), \quad \alpha_1 + \alpha_2 = \alpha.$$

On the other hand, recall that the rotated kernel is defined by:

$$K_H^{(2k+1,k+1)}(x',y') = K_{H_x}^{(2k+1,k+1)}(x') \otimes K_{H_y}^{(2k+1,k+1)}(y'), \quad (x',y') = \mathcal{P}_{\mathcal{B}_2 \leftarrow \mathcal{B}_1} \begin{pmatrix} x \\ y \end{pmatrix}.$$

This affords a great advantage; a single kernel direction allows for differentiation in terms of the original basis under all variables since:

$$K_{H_x}^{(2k+1,k+1)}(x') = K_{H_x}^{(2k+1,k+1)}(x,y). \tag{4.12}$$

Exploiting this fact, it is possible to avoid tensor products and reduce the filter dimension, transforming the convolution into a line integral whilst preserving the 2D SIAC properties. The only thing that needs to be proven is that the kernel derivatives can still be expressed as a combination of divided differences in the x and y directions. Then, Theorem 2.1.2 can be applied to obtain the desired $2k+1$ order of accuracy.

4.2.1 Univariate B-Splines Along Lines in $\mathbb{R}^2$

Let $\Gamma \subset \mathbb{R}^2$ be the line parametrized by the arc length

$$\Gamma(t) = t(\cos\theta, \sin\theta) \qquad t \in \mathbb{R},\ \theta \text{ fixed.} \tag{4.13}$$

This line can be identified with the kernel axis k_x. Notice that

$$\left.\begin{array}{ll} x = t\cos\theta & \Rightarrow x\cos\theta = t\cos^2\theta \\ y = t\sin\theta & \Rightarrow y\sin\theta = t\sin^2\theta \end{array}\right\} x\cos\theta + y\sin\theta = t. \tag{4.14}$$

On the other hand, using the rotation matrix (eq. (3.8)) gives:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{4.15}$$

Hence, $x' = x\cos\theta + y\sin\theta = t$.

Define the inverse curve

$$\begin{tikzcd} t \arrow[r, "\Gamma"] & (x,y) \arrow[l, bend left, "\Gamma^{-1}"] \end{tikzcd}$$

by:

$$\Gamma^{-1}(x,y) = x\cos\theta + y\sin\theta. \tag{4.16}$$

Remark 4.2.1.

$$t = t(x,y) = \Gamma^{-1}\left(\Gamma(t)\right) = \Gamma^{-1}(x,y) = x\cos\theta + y\sin\theta \quad \forall (x,y) \in \Gamma(t). \tag{4.17}$$

Definition 4.2.1. *(SIAC Line kernels). Consider the line* Γ *and its inverse* Γ^{-1} *given by equations* (4.13) *and* (4.16)*. Then, the B-Spline along the* Γ *line is defined by*

$$\tilde{\psi}_\theta^{(k+1)}(x,y) = \begin{cases} \psi^{(k+1)}\left(\Gamma^{-1}(x,y)\right) & \text{if } (x,y) \in \Gamma(t) \\ 0 & \text{otherwise,} \end{cases} \tag{4.18}$$

and has compact support

$$\text{supp } \tilde{\psi}_\theta^{(k+1)} = (t\cos\theta, t\sin\theta), \quad t \in \left[-\frac{k+1}{2}, \frac{k+1}{2}\right]. \tag{4.19}$$

$$\begin{tikzcd} \Gamma(t) \subset \mathbb{R}^2 \arrow[r] & \mathbb{R} \arrow[r] & \mathbb{R} \\ (t\cos\theta, t\sin\theta) \arrow[r, "\Gamma^{-1}"] \arrow[rr, bend right, "\tilde{\psi}_\theta^{(k+1)}"'] & t \arrow[r, "\psi^{(k+1)}"] & \psi^{(k+1)}(t) \end{tikzcd}$$

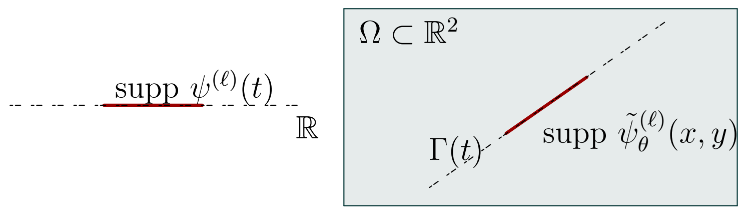

supp $\psi^{(\ell)}(t)$
$\mathbb{R}$
$\Omega \subset \mathbb{R}^2$
$\Gamma(t)$
supp $\tilde{\psi}_\theta^{(\ell)}(x,y)$

Figure 4.1: Illustration of an univariate B-Spline support along a line in $\mathbb{R}^2$.

Here, $\psi^{(k+1)}(\cdot)$ *denotes the univariate B-Spline of order* $k \geq 0$.
The ***SIAC Line kernel*** *is construced as a linear combination of these (scaled) B-Splines and the symmetric version has the following formula:*

$$K_{H,\Gamma}^{(2k+1,k+1)}(t) = \sum_{\gamma=-k}^{k} c_\gamma \psi_{\theta,H}^{(k+1)}(t-\gamma) \tag{4.20}$$

in arc length coordinates, or alternatively by:

$$K_{H,\Gamma}^{(2k+1,k+1)}(x,y) = \sum_{\gamma=-k}^{k} c_\gamma \tilde{\psi}_{\theta,H}^{(k+1)}\left(\Gamma^{-1}\left(x-\gamma\cos\theta, y-\gamma\sin\theta\right)\right) \tag{4.21}$$

in the Cartesian system.

The 2D convolution for the SIAC Line filter is given by:

$$u^\star(\overline{x},\overline{y}) = \frac{1}{H}\int_\Gamma K_{\Gamma,H}\left(\frac{t}{H}\right)u_h(\Gamma(t))dt, \tag{4.22}$$

where it was used that $\Gamma(t) = t(\cos\theta,\sin\theta) + (\overline{x},\overline{y})$ and $||\Gamma'(t)|| = 1$.

4.2.2 Differentiation and Divided Differences

Now that B-Splines for line filters have been introduced, it is necessary to characterise the derivatives of the term in equation (4.5) in Theorem 4.1.1,

$$\left\| D^\alpha K_h^{(2k+1,k+1)} \star (u-u_h)\right\|_{-(k+1),\Omega_1}. \tag{4.23}$$

Let $\ell = k+1$ and consider the B-Spline from Definition 4.2.1 (equation (4.18)):

$$\tilde{\psi}^{(\ell)}_{\theta,H}(x,y) = \psi^{(\ell)}_H\left(\Gamma^{-1}(x,y)\right).$$

Note 4.2.1. *Using* (4.17)*, we also have*

$$\tilde{\psi}^{(\ell)}_{\theta,H}(x,y) = \psi^{(\ell)}_H(t), \quad \textit{where } t = t(x,y).$$

Furthremore,

$$\frac{\partial t}{\partial x} = \cos\theta \quad \textit{and} \quad \frac{\partial t}{\partial x} = \sin\theta.$$

Then,

$$D^\alpha\tilde{\psi}^{(\ell)}_{\theta,H}(x,y) = \frac{\partial^{\alpha_1}}{\partial x^{\alpha_1}}\left(\frac{\partial^{\alpha_2}}{\partial y^{\alpha_2}}\left(\psi^{(\ell)}_H\left(\Gamma^{-1}(x,y)\right)\right)\right) = \frac{\partial^{\alpha_1}}{\partial x^{\alpha_1}}\left(\sin^{\alpha_2}\theta\cdot\frac{d^{\alpha_2}\psi^{(\ell)}_H(t)}{dt^{\alpha_2}}\right) \tag{4.24}$$

$$= \sin^{\alpha_2}\theta\frac{\partial^{\alpha_1}}{\partial x^{\alpha_1}}\left(\frac{d^{\alpha_2}\psi^{(\ell)}_H(t)}{dt^{\alpha_2}}\right) = \sin^{\alpha_2}\theta\cos^{\alpha_1}\theta\left(\frac{d^\alpha\psi^{(\ell)}(t)}{dt^\alpha}\right) \tag{4.25}$$

$$= \sin^{\alpha_2}\theta\cos^{\alpha_1}\theta\left(\partial^\alpha_H\psi^{(\ell-\alpha)}_H(t)\right), \qquad \alpha_1+\alpha_2 = \alpha. \tag{4.26}$$

This formula establishes a relation between the derivatives of the B-Spline along the line and the divided differences of the B-spline along the arc length parameter. However, in order to apply Lemma 2.1.1 in equation (4.23) and bound the filtered solution, a particular type of divided differences are introduced.

Definition 4.2.2. *(Directional Divided Difference). Consider the direction given by the vector $\vec{u} = (u_x, u_y)$. Then the scaled directional divided difference with respect to $\vec{u}$ is defined by*

$$\partial_{\vec{u},H}f(x,y) = \frac{1}{H}\left(f\left(x+\frac{H}{2}u_x, y+\frac{H}{2}u_y\right) - f\left(x-\frac{H}{2}u_x, y-\frac{H}{2}u_y\right)\right), \tag{4.27}$$

and the α-directional divided difference is defined by

$$\partial^\alpha_{\vec{u},H}f(x,y) = \partial_{\vec{u},H}\left(\partial^{\alpha-1}_{\vec{u},H}f(x,y)\right), \quad \alpha > 1. \tag{4.28}$$

The following Lemma provides a relation between the directional divided differences and the basis vectors.

Lemma 4.2.1. *Let f be a smooth function. Then, its (scaled) α-directional divided difference along the direction vector $u_{\theta,H} = (\cos\theta, \sin\theta)$ can be expressed as a sum of α-directional divided differences using the basis vectors:*

$$u_x^\theta = (\cos\theta, 0), \quad u_y^\theta = (0, \sin\theta)$$

through the formula

$$\partial^\alpha_{u_{\theta,H}} f(x,y) = \sum_{m=0}^{\alpha} \begin{pmatrix} \alpha \\ m \end{pmatrix} \partial^{\alpha-m}_{u^\theta_x,H} \partial^m_{u^\theta_y,H} f\left(x - \frac{m}{2}H\cos\theta, y + \frac{\alpha-m}{2}H\sin\theta\right). \tag{4.29}$$

Proof. Consider the first order divided difference. By definition:

$$\partial_{u_{\theta,H}} f(x,y) = \frac{1}{H} f\left(x + \frac{H}{2}\cos\theta, y + \frac{H}{2}\sin\theta\right) - \frac{1}{H} f\left(x - \frac{H}{2}\cos\theta, y - \frac{H}{2}\sin\theta\right).$$

Adding and subtracting the term

$$\frac{1}{H}\left(f\left(x - \frac{H}{2}\cos\theta, y + \frac{H}{2}\sin\theta\right)\right),$$

in the previous equation gives:

$$\begin{aligned}\partial_{u_{\theta,H}} f(x,y) =& \frac{1}{H} f\left(x + \frac{H}{2}\cos\theta, y + \frac{H}{2}\sin\theta\right) - \frac{1}{H} f\left(x - \frac{H}{2}\cos\theta, y + \frac{H}{2}\sin\theta\right) \\ &+ \frac{1}{H} f\left(x - \frac{H}{2}\cos\theta, y + \frac{H}{2}\sin\theta\right) - \frac{1}{H} f\left(x - \frac{H}{2}\cos\theta, y - \frac{H}{2}\sin\theta\right).\end{aligned}$$

Since

$$u_x^\theta = (\cos\theta, 0), \quad u_y^\theta = (0, \sin\theta),$$

the equation can be written as:

$$\partial_{u_{\theta,H}} f(x,y) = \partial_{u^\theta_x,H} f\left(x, y + \frac{H}{2}\sin\theta\right) + \partial_{u^\theta_y,H} f\left(x - \frac{H}{2}\cos\theta, y\right). \tag{4.30}$$

On the other hand, replace $\alpha = 1$ in the sum given by equation (4.29).

$$\begin{aligned}\sum_{m=0}^{1} \begin{pmatrix} 1 \\ m \end{pmatrix} \partial^{1-m}_{u^\theta_x,H} \partial^m_{u^\theta_y,H} f\left(x - \frac{m}{2}H\cos\theta, y + \frac{1-m}{2}H\sin\theta\right) = \\ \partial_{u^\theta_x,H} f\left(x, y + \frac{1}{2}H\sin\theta\right) + \partial_{u^\theta_y,H} f\left(x - \frac{1}{2}H\cos\theta, y\right).\end{aligned}$$

Hence the formula holds for the first directional divided difference. Assume now that the formula holds for $\alpha - 1$. Then,

$$\begin{aligned}&\partial^\alpha_{u_{\theta,H}} f(x,y) = \partial_{u_{\theta,H}}\left(\partial^{\alpha-1}_{u_{\theta,H}} f(x,y)\right) \\ =&\partial_{u_{\theta,H}}\left(\sum_{m=0}^{\alpha-1} \begin{pmatrix} \alpha-1 \\ m \end{pmatrix} \partial^{\alpha-1-m}_{u^\theta_x,H} \partial^m_{u^\theta_y,H} f\left(x - \frac{m}{2}H\cos\theta, y + \frac{\alpha-1-m}{2}H\sin\theta\right)\right) \\ =&\sum_{m=0}^{\alpha-1} \begin{pmatrix} \alpha-1 \\ m \end{pmatrix} \partial_{u_{\theta,H}}\left(\partial^{\alpha-1-m}_{u^\theta_x,H} \partial^m_{u^\theta_y,H} f\left(x - \frac{m}{2}H\cos\theta, y + \frac{\alpha-1-m}{2}H\sin\theta\right)\right).\end{aligned}$$

Using the result from equation (4.30)

$$\partial_{u_\theta,H} f(x,y) = \partial_{u_x^\theta,H} f\left(x, y+\frac{H}{2}\sin\theta\right) + \partial_{u_y^\theta} f\left(x-\frac{H}{2}\cos\theta, y\right),$$

$$\begin{aligned}
&\partial^\alpha_{u_\theta,H} f(x,y) = \\
&= \underbrace{\sum_{m=0}^{\alpha-1}\begin{pmatrix}\alpha-1\\ m\end{pmatrix}\partial_{u_x^\theta,H}\left(\partial^{\alpha-1-m}_{u_x^\theta,H}\partial^m_{u_y^\theta,H} f\left(x-\frac{m}{2}H\cos\theta, y+\frac{\alpha-1-m+1}{2}H\sin\theta\right)\right)}_{(i)} \\
&+ \underbrace{\sum_{m=0}^{\alpha-1}\begin{pmatrix}\alpha-1\\ m\end{pmatrix}\partial_{u_y^\theta,H}\left(\partial^{\alpha-1-m}_{u_x^\theta,H}\partial^m_{u_y^\theta,H} f\left(x-\frac{m+1}{2}H\cos\theta, y+\frac{\alpha-1-m}{2}H\sin\theta\right)\right)}_{(ii)}
\end{aligned}$$

$$\begin{aligned}
(i) &= \begin{pmatrix}\alpha-1\\ 0\end{pmatrix}\partial^\alpha_{u_x^\theta,H} f\left(x, y+\frac{\alpha}{2}H\sin\theta\right) \\
&+ \sum_{m=1}^{\alpha-1}\begin{pmatrix}\alpha-1\\ m\end{pmatrix}\partial^{\alpha-m}_{u_x^\theta,H}\partial^m_{u_y^\theta,H} f\left(x-\frac{m}{2}H\cos\theta, y+\frac{\alpha-m}{2}H\sin\theta\right)
\end{aligned}$$

Changing $m \to m+1$ in (ii) gives

$$\begin{aligned}
(ii) &= \sum_{m=1}^{\alpha-1}\begin{pmatrix}\alpha-1\\ m-1\end{pmatrix}\partial^{\alpha-m}_{u_x^\theta,H}\partial^m_{u_y^\theta,H} f\left(x-\frac{m}{2}H\cos\theta, y+\frac{\alpha-m}{2}H\sin\theta\right) \\
&+ \begin{pmatrix}\alpha-1\\ \alpha-1\end{pmatrix}\partial^\alpha_{u_y^\theta,H} f\left(x-\frac{\alpha}{2}H\cos\theta, y\right)
\end{aligned}$$

$$\begin{aligned}
&(i)+(ii) = \\
&\sum_{m=1}^{\alpha-1}\underbrace{\left(\begin{pmatrix}\alpha-1\\ m\end{pmatrix}+\begin{pmatrix}\alpha-1\\ m-1\end{pmatrix}\right)}_{=\binom{\alpha}{m}}\partial^{\alpha-m}_{u_x^\theta,H}\partial^m_{u_y^\theta,H} f\left(x-\frac{m}{2}H\cos\theta, y+\frac{\alpha-m}{2}H\sin\theta\right) \\
&+ \underbrace{\begin{pmatrix}\alpha-1\\ 0\end{pmatrix}}_{=\binom{\alpha}{0}}\partial^\alpha_{u_x^\theta,H} f\left(x, y+\frac{\alpha}{2}H\sin\theta\right) + \underbrace{\begin{pmatrix}\alpha-1\\ \alpha-1\end{pmatrix}}_{=\binom{\alpha}{\alpha}}\partial^\alpha_{u_y^\theta,H} f\left(x-\frac{\alpha}{2}H\cos\theta, y\right),
\end{aligned}$$

which gives the formula

$$\partial^\alpha_{u_\theta,H} f(x,y) = \sum_{m=0}^{\alpha}\begin{pmatrix}\alpha\\ m\end{pmatrix}\partial^{\alpha-m}_{u_x^\theta,H}\partial^m_{u_y^\theta,H} f\left(x-\frac{m}{2}H\cos\theta, y+\frac{\alpha-m}{2}H\sin\theta\right).$$

□

The previous Lemma allows for calculating directional divided differences in terms of the basis building vectors. Now the relation between the divided differences along the arc length parameter and directional divided differences is discussed.

Lemma 4.2.2. *For a B-Spline of the form of equation* (4.18)*, the (scaled) directional divided differences along the line* Γ *are equal to the (scaled) divided differences of the univariate B-Spline along the arc length parameter, i.e.,*

$$\partial^{\alpha}_{u_\theta,H}\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}(x,y)=\partial^{\alpha}_{H}\psi^{(\ell-\alpha)}_{H}(t),\quad u_\theta=(\cos\theta,\sin\theta). \tag{4.31}$$

Proof. Start with the first order divided difference: $\alpha=1$.

$$\partial_{u_\theta,H}\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}(x,y)=\frac{\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}\left(x+\frac{H}{2}\cos\theta,y+\frac{H}{2}\sin\theta\right)}{H}-\frac{\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}\left(x-\frac{H}{2}\cos\theta,y-\frac{H}{2}\sin\theta\right)}{H}$$

Examining the first term:

$$\begin{aligned}\frac{\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}\left(x+\frac{H}{2}\cos\theta,y+\frac{H}{2}\sin\theta\right)}{H}&=\frac{\psi^{(\ell-\alpha)}_{H}\left(\Gamma^{-1}\left(x+\frac{H}{2}\cos\theta,y+\frac{H}{2}\sin\theta\right)\right)}{H}\\&\overset{\substack{(4.16)\\ \downarrow}}{=}\frac{\psi^{(\ell-\alpha)}_{H}\left(\left(x+\frac{H}{2}\cos\theta\right)\cos\theta+\left(y+\frac{H}{2}\sin\theta\right)\sin\theta\right)}{H}\\&=\frac{\psi^{(\ell-\alpha)}_{H}\left(x\cos\theta+\frac{H}{2}\cos^2\theta+y\sin\theta+\frac{H}{2}\sin^2\theta\right)}{H}\\&=\frac{\psi^{(\ell-\alpha)}_{H}\left(t+\frac{H}{2}\right)}{H}.\end{aligned}$$

The second term gives a similar result:

$$\frac{\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}\left(x-\frac{H}{2}\cos\theta,y-\frac{H}{2}\sin\theta\right)}{H}=\frac{\psi^{(\ell-\alpha)}_{H}\left(t-\frac{H}{2}\right)}{H}.$$

Hence,

$$\partial_{u_\theta,H}\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}(x,y)=\frac{\psi^{(\ell-\alpha)}_{H}\left(t+\frac{H}{2}\right)}{H}-\frac{\psi^{(\ell-\alpha)}_{H}\left(t-\frac{H}{2}\right)}{H}=\partial_H\psi^{(\ell-\alpha)}(t).$$

For higher order divided differences, recall that the α-th divided difference of a function is defined by

$$\partial^{\alpha}f=\partial(\partial^{\alpha-1}f),\quad \alpha>1. \tag{4.32}$$

Assume

$$\partial^{\alpha-1}_{u_\theta,H}\tilde{\psi}^{(\ell-\alpha)}_{H}(x,y)=\partial^{\alpha-1}_{H}\psi^{(\ell-\alpha)}(t),\quad 1<\alpha\le\ell.$$

Then,

$$\partial^{\alpha}_{u_\theta,H}\tilde{\psi}^{(\ell-\alpha)}_{H}(x,y)=\partial_{u_\theta,H}\left(\partial^{\alpha-1}_{u_\theta,H}\tilde{\psi}^{(\ell-\alpha)}_{H}(x,y)\right)\quad 1<\alpha\le\ell.$$

$$\begin{aligned}&\partial_{u_\theta,H}\left(\partial^{\alpha-1}_{u_\theta,H}\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}(x,y)\right)=\\&=\frac{\partial^{\alpha-1}_{u_\theta,H}\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}\left(x+\frac{H}{2}\cos\theta,y+\frac{H}{2}\sin\theta\right)}{H}-\frac{\partial^{\alpha-1}_{u_\theta,H}\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}\left(x-\frac{H}{2}\cos\theta,y-\frac{H}{2}\sin\theta\right)}{H}\\&=\frac{\partial^{\alpha-1}_{H}\psi^{(\ell-\alpha)}_{H}\left(t+\frac{H}{2}\right)}{H}-\frac{\partial^{\alpha-1}_{H}\psi^{(\ell-\alpha)}_{H}\left(t-\frac{H}{2}\right)}{H}\\&=\partial^{\alpha}_{H}\psi^{(\ell-\alpha)}_{H}(t).\end{aligned}$$

□

Corollary 4.2.1. *The derivatives of the B-Splines from Definition 4.2.1 can be expressed as a sum of directional divided differences through the formula:*

$$D^\alpha \tilde{\psi}^{(\ell)}_{\theta,H}(x,y) =$$
$$\sin^{\alpha_2}\theta\cos^{\alpha_1}\theta\sum_{k=0}^{\alpha}\binom{\alpha}{k}\partial^{\alpha-k}_{u^\theta_x,H}\partial^{k}_{u^\theta_y,H}\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}\left(x-\frac{k}{2}H\cos\theta, y+\frac{\alpha-k}{2}H\sin\theta\right).$$

Proof. Differentiating the B-Spline gives

$$D^\alpha \tilde{\psi}^{(\ell)}_{\theta,H}(x,y) = \sin^{\alpha_2}\theta\cos^{\alpha_1}\theta\partial^\alpha_H\psi^{(\ell-\alpha)}_H(t)$$

(see equation (4.24)). Now use Lemma 4.2.2 to express the derivatives in terms of the directional divided differences:

$$\partial^\alpha_H\psi^{(\ell-\alpha)}_H(t) = \partial^\alpha_{u_\theta,H}\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}(x,y).$$

Using Lemma 4.2.1 completes the proof. □

Remark 4.2.2. *Corollary 4.2.1 shows that the derivatives of the B-Spline can be computed as a combination of directional divided differences using the original Cartesian basis.*

Finally, for a smooth function v,

$$D^\alpha\tilde{\psi}^{(\ell)}_{\theta,H}\star v = \cos^{\alpha_1}\theta\sin^{\alpha_2}\theta\cdot\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}\star\partial^\alpha_{u_\theta,H}v, \quad \alpha_1+\alpha_2=\alpha. \tag{4.33}$$

Since the convolution and the divided differences are linear operators,

$$\begin{aligned}
D^\alpha\left(\tilde{\psi}^{(\ell)}_{\theta,H}\star v\right)(x,y) = \left(D^\alpha\tilde{\psi}^{(\ell)}_{\theta,H}\right)\star v(x,y) &= \cos^{\alpha_1}\theta\sin^{\alpha_2}\theta\cdot\left(\partial^\alpha_{u_\theta,H}\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}\star v\right)(x,y)\\
&= \cos^{\alpha_1}\theta\sin^{\alpha_2}\theta\int_{\mathbb{R}}\int_{\mathbb{R}}\partial^\alpha_{u_\theta,H}\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}(x-\xi_x,y-\xi_y)v(\xi_x,\xi_y)d\xi_x d\xi_y\\
&= \cos^{\alpha_1}\theta\sin^{\alpha_2}\theta\int_{\mathbb{R}}\int_{\mathbb{R}}\partial^\alpha_{u_\theta,H}\tilde{\psi}^{(\ell-\alpha)}_{\theta,H}(\eta_x,\eta_y)v(x-\eta_x,y-\eta_y)d\eta_x d\eta_y\\
&= \cos^{\alpha_1}\theta\sin^{\alpha_2}\theta\int_{\mathbb{R}}\int_{\mathbb{R}}\psi^{(\ell-\alpha)}_{\theta,H}(\eta_x,\eta_y)\partial^\alpha_{u_\theta,H}\tilde{v}(x-\eta_x,y-\eta_y)d\eta_x d\eta_y\\
&= \cos^{\alpha_1}\theta\sin^{\alpha_2}\theta\left(\psi^{(\ell-\alpha)}_{\theta,H}\star\partial^\alpha_{u_\theta,H}v\right)(x,y).
\end{aligned}$$

4.3 Error Estimates for SIAC Line Filters

Now that the foundations for line filtering have been discussed, superconvergent error estimates are given in the following theorem.

Theorem 4.3.1. *Let u be the exact solution to problem* (2.1) *with $d=2$ and assume periodic boundary conditions. Let u_h be the DG approximation over a uniform mesh and denote by h_x and h_y the DG mesh size. Consider the line filter $K^{(2k+1,k+1)}_{\Gamma,H}$ along $\Gamma(t)=t(\cos\theta,\sin\theta)$, θ fixed. If $\theta=\arctan\left(\frac{h_y}{h_x}\right)$ and $H=h_x\cos\theta+h_y\sin\theta$, then:*

$$\left\|u-K^{(2k+1,k+1)}_{\Gamma,H}\star u_h\right\|_{0,\Omega_0}\le Ch^{2k+1}. \tag{4.34}$$

Proof. Write

$$\left\| u - K_H^{(2k+1,k+1)} \star u_h \right\|_{0,\Omega_0} \leq \underbrace{\left\| u - K_H^{(2k+1,k+1)} \star u \right\|_{0,\Omega_0}}_{\Theta_1} + \underbrace{\left\| K_H^{(2k+1,k+1)} \star (u - u_h) \right\|_{0,\Omega_0}}_{\Theta_2}.$$

Since the line filter preserves the polynomial reproduction property:

$$K_{\Gamma,H}^{(2k+1,k+1)} \star x^p = x^p, \quad p = 0, \dots, 2k, \tag{4.35}$$

the first term is bounded as shown in the proof of Theorem 4.1.1 using equation (4.4).

The second term needs to be written in terms of the divided differences with an expression similar to

$$D^\alpha \left(K_H^{(2k+1,k+1)} \star (u - u_h) \right) = K_H^{(2k+1,k+1-\alpha)} \star \partial_H^\alpha (u - u_h)$$

in order to obtain a bound of the form of:

$$\Theta_2 \leq C_1 C_2 \sum_{|\alpha| \leq k+1} \|\partial_h^\alpha (u - u_h)\|_{-(k+1),\Omega_1}. \tag{4.36}$$

Let $\ell = k+1$, $r = 2k$, and denote the error by $e = u - u_h$. The kernel is a linear combination of B-Splines so it suffices to study one B-Spline alone. Lemma 4.2.1 allows to write

$$D^\alpha \tilde{\psi}_{\theta,H}^{(\ell)} \star e =$$

$$\sin^{\alpha_1} \theta \cos^{\alpha_2} \theta \tilde{\psi}_{\theta,H}^{(\ell)} \star \left(\sum_{m=0}^{\alpha} \binom{\alpha}{m} \partial_{u_x^\theta,H}^{\alpha-m} \partial_{u_y^\theta,H}^{m} e \left(x - \frac{m}{2} H \cos\theta, y + \frac{\alpha - m}{2} H \sin\theta \right) \right),$$

where $u_x^\theta = (\cos\theta, 0)$ and $u_y^\theta = (0, \sin\theta)$.

Note 4.3.1. *Since*

$$\theta = \arctan\left(\frac{h_y}{h_x}\right) \quad \textit{and} \quad H = h_x \cos\theta + h_y \sin\theta,$$

the following equations hold

$$H = \frac{h_x}{\cos\theta} = \frac{h_y}{\sin\theta}. \tag{4.37}$$

The second equality

$$\frac{h_x}{\cos\theta} = \frac{h_y}{\sin\theta},$$

comes directly from the definition of θ. To show the first one, assume $H = \frac{h_x}{\cos\theta}$. Then,

$$\begin{aligned} H = & \frac{h_x}{\cos\theta} \Rightarrow H \cos^2\theta = \cos\theta \cdot h_x \\ H\cos^2\theta = & H(1 - \sin^2\theta) \\ H = & \cos\theta \cdot h_x + \underbrace{H \sin^2\theta}_{H = h_y/\sin\theta} = h_x \cos\theta + h_y \sin\theta. \end{aligned}$$

This allows us to write the directional divided differences of the error function in the canonical basis $\mathcal{B}_1 = \{\boldsymbol{e}_1, \boldsymbol{e}_2\}$ using the mesh size:

$$\begin{aligned}
\partial_{u_\theta^x, H} f(x,y) &= \frac{1}{H}\left(f\left(x + \frac{H}{2}\cos\theta, y\right) - f\left(x - \frac{H}{2}\cos\theta, y\right)\right) \\
&= \frac{1}{H}\left(f\left(x + \frac{h_x}{2}, y\right) - f\left(x - \frac{h_x}{2}, y\right)\right) \\
&= \cos\theta \cdot \partial_{\mathbf{e}_1, h_x} f(x,y).
\end{aligned}$$

Analogously, the second direction gives

$$\partial_{u_\theta^y, H} f(x,y) = \sin\theta \cdot \partial_{\mathbf{e}_2, h_y} f(x,y).$$

Hence

$$\begin{aligned}
&D^\alpha \tilde{\psi}_{\theta,H}^{(\ell)} \star e = \\
&= \sin^{\alpha_1+1}\theta \cos^{\alpha_2+1}\theta \tilde{\psi}_{\theta,H}^{(\ell)} \star \left(\sum_{m=0}^{\alpha} \binom{\alpha}{m} \partial_{\mathbf{e}_1,h_x}^{\alpha-m} \partial_{\mathbf{e}_2,h_y}^{m} e\left(x - \frac{m}{2}h_x, y + \frac{\alpha-m}{2}h_y\right)\right),
\end{aligned}$$

giving:

$$\Theta_2 \le C_1 C_2 C(\theta) \sum_{|\alpha| \le k+1} \left\| \sum_{m=0}^{\alpha} \binom{\alpha}{m} \partial_{\mathbf{e}_1,h_x}^{\alpha-m} \partial_{\mathbf{e}_2,h_y}^{m} e \right\|_{-(k+1),\Omega_1}. \tag{4.38}$$

The rest of the proof follows from Theorem 4.1.1. $\square$

Remark 4.3.1. *When a B-Spline is differentiated, as a consequence of the chain rule a $\sin\theta$ or $\cos\theta$ term appears. As a result, the constant term in equation (4.38) now includes the multiplying factor*

$$\sin^{\alpha_1+2}\theta \cos^{\alpha_1+1}$$

which is always less than one (and decreasing with every power) since the rotation angle is defined by $\arctan(h_y/h_x)$. This means that the constant in front of equation (4.34) can actually be reduced. In the numerical experiments presented in the following section, there are cases where the line filter outperforms the 2D axis aligned filter in terms of error reduction.

4.4 Numerical Results

The numerical experiments were done for the 2D transport equation:

$$\begin{cases} u_t + u_x + u_y = 0, & (x,y) \in [0, 2\pi]^2,\ t \in [0,T] \\ u_0(x,y) = u(x,0) \end{cases} \tag{4.39}$$

at final time $T = 2$. The unfiltered solution was obtained with a DG scheme using an upwind flux over an uniform mesh. In the following, the various aspects of filtering will be discussed: smoothness and accuracy enhancement, including error reduction.

4.4.1 Recovering Smoothness

Since line filtering implies post-processing along a single direction, one can expect that it is only along that line where the filtered solution gains smoothness. Consider the L^2 projection of the function

$$u(x, y) = \sin(x), \quad (x, y) \in [0, 2\pi]^2.$$

Since the field depends only in one variable, the line filters are expected to behave similarly in every direction and identically with respect to each other, provided the appropriate scaling is selected. Figure 4.2 shows the error profiles corresponding to a horizontal and diagonal domain slice before and after line filtering along the $\theta = 0, \pi/4$ and $3\pi/4$ directions. The plots highlight the importance of the scaling choice; observe how the $\theta = 0$ line filter only recovers smoothness when the H is set equal to the mesh size. In Theorem 4.3.1 it was shown that the scaling should be set to $H = \arctan(h_x/h_y) = \sqrt{2}$ (the mesh is made of uniform squares). The slice plots show how for the rotations $\theta = \pi/4$ and $\theta = 3\pi/4$, a smaller scaling results in a oscillatory error profile. Regarding the magnitude of the error, despite the lack of smoothness in the filtered solution, there is general error reduction for all scalings and rotations. Table 4.1 shows the global L^2 errors and orders before and after applying these line filter over the entire field. As expected, using the right angle-scaling pairs results in the same convergence rate and error.

The next study was done for the DG solution to Problem 4.39 and introducing multivariate fields through the initial condition:

$$u(x, t) = \sin x \cos y.$$

Figure 4.3 shows different error profiles corresponding to a horizontal, vertical and diagonal domain slice. The plots show how the zero rotation produces a smooth solution along the filtering direction only (horizontal) and without much error reduction. On the other hand, rotating the filter produces a smooth profile in all directions. Both the $\pi/4$ and $3\pi/4$-line filters are able to recover smoothness and clearly reduce the error from the DG solution. Figure 4.4 shows the error profiles for the same problem but using the initial condition:

$$u(x, y) = \sin(x + y).$$

Again, for the zero rotation line filter, there is only smoothness recovery along the horizontal cut. Furthermore, just like the previous case, the magnitude of the error of the filtered solution is very close to the original DG solution. The performance of the $3\pi/4$ rotation is shown for two different scalings. Observe how although the value $\mu = 1$ results in error reduction, the filtered solution is still oscillatory whilst using $\mu = \sqrt{2}$ not only produces a smooth solution in all directions but also the error decreases.

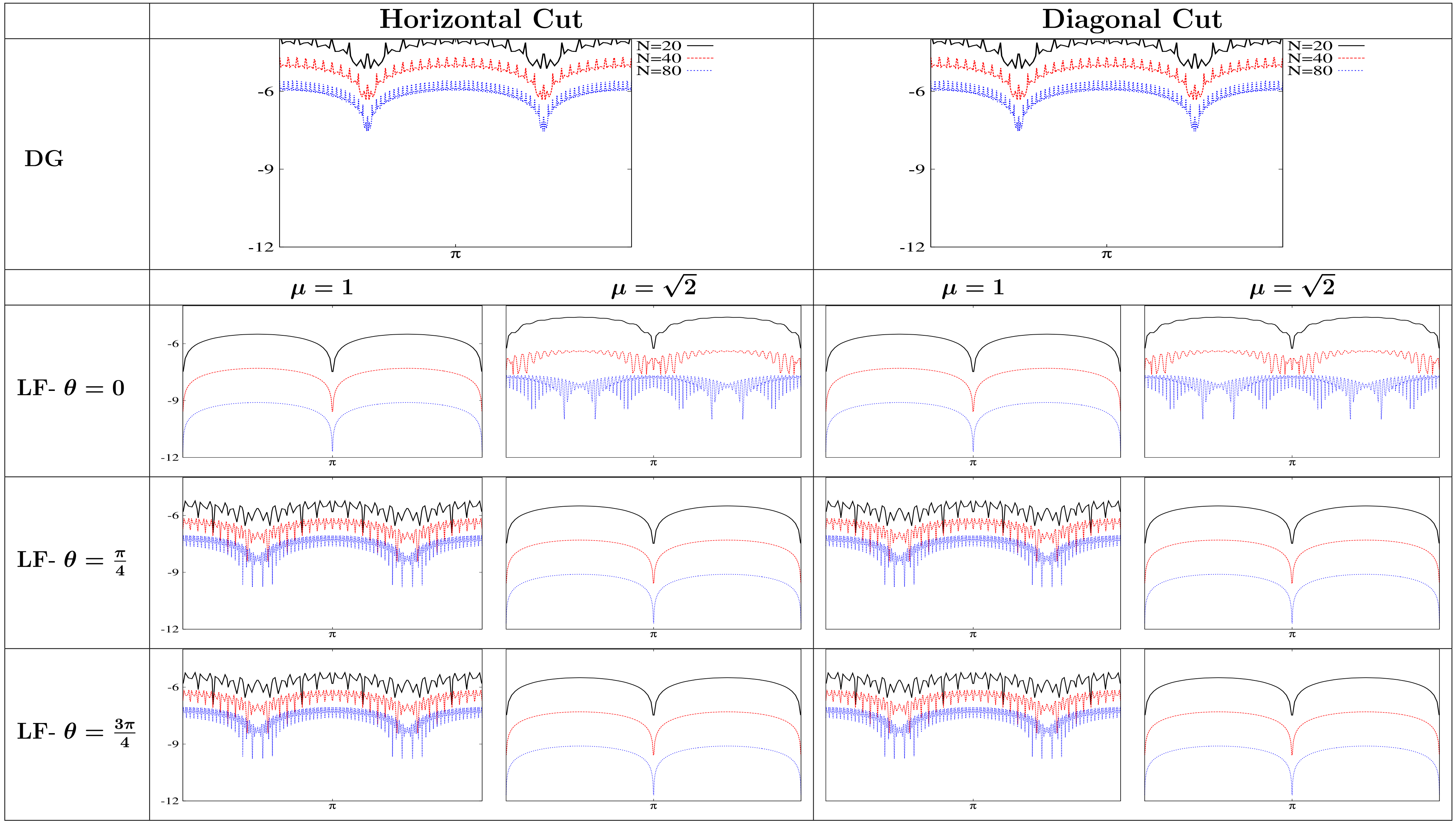


Figure 4.2: Point-wise error profile slices along a horizontal and diagonal direction applying several Line Filters on the DG solution to Problem (4.39) with initial condition $u_0(x, y) = \sin x$ and using $\mathbb{P}^2$ polynomials.

	Unfiltered		**Line Filtered**						
			$\boldsymbol{H = \mu h}$	$\boldsymbol{\theta = 0}$		$\boldsymbol{\theta = \pi/4}$		$\boldsymbol{\theta = 3\pi/4}$	
N	L^2-Error	Order	μ	L^2-Error	Order	L^2-Error	Order	L^2-Error	Order
				$\mathbb{P}^1$					
20	2.6e-03	-	0.9	8.2e-05	-	6.6e-04	-	6.6e-04	-
			1	9.3e-05	-	4.0e-04	-	4.0e-04	-
			$\sqrt{2}$	3.5e-04	-	9.3e-05	-	9.3e-05	-
40	6.5e-04	2.00	0.9	1.0e-05	3.03	1.7e-04	2.00	1.7e-04	2.00
			1	5.8e-06	3.99	9.9e-05	2.03	9.9e-05	2.03
			$\sqrt{2}$	4.3e-05	3.02	5.8e-06	3.99	5.8e-06	3.99
80	1.6e-04	2.00	0.9	2.1e-06	2.26	4.2e-05	2.00	4.2e-05	2.00
			1	3.6e-07	4.00	2.5e-05	2.01	2.5e-05	2.01
			$\sqrt{2}$	9.7e-06	2.15	3.6e-07	4.00	3.6e-07	4.00
				$\mathbb{P}^2$					
20	6.9e-05	-	0.9	1.2e-06	-	6.6e-06	-	6.6e-06	-
			1	2.2e-06	-	3.0e-06	-	3.0e-06	-
			$\sqrt{2}$	1.8e-05	-	2.2e-06	-	2.2e-06	-
40	8.6e-06	3.00	0.9	2.1e-08	5.81	8.2e-07	3.01	8.2e-07	3.01
			1	3.5e-08	5.98	3.6e-07	3.07	3.6e-07	3.07
			$\sqrt{2}$	3.0e-07	5.89	3.5e-08	5.98	3.5e-08	5.98
80	1.1e-06	3.00	0.9	1.1e-09	4.26	4.5e-08	4.21	4.5e-08	4.21
			1	5.6e-10	5.99	1.0e-07	1.81	1.0e-07	1.81
			$\sqrt{2}$	1.3e-08	4.55	5.6e-10	5.99	5.6e-10	5.99
				$\mathbb{P}^3$					
20	1.4e-06	-	0.9	3.0e-08	-	3.4e-08	-	3.4e-08	-
			1	6.9e-08	-	1.3e-08	-	1.3e-08	-
			$\sqrt{2}$	1.1e-06	-	6.9e-08	-	6.9e-08	-
40	8.5e-08	4.00	0.9	1.2e-10	7.96	2.1e-09	4.03	2.1e-09	4.03
			1	2.7e-10	7.97	7.0e-10	4.21	7.0e-10	4.21
			$\sqrt{2}$	4.4e-09	7.94	2.7e-10	7.97	2.7e-10	7.97
80	5.3e-09	4.00	0.9	1.0e-12	6.89	1.3e-10	4.01	1.3e-10	4.01
			1	1.1e-12	7.98	4.3e-11	4.04	4.3e-11	4.04
			$\sqrt{2}$	1.9e-11	7.86	1.1e-12	7.98	1.1e-12	7.98
				$\mathbb{P}^4$					
20	2.2e-08	-	0.9	7.7e-10	-	1.1e-10	-	1.1e-10	-
			1	2.2e-09	-	8.1e-11	-	8.1e-11	-
			$\sqrt{2}$	6.7e-08	-	2.2e-09	-	2.2e-09	-
40	6.7e-10	5.00	0.9	7.2e-13	10.05	3.3e-12	5.09	3.3e-12	5.09
			1	2.2e-12	9.99	1.4e-12	5.88	1.2e-12	6.06
			$\sqrt{2}$	7.0e-11	9.91	2.2e-12	9.99	2.2e-12	9.99
80	2.1e-11	5.00	0.9	5.6e-14	3.68	1.1e-13	4.84	1.1e-13	4.84
			1	5.5e-14	5.30	9.6e-14	3.83	7.8e-14	3.95
			$\sqrt{2}$	3.5e-14	10.97	5.0e-14	5.42	5.0e-14	5.43

Table 4.1: L^2 errors and orders before and after applying several Line Filters to the L^2 projection of the function $u(x, y) = \sin(x)$ on $\Omega = [0, 2\pi]^2$.

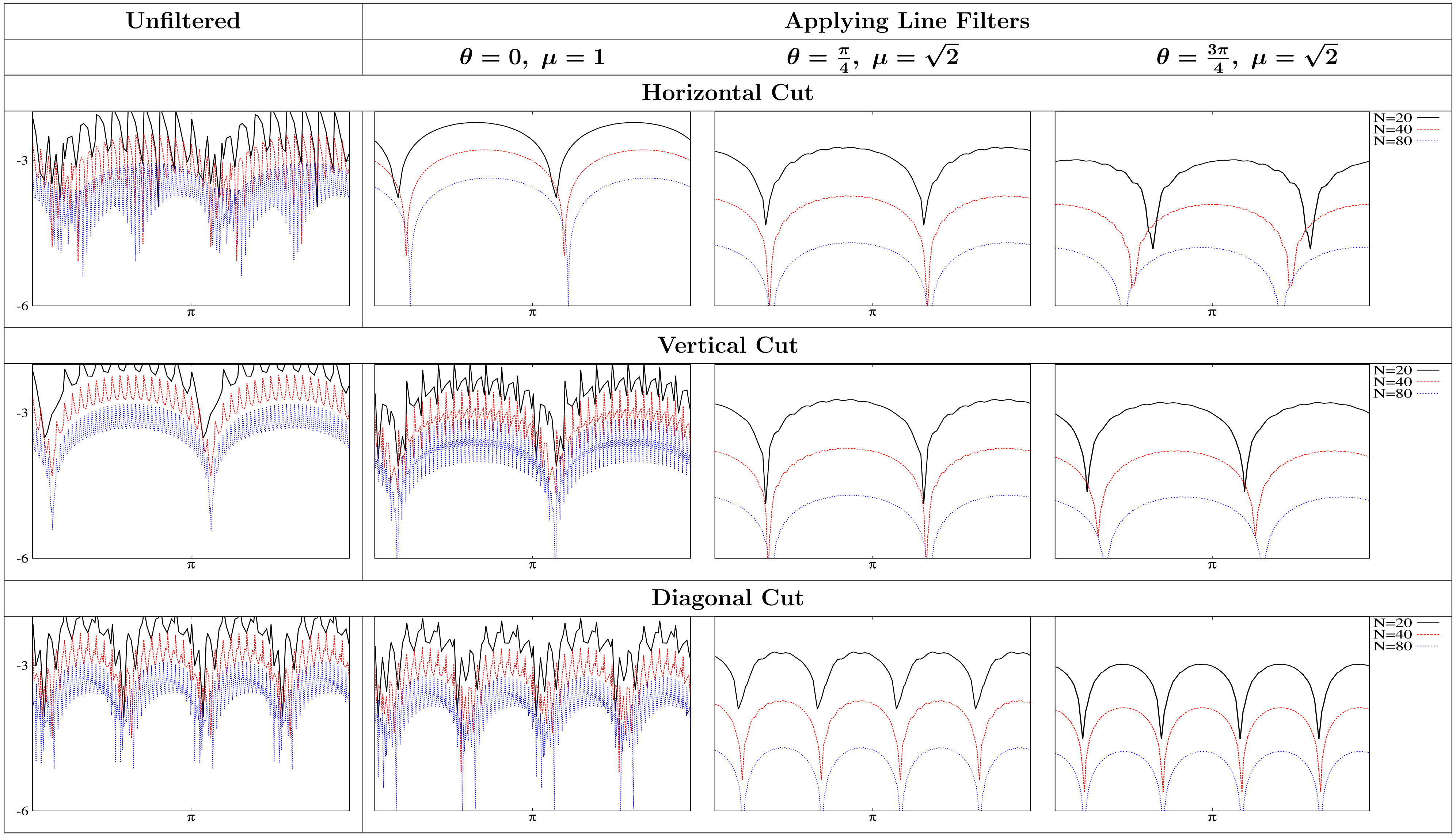


Figure 4.3: Point-wise error profile slices along several directions applying three different Line Filters on the DG solution to Problem (4.39) with initial condition $u_0(x, y) = \sin x \cos y$ and using $\mathbb{P}^1$ polynomials.

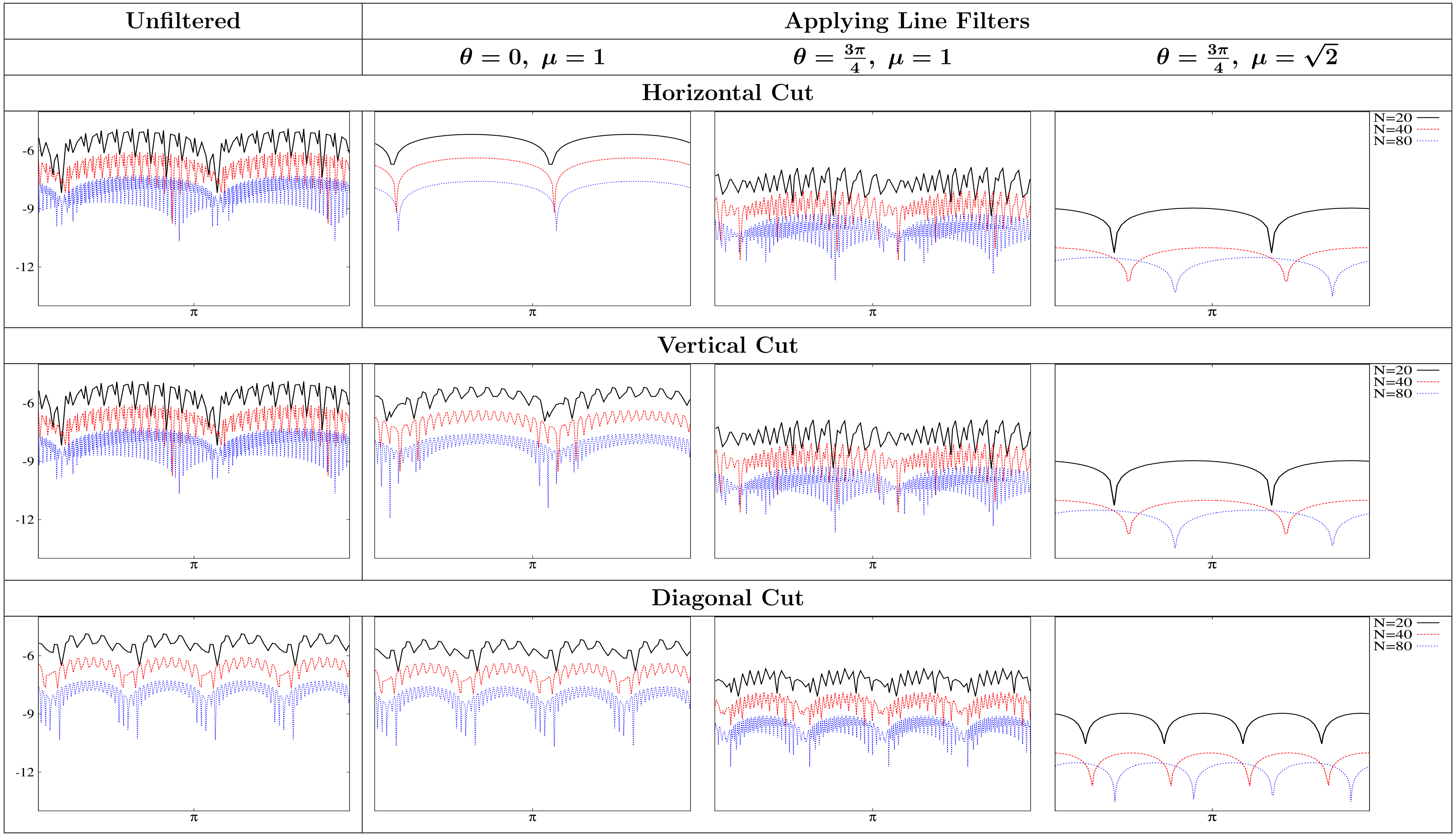


Figure 4.4: Point-wise error profile slices along several directions applying three different Line Filters on the DG solution to Problem (4.39) with initial condition $u_0(x, y) = \sin(x + y)$ and using $\mathbb{P}^3$ polynomials.

4.4.2 Accuracy Enhancement

It has just been discussed the line filters ability to gain smoothness if choosing the correct rotation and scaling. Now attention is turned towards global error and convergence order. The goal of this study was to compare the line filtered solution with the DG solution. Some numerical results include the original tensor product filter aligned with the Cartesian axis. However, since this chapter investigates the performance of line filters, the experiments for the 2D filter stop at 40×40 elements. Although this may not seem complete, since the results show three polynomial degrees, it suffices to give insight into the filters behaviour relative to each other.

The performance of line filters was studied in several ways in order to understand which features of the solution are more relevant for maximum error reduction. In particular, the line filters performance was tested subject to the mesh type, initial condition and flow direction. Consider the DG solution at time zero and project the function

$$u(x, y) = \sin(x + y)$$

onto the space $\Omega = [0, 2\pi]^2$ using a uniform square mesh. Figure 4.5 shows the contour-line error profiles using three different line filters. The second column of plots show the performance of the horizontal line filter. The results show that this filter is unable to improve the smoothness or the size of the error from the original solution. The $\pi/4$ rotation effectively reduces the oscillations for the larger scaling $\mu = \sqrt{2}$ but at the expense of less error reduction compared to the value $\mu = 1$. In fact, the $\mathbb{P}^4$ case shows that the pair $\theta = \pi/4,\ \mu = \sqrt{2}$ produce a solution with greater error than the original one. The rotation $\theta = 3\pi/4$, on the other hand, exhibits excellent performance for the theoretical scaling $\mu = \sqrt{2}$, both in terms of smoothness and error reduction. The other scaling indeed reduces the error of the original solution (though not as much as if using $\mu = \sqrt{2}$) but the filtered solution still exhibits oscillations. Table 4.2 shows the global L^2 errors and convergence rates. Both the $\pi/4$ and $3\pi/4$ rotations achieve the expected superconvergence when using the scaling $\mu = \sqrt{2}$. This is clearer for the $\theta = \pi/4$ case which attains $2k + 2$ order for all polynomial degrees (k being the polynomial degree used for the approximation). The $\theta = 3\pi/4$ convergence rates drop at the higher degree polynomials but this clearly due to round off errors since the global errors are already at 10^{-14}, *i.e.*, machine double precision limits. An important remark for the zero rotation is that the filtered solution results in lower error but the convergence rates stay at the original $k + 1$ order.

Figure 4.6 and Table 4.3 show numerical results comparing the line filter with the original tensor product filter for the DG solution to Problem (4.39). Two initial conditions were considered: $u(x, y) = \sin(x + y)$ and $u(x, y) = \sin x \cos y$. For the first initial condition, as the filters order increase, the $3\pi/4$-line filter has excellent behaviour and actually outperforms the 2D filter in terms of error reduction. Regarding convergence orders, it seems like both the tensor product filter and the $\pi/4$-line filter have a faster rate than the $3\pi/4$-line filter. However, for the second initial condition,

$u(x, y) = \sin x \cos y$, the three filters exhibit similar orders of accuracy. In this case, the tensor product filter has consistently a lower error. Nevertheless, in all cases, the filtered solutions clearly reduce the error compared to the original DG solution.

The first set of numerical experiments on line filtering suggested that the best orientation is the $3\pi/4$ direction. This orientation is also the line of symmetry of the wave $u(x, y) = \sin(x + y)$ and it is also tangent to the flow direction $u_t + u_x + u_y = 0$. Hence, two additional studies were considered. The first corresponds to the same DG solution to Problem (4.39) but this time using the initial condition

$$u(x, y) = \sin(x + 3y).$$

The contour-line plots in Figure 4.7 show the error profiles before and after filtering using the $3\pi/4$-line filter compared to the $\theta = \arctan(3)$ and its perpendicular direction $(u(x, y) = \sin(x + 3y),\ \tan(\theta) = 3/1)$. Table 4.4 shows the global L^2 errors and orders. The $3\pi/4$ rotation has an overall better performance, indicating that the initial condition plays a minor role on the filter orientation choice. In terms of the order of accuracy, both the orientations $\theta = \pi/4$ and $\theta = \arctan(3)$ have a faster convergence rate than the perpendicular directions. This can be observed as well in Table 4.3 for the initial condition $u(x, y) = \sin(x + y)$ and the pair $\theta = \pi/4,\ 3\pi/4$. However, for the $\theta = \pi/4$ case, this occurs at the expense of lower error reduction as shown, for example, for the $\mathbb{P}^3$ case in Table 4.4.

The next question was whether the flow direction could result in better filtering. Therefore, the following problem was considered:

$$\begin{cases} u_t + 1.3u_x + 0.8u_y = 0 \\ \qquad u(x, y, 0) = \sin x \cos y. \end{cases}$$

Two pairs of Line filters were applied corresponding to the flow direction and its tangent as well as the mesh based Line filters using the $\pi/4$ and $3\pi/4$ orientations. The flow direction based filters were tested for three different scalings. In addition to the value $\mu = 1$, an alternative scaling was selected using the B-Splines parametrization for line filtering:

$$H = h_x \cos\theta + h_y \sin\theta.$$

Since the mesh consists of uniform squares, $h_x = h_y$ and $\mu = \cos\theta + \sin\theta$. Finally, using the flow direction vector $\vec{u} = (1.3, 0.8)$, the vector magnitude was considered, giving the value $\mu = \sqrt{0.8^2 + 1.3^2}$. Figure 4.8 shows contour-line plots of the error profiles before and after applying these filters. The plots suggest that as the polynomial degree increases, the flow direction aligned filter (and its tangent direction) have the greatest error reduction (using the scaling $\mu = \sin\theta + \cos\theta$). On the other hand, the $3\pi/4$ Line filter has a smoother profile for all polynomial degrees. The flow direction based filters exhibit an interesting behaviour: the rotation $\theta = \arctan(0.8/1.3)$ seems to recover smoothness horizontally whereas its perpendicular orientation reduces the oscillations more towards the vertical direction. Table 4.5 shows the global L^2 errors and orders for the previous filters and also including the $\pi/4$ rotation. The differences in the performance between each orientation and its perpendicular is very small and decreasing with each polynomial order. Consider the $\mathbb{P}^3$ case for the rotations $\pi/4$ and $3\pi/4$ which have identical values. Furthermore, the global errors suggest the same as demonstrated in the plots in Figure 4.8: asymptotically, choosing a direction in relation to the flow results in greater error reduction than using a mesh based orientation. However, the convergence rates of these orientations are not as high as the $3\pi/4$ and $\pi/4$ rotations.

Finally, in the last experiment, the mesh was transformed into uniform rectangular elements of size $h_x = 2h_y$. In this case, by Theorem 4.3.1, the rotation angle should be chosen to $\theta = \arctan(1/2)$ or its supplementary angle $\theta = \pi - \arctan(1/2)$ and the scaling should be $H = \frac{\sqrt{5}h_x}{2}$ or equivalently $H = \sqrt{5}h_y$. Figure 4.9 shows the error profiles for the DG solution and the filtered solution comparing the $\theta = 3\pi/4$ with the $\theta = \pi - \arctan(1/2)$ rotation. The plots include three different scalings corresponding to $\mu = \sqrt{5}/2,\ \sqrt{2},\ \sqrt{5}$. These plots highlight the importance of the support size for smoothness recovery. Observe how the value $\mu = \sqrt{2}$ is not able to eliminate the oscillations for the $\theta = \pi - \arctan(1/2)$ rotation or how the other two values $\mu = \sqrt{5}/2$ and $\mu = \sqrt{5}$ affect the solution for the $3\pi/4$-line filter. The performance of these filters is almost identical for the right angle-scaling pairs. This can be verified in Table 4.6 which shows the global errors and convergence order for the L^2 norm.

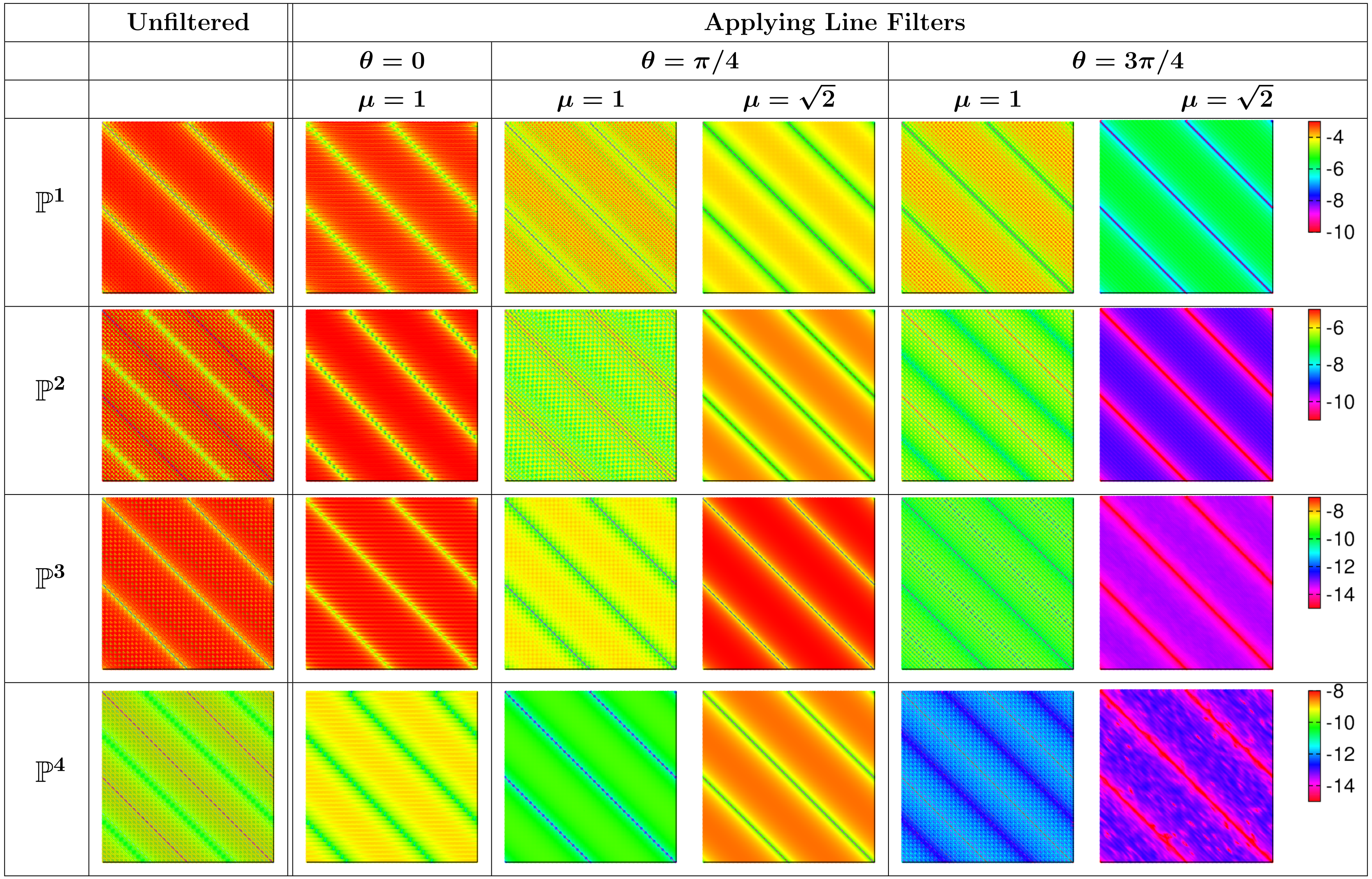


Figure 4.5: Contour line error plots (log) before and after applying Line Filters on the L^2 projection of the function $u(x, y) = \sin(x+y)$ on $\Omega = [0, 2\pi]^2$ and using 40×40 elements.

	Unfiltered		Line Filtered						
			$H = \mu h$	$\theta = 0$		$\theta = \pi/4$		$\theta = 3\pi/4$	
N	L^2-Error	Order	μ	L^2-Error	Order	L^2-Error	Order	L^2-Error	Order
				$\mathbb{P}^1$					
			0.9	2.6e-03	-	1.0e-03	-	9.3e-04	-
20	3.7e-03	-	1	2.6e-03	-	7.3e-04	-	5.5e-04	-
			$\sqrt{2}$	2.6e-03	-	1.2e-03	-	3.4e-05	-
			0.9	6.5e-04	2.00	2.4e-04	2.10	2.3e-04	1.99
40	9.2e-04	2.00	1	6.5e-04	2.00	1.5e-04	2.32	1.4e-04	2.00
			$\sqrt{2}$	6.5e-04	2.01	7.8e-05	3.96	2.1e-06	3.99
			0.9	1.6e-04	2.00	5.9e-05	2.02	5.9e-05	2.00
80	2.3e-04	2.00	1	1.6e-04	2.00	3.5e-05	2.06	3.5e-05	2.00
			$\sqrt{2}$	1.6e-04	2.00	4.9e-06	3.99	1.3e-07	4.00
				$\mathbb{P}^2$					
			0.9	6.9e-05	-	1.4e-05	-	9.4e-06	-
20	9.8e-05	-	1	7.1e-05	-	1.8e-05	-	4.1e-06	-
			$\sqrt{2}$	6.9e-05	-	1.3e-04	-	2.2e-08	-
			0.9	8.6e-06	3.00	1.2e-06	3.54	1.2e-06	3.01
40	1.2e-05	3.00	1	8.6e-06	3.04	6.3e-07	4.87	5.0e-07	3.01
			$\sqrt{2}$	8.6e-06	3.00	2.2e-06	5.92	3.5e-10	6.00
			0.9	1.1e-06	3.00	1.5e-07	3.06	1.5e-07	3.00
80	1.5e-06	3.00	1	1.1e-06	3.00	6.5e-08	3.28	6.3e-08	3.00
			$\sqrt{2}$	1.1e-06	3.00	3.5e-08	5.98	5.4e-12	6.00
				$\mathbb{P}^3$					
			0.9	1.4e-06	-	4.7e-07	-	4.8e-08	-
20	1.9e-06	-	1	1.4e-06	-	1.1e-06	-	1.6e-08	-
			$\sqrt{2}$	1.7e-06	-	1.6e-05	-	8.6e-12	-
			0.9	8.5e-08	4.00	3.7e-09	6.99	2.9e-09	4.03
40	1.2e-07	4.00	1	8.5e-08	4.00	4.5e-09	7.89	9.6e-10	4.03
			$\sqrt{2}$	8.6e-08	4.34	6.9e-08	7.87	4.2e-14	7.69
			0.9	5.3e-09	4.00	1.9e-10	4.31	1.8e-10	4.01
80	7.5e-09	4.00	1	5.3e-09	4.00	6.5e-11	6.11	6.0e-11	4.01
			$\sqrt{2}$	5.3e-09	4.00	2.7e-10	7.97	1.0e-14	2.06
				$\mathbb{P}^4$					
			0.9	2.2e-08	-	2.4e-08	-	1.5e-10	-
20	3.1e-08	-	1	2.2e-08	-	6.7e-08	-	5.4e-11	-
			$\sqrt{2}$	7.1e-08	-	2.0e-06	-	5.4e-14	-
			0.9	6.7e-10	5.00	2.5e-11	9.90	4.6e-12	5.06
40	9.5e-10	5.00	1	6.7e-10	5.01	7.0e-11	9.91	1.7e-12	5.01
			$\sqrt{2}$	6.8e-10	6.70	2.2e-09	9.82	5.5e-14	-0.02
			0.9	2.1e-11	5.00	1.5e-13	7.34	1.5e-13	4.91
80	3.0e-11	5.00	1	2.1e-11	5.00	1.1e-13	9.29	9.4e-14	4.16
			$\sqrt{2}$	2.1e-11	5.01	2.2e-12	9.99	5.5e-14	0.01

Table 4.2: L^2 error and order before and after applying several Line Filters on the L^2 projection of the function $u(x, y) = \sin(x + y)$ on $\Omega = [0, 2\pi]^2$.

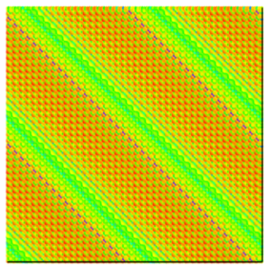
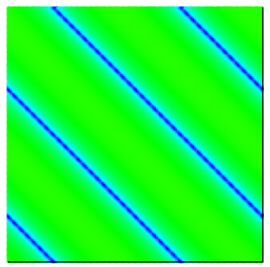
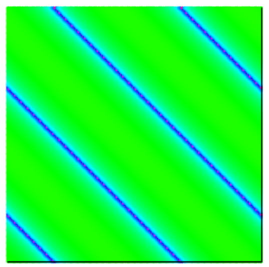
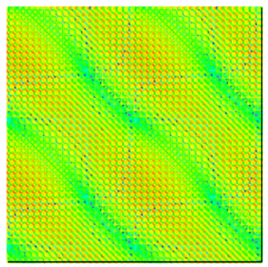
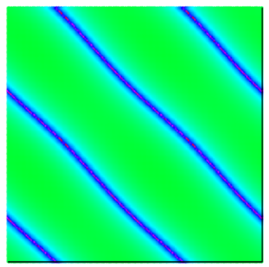
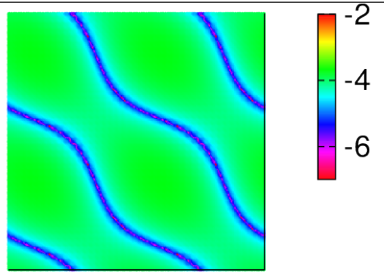
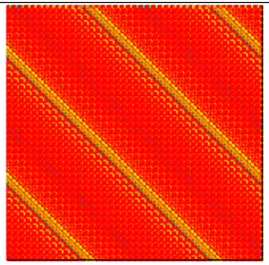
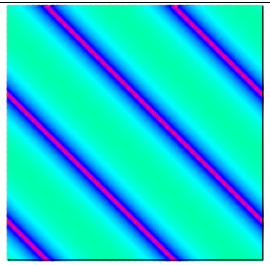
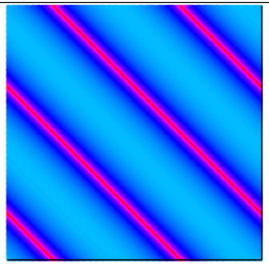
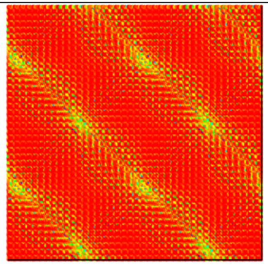
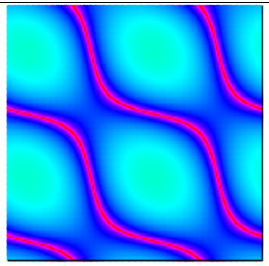
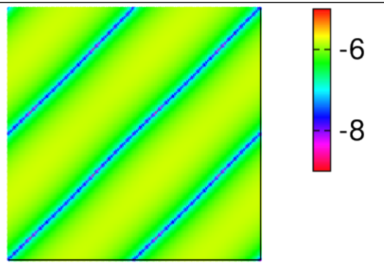
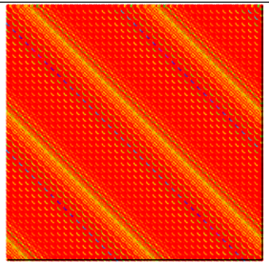
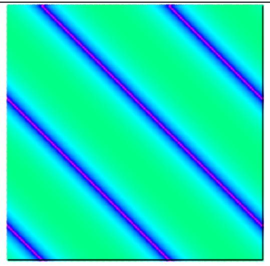
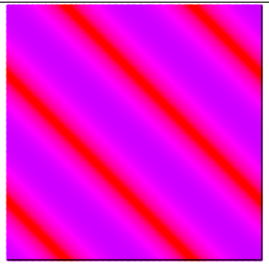
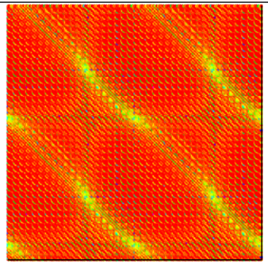
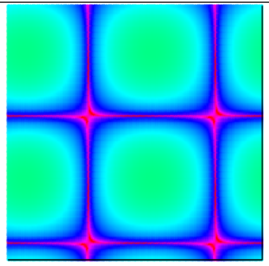
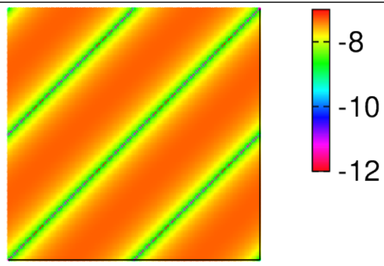
	$u_0(x,y) = \sin(x+y)$			$u_0(x,y) = \sin x \cdot \cos y$		
	DG	**TP** $\theta = 0,\ \mu = 1$	**LF** $\theta = \frac{3\pi}{4},\ \mu = \sqrt{2}$	**DG**	**TP** $\theta = 0,\ \mu = 1$	**LF** $\theta = \frac{3\pi}{4},\ \mu = \sqrt{2}$
$\mathbb{P}^1$						
$\mathbb{P}^2$						
$\mathbb{P}^3$						

Figure 4.6: Contour line error plots (log) before and after applying the Tensor Proudct (TP) filter and the Line Filter (LF) on the DG solution to the problem $u_t + u_x + u_y = 0$ on $\Omega = [0, 2\pi]^2$ with final time $T = 2$ and using 40×40 elements.

N	DG		TPF		LF			
			$\theta = 0,\ \mu = 1$		$\theta = \pi/4,\ \mu = \sqrt{2}$		$\theta = 3\pi/4,\ \mu = \sqrt{2}$	
Initial Condition: $\boldsymbol{u(x,y) = \sin(x+y)}$								
	L^2-Error	Order	L^2-Error	Order	L^2-Error	Order	L^2-Error	Order
$\mathbb{P}^1$								
20	9.7e-03	-	1.6e-03	-	2.7e-03	-	1.5e-03	-
40	2.4e-03	2.02	2.0e-04	3.05	2.6e-04	3.33	1.9e-04	2.98
80	5.9e-04	2.01	NA	NA	2.8e-05	3.21	2.4e-05	2.99
$\mathbb{P}^2$								
20	2.4e-04	-	6.1e-06	-	1.4e-04	-	1.5e-06	-
40	2.9e-05	3.01	1.2-e07	5.71	2.3e-06	5.91	4.7e-08	4.98
80	3.6e-06	3.01	NA	NA	3.7e-08	5.95	1.5e-09	5.00
$\mathbb{P}^3$								
20	4.5e-06	-	1.4e-07	-	1.6e-05	-	7.7e-10	-
40	2.8e-07	4.01	5.6e-10	7.96	6.9e-08	7.87	6.9e-12	6.79
80	1.7e-08	4.00	NA	NA	2.7e-10	7.97	2.9e-14	7.90
Initial Condition: $\boldsymbol{u(x,y) = \sin x \cos y}$								
	L^2-Error	Order	L^2-Error	Order	L^2-Error	Order	L^2-Error	Order
$\mathbb{P}^1$								
20	5.2e-03	-	8.3e-04	-	1.3e-03	-	9.7e-04	-
40	1.3e-03	2.02	1.0e-04	3.05	1.3e-04	3.33	1.0e-04	3.23
80	3.2e-04	2.01	NA	NA	1.4e-05	3.21	1.2e-05	3.08
$\mathbb{P}^2$								
20	1.3e-04	-	3.7e-06	-	6.8e-05	-	6.7e-05	-
40	1.6e-05	3.01	6.8e-08	5.77	1.1e-06	5.91	1.1e-06	5.92
80	2.0e-06	3.00	NA	NA	1.8e-08	5.95	1.8e-08	5.98
$\mathbb{P}^3$								
20	2.4e-06	-	9.8e-08	-	8.1e-06	-	8.1e-06	-
40	1.5e-07	4.01	3.9e-10	7.96	3.4e-08	7.87	3.4e-08	7.87
80	9.5e-09	4.00	NA	NA	1.4e-10	7.97	1.4e-10	7.97

Table 4.3: L^2 errors and orders before and after applying several Line Filters on the DG solution to the problem $u_t + u_x + u_y = 0$ on $\Omega = [0, 2\pi]^2$ with final time $T = 2$ and two different initial conditions.

	Unfiltered		Line Filtered								
			$H = \mu h$	$\theta = \pi/4$		$\theta = 3\pi/4$		$\theta = \arctan(3)$		$\theta = \arctan(3) + \pi/2$	
N	L^2-Error	Order	μ	L^2-Error	Order	L^2-Error	Order	L^2-Error	Order	L^2-Error	Order
				$\mathbb{P}^1$							
20	5.8e-02	-	1	4.6e-02	-	4.1e-02	-	4.7e-02	-	5.2e-02	-
			$\sqrt{2}$	5.6e-02	-	4.1e-02	-	6.4e-02	-	4.7e-02	-
40	1.3e-02	2.15	1	6.2e-03	2.90	5.8e-03	2.84	5.9e-03	2.97	1.1e-02	2.29
			$\sqrt{2}$	6.5e-03	3.09	5.4e-03	2.91	7.3e-03	3.14	8.5e-03	2.46
80	3.2e-03	2.04	1	8.7e-04	2.83	8.3e-04	2.80	7.6e-04	2.96	2.4e-03	2.12
			$\sqrt{2}$	7.6e-04	3.12	6.9e-04	2.98	8.4e-04	3.11	1.8e-03	2.23
				$\mathbb{P}^2$							
20	3.6-e03	-	1	1.5e-03	-	5.2e-04	-	2.3e-03	-	2.5e-03	-
			$\sqrt{2}$	7.3e-03	-	5.0e-04	-	1.3e-02	-	1.8e-03	-
40	4.6e-04	2.95	1	5.8e-05	4.71	3.7e-05	3.80	4.8e-05	5.56	3.5e-04	2.86
			$\sqrt{2}$	1.5e-04	5.63	1.4e-05	5.15	2.7e-04	5.57	2.4e-04	2.87
80	5.9e-05	2.97	1	4.7e-06	3.63	4.0e-06	3.23	1.2e-06	5.30	4.6e-05	2.94
			$\sqrt{2}$	2.6e-06	5.82	4.1e-07	5.12	4.9e-06	5.78	3.2e-05	2.94
				$\mathbb{P}^3$							
20	2.1e-04	-	1	7.7e-04	-	4.2e-04	-	5.3e-04	-	2.6e-03	-
			$\sqrt{2}$	3.3e-03	-	3.9e-04	-	6.3e-03	-	1.7e-03	-
40	1.3e-05	3.95	1	1.2e-06	9.36	2.7e-07	12.60	3.8e-05	3.79	6.4e-06	8.67
			$\sqrt{2}$	1.6e-05	7.66	8.2e-08	12.19	3.2e-05	7.61	4.1e-06	8.69
80	8.5e-07	3.98	1	1.9e-08	5.95	1.5e-08	4.21	1.2e-08	11.64	4.1e-07	3.95
			$\sqrt{2}$	6.9e-08	7.87	3.9e-12	7.72	1.7e-07	7.60	2.7e-07	3.95

Table 4.4: L^2 errors and orders before and after applying several Line Filters on the DG solution to the problem $u_t + u_x + u_y = 0$ on $\Omega = [0, 2\pi]^2$ with initial condition $u(x, y) = \sin(x + 3y)$ and final time $T = 2$.

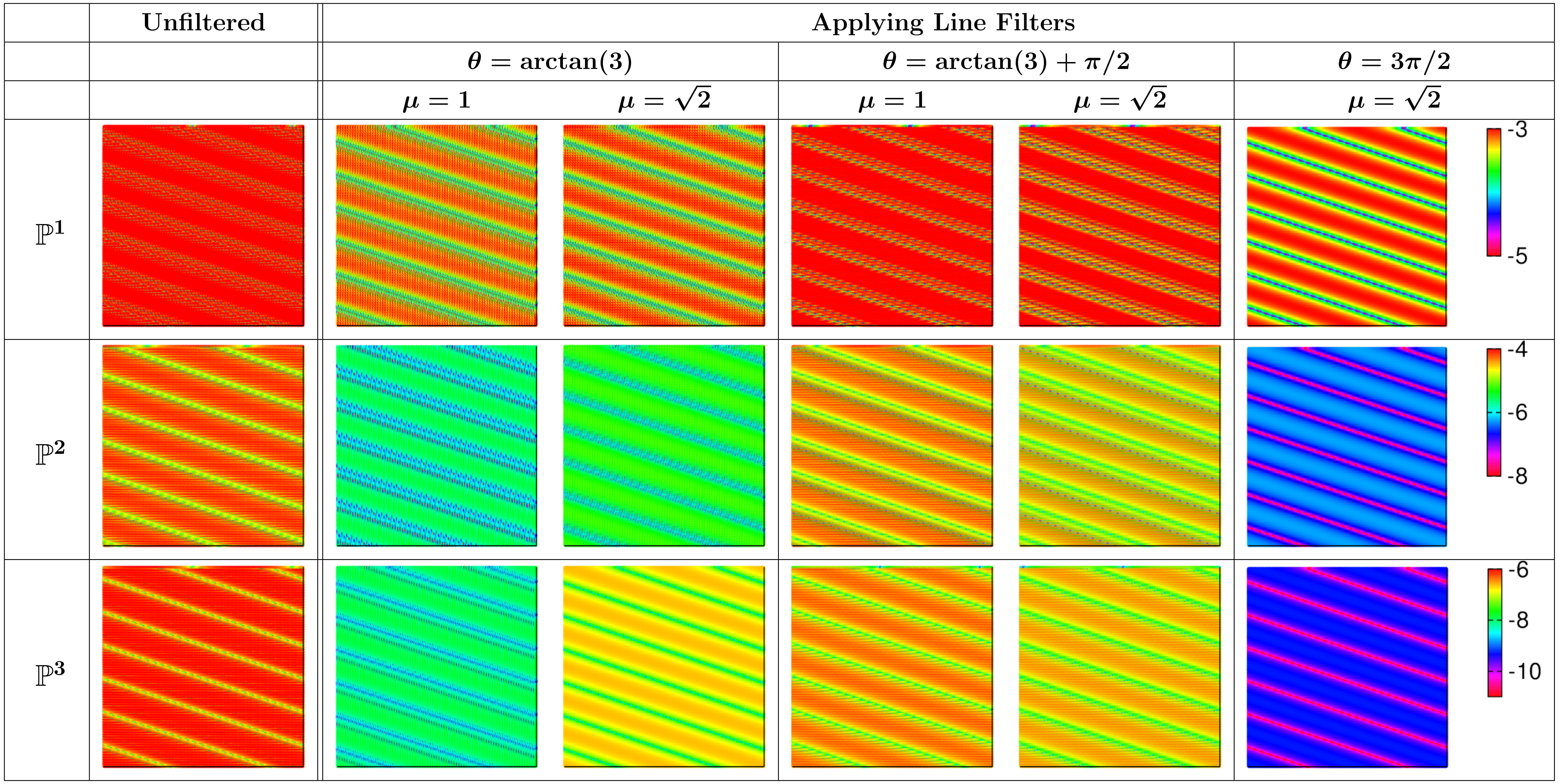


Figure 4.7: Contour line error plots (log) before and after applying Line Filters on the DG solution to the problem $u_t + u_x + u_y = 0$ with initial condition $u(x, y) = \sin(x + 3y)$ on $\Omega = [0, 2\pi]^2$ with final time $T = 2$ and using 80×80 elements.

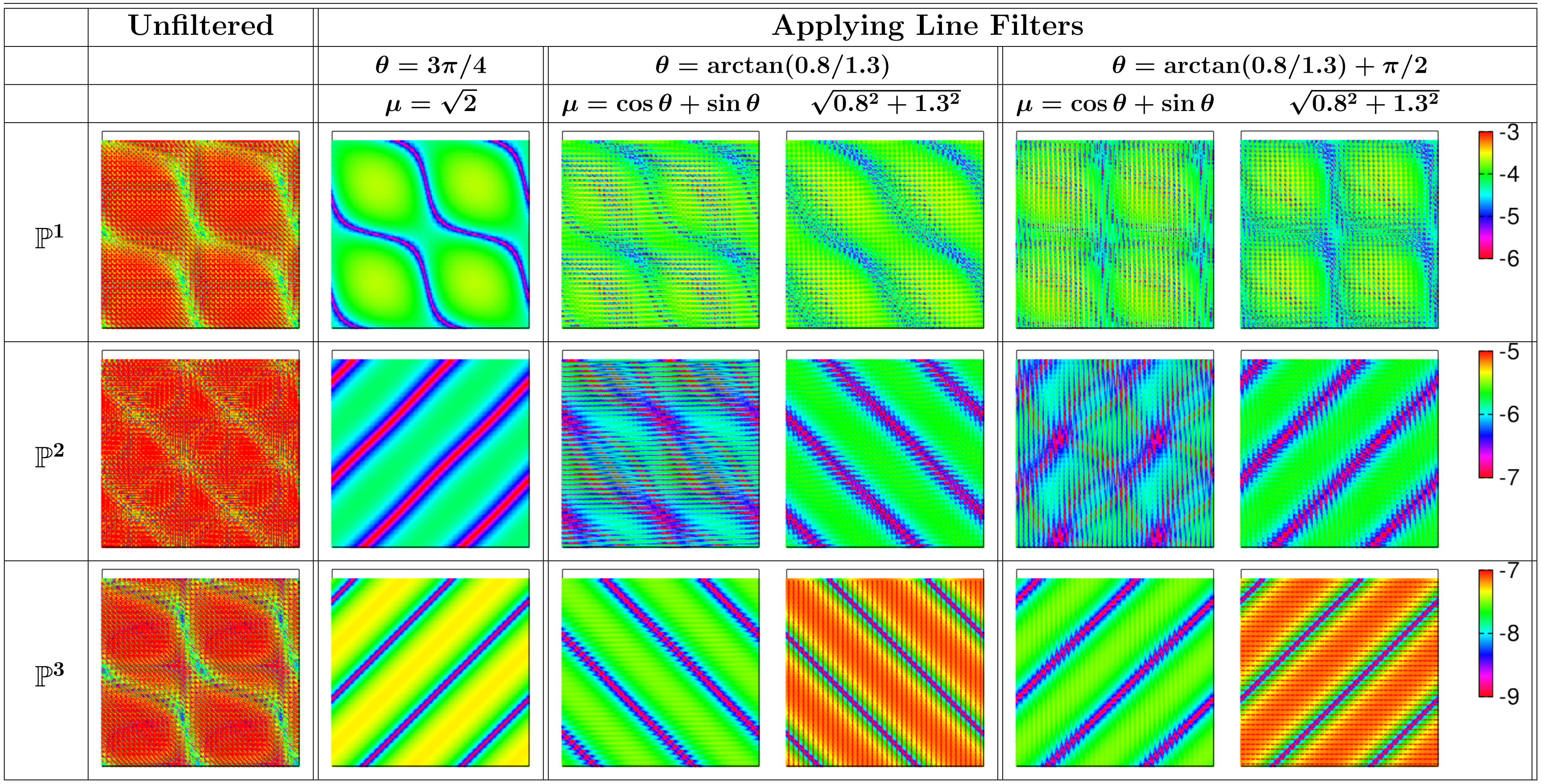


Figure 4.8: Contour line error plots (log) before and after applying different line filters oriented using the mesh and flow direction respectively for the DG solution to the problem $u_t + 1.3u_x + 0.8u_y = 0$ on $\Omega = [0, 2\pi]^2$ with initial condition $u_0(x, y) = \cos x \sin y$ and final time $T = 2$ and using 40×40 elements.

	Unfiltered		**Applying Line Filters**								
			$\boldsymbol{\theta = \pi/4,\ \mu = \sqrt{2}}$		$\boldsymbol{\theta = 3\pi/4,\ \mu = \sqrt{2}}$		$\boldsymbol{H = \mu h}$	$\boldsymbol{\theta = \arctan(0.8/1.3)}$		$\boldsymbol{\theta = \arctan(0.8/1.3) + \pi/2}$	
N	L^2-Error	Order	L^2-Error	Order	L^2-Error	Order	μ	L^2-Error	Order	L^2-Error	Order
							$\mathbb{P}^1$				
20	5.5e-03	-	1.1e-03	-	1.4e-03	-	1	1.8e-03	-	1.9e-03	-
							$\sin(\theta) + \cos(\theta)$	1.4e-03	-	1.2e-03	-
							$\sqrt{0.8^2 + 1.3^2}$	1.6e-03	-	1.2e-03	-
40	1.4e-03	2.02	1.2e-04	3.20	1.4e-04	3.31	1	3.9e-04	2.21	4.1e-04	2.19
							$\sin\theta + \cos\theta$	1.8e-04	2.98	1.8e-04	2.77
							$\sqrt{0.8^2 + 1.3^2}$	1.6e-04	3.25	1.4e-04	3.10
80	3.4e-04	2.01	1.6e-05	3.19	1.4e-05	3.08	1	9.4e-05	2.04	9.8e-05	2.08
							$\sin\theta + \cos\theta$	3.2e-05	2.48	3.4e-05	2.39
							$\sqrt{0.8^2 + 1.3^2}$	2.2e-05	2.87	2.2e-05	2.69
							$\mathbb{P}^2$				
20	1.3e-04	-	6.8e-05	-	6.7e-05	-	1	3.5e-05	-	4.0e-05	-
							$\sin\theta + \cos\theta$	5.0e-05	-	5.0e-05	-
							$\sqrt{0.8^2 + 1.3^2}$	9.0e-05	-	9.0e-05	-
40	1.7e-05	3.01	1.1e-06	5.91	1.1e-06	5.92	1	4.1e-06	3.09	4.5e-06	3.14
							$\sin\theta + \cos\theta$	1.1e-06	5.55	1.1e-06	5.49
							$\sqrt{0.8^2 + 1.3^2}$	1.5e-06	5.88	1.5e-06	5.88
80	2.1e-06	3.00	1.8e-08	5.95	1.8e-08	5.97	1	5.2e-07	3.00	5.4e-07	3.06
							$\sin\theta + \cos\theta$	7.7e-08	3.80	8.4e-08	3.72
							$\sqrt{0.8^2 + 1.3^2}$	3.8e-08	5.35	3.9e-08	5.29
							$\mathbb{P}^3$				
20	2.6e-06	-	8.1e-06	-	8.1e-06	-	1	5.7e-07	-	5.7e-07	-
							$\sin\theta + \cos\theta$	5.3e-06	-	5.3e-06	-
							$\sqrt{0.8^2 + 1.3^2}$	5.7e-04	-	5.4e-04	-
40	1.6e-07	4.00	3.4e-08	7.87	3.4e-8	7.87	1	2.1e-08	4.76	2.2e-08	4.71
							$\sin\theta + \cos\theta$	2.2e-08	7.88	2.2e-08	7.88
							$\sqrt{0.8^2 + 1.3^2}$	5.5e-04	0.05	5.4e-04	-0.01
80	1.0e-08	4.00	1.4e-10	7.97	1.4e-10	7.97	1	1.3e-09	4.04	1.3e-09	4.06
							$\sin\theta + \cos\theta$	1.3e-10	7.47	1.3e-10	7.43
							$\sqrt{0.8^2 + 1.3^2}$	5.4e-04	0.02	5.4e-04	0.00

Table 4.5: L^2-errors and orders before and after post-processing taking the $3\pi/4$ direction and two additional orientations corresponding to the fl

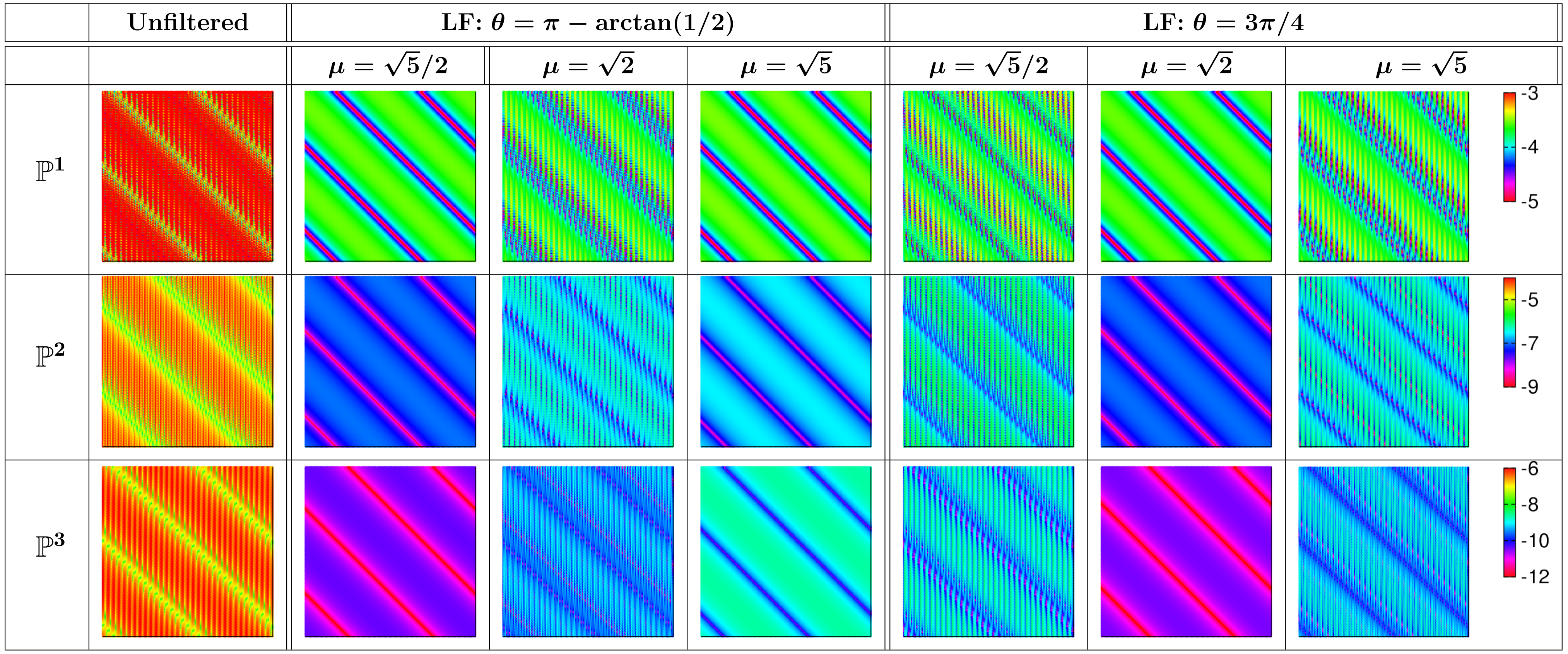


Figure 4.9: Contour line error plots (log) before and after applying Line Filters on the DG solution to the problem $u_t + u_x + u_y = 0$ with initial condition $u(x, y) = \sin(x + y)$ on $\Omega = [0, 2\pi]^2$ with final time $T = 2$ and using a rectangular mesh made of 32×64 elements.

	Unfiltered		Line Filtered								
			$H = \mu h$	$\theta = \pi/4$		$\theta = 3\pi/4$		$\theta = \arctan(1/2)$		$\theta = \pi - \arctan(1/2)$	
N	L^2-Error	Order	μ	L^2-Error	Order	L^2-Error	Order	L^2-Error	Order	L^2-Error	Order
			$\mathbb{P}^1$								
			$\sqrt{5}/2$	2.9e-03	-	1.7e-03	-	2.5e-03	-	**1.7e-03**	**-**
16×32	1.e-02	-	$\sqrt{2}$	4.4e-03	-	**1.6e-03**	**-**	3.9e-03	-	1.7e-03	-
			$\sqrt{5}$	1.8e-02	-	1.7e-03	-	1.5e-02	-	1.8e-03	-
			$\sqrt{5}/2$	3.2e-04	3.17	2.6e-04	2.76	2.6e-04	3.26	**2.1e-04**	**2.98**
32×64	2.7e-03	2.02	$\sqrt{2}$	3.9e-04	3.51	**2.1e-04**	**2.98**	3.7e-04	3.42	2.3e-04	2.89
			$\sqrt{5}$	1.3e-03	3.74	2.3e-04	2.86	1.1e-03	3.74	2.2e-04	3.05
			$\sqrt{5}/2$	4.8e-05	2.75	4.6e-05	2.49	4.8e-05	2.47	**2.6e-05**	**2.99**
64×128	6.7e-04	2.01	$\sqrt{2}$	3.7e-05	3.37	**2.6e-05**	**2.99**	4.3e-05	3.10	3.5e-05	2.70
			$\sqrt{5}$	1.0e-04	3.73	3.7e-05	2.66	8.4e-05	3.73	2.7e-05	3.03
			$\mathbb{P}^2$								
			$\sqrt{5}/2$	1.3e-04	-	6.5e-06	-	9.5e-05	-	**2.5e-06**	**-**
16×32	3.3e-04	-	$\sqrt{2}$	5.0e-04	-	**2.3e-06**	**-**	3.7e-04	-	4.2e-06	-
			$\sqrt{5}$	6.5e-03	-	4.5e-06	-	4.9e-03	-	1.1e-05	-
			$\sqrt{5}/2$	2.4e-06	5.76	7.6e-07	3.11	1.6e-06	5.90	**7.6e-08**	**5.02**
32×64	4.1e-05	3.01	$\sqrt{2}$	8.5e-06	5.87	**7.4e-08**	**4.99**	6.2e-06	5.88	3.7e-07	3.52
			$\sqrt{5}$	1.3e-04	5.69	4.8e-07	3.22	9.2e-05	5.72	2.1e-07	5.70
			$\sqrt{5}/2$	1.0e-07	4.51	9.4e-08	3.02	2.6e-08	5.92	**2.4e-09**	**5.01**
64×128	5.1e-06	3.01	$\sqrt{2}$	1.4e-07	5.96	**2.3e-09**	**4.99**	1.1e-07	5.82	4.4e-08	3.08
			$\sqrt{5}$	2.1e-06	5.92	5.9e-08	3.03	1.5e-06	5.93	4.4e-09	5.56
			$\mathbb{P}^3$								
			$\sqrt{5}/2$	1.5e-05	-	2.4e-08	-	9.8e-06	-	**3.4e-09**	**-**
16×32	7.8e-06	-	$\sqrt{2}$	9.0e-05	-	**1.8e-09**	**-**	6.0e-05	-	1.6e-08	- -
			$\sqrt{5}$	2.7e-03	-	1.5e-08	-	NA	-	NA	-
			$\sqrt{5}/2$	6.3e-08	7.87	1.5e-09	4.02	4.1e-08	7.89	**2.8e-11**	**6.94**
32×64	4.8e-07	4.01	$\sqrt{2}$	4.0e-07	7.80	**2.3e-11**	**6.2**	2.7e-07	7.82	5.7e-10	4.86
			$\sqrt{5}$	1.5e-05	7.50	9.2e-10	4.05	9.8e-06	NA	1.6e-09	-
			$\sqrt{5}/2$	2.7e-10	7.85	9.3e-11	3.99	1.6e-10	7.97	**1.5e-11**	**0.91**
64×128	3.0e-08	4.00	$\sqrt{2}$	1.6e-09	7.95	**1.5e-11**	**0.67**	2.7e-10	9.97	3.6e-11	3.97
			$\sqrt{5}$	6.3e-08	7.88	5.9e-11	3.97	4.1e-08	7.89	1.6e-11	6.67

4.5 Discussion

This chapter presented a solid theoretical background for the SIAC Line filters. Theorem 4.3.1 shows how these filters can attain the expected $2k + 1$ superconvergence of SIAC filters and avoid tensor products. The numerical experiments supported the theory and revealed that generally, the filtered solution has a lower error. From the numerical results, it was concluded that for quadrilateral based meshes, the optimal orientation seems to be the $3\pi/4$ direction. This was tested subject to changes in the initial condition, flow direction and mesh variations. However, the study was limited to linear hyperbolic problems and uniform meshes. Furthermore, the filter performance should be tested for general meshes. Line filtering is very flexible and should allow for a variety of mesh shapes which are currently employed in engineering problems, including unstructured triangular meshes as well as curvilinear elements.

The formulation of the Line filters presented here has only considered symmetric kernels. The next step for these filters should be to introduce one-sided kernels for application near domain boundaries. The existing theoretical error estimates for traditional one-sided filtering should easily extend to Line filtering similar to the symmetric case. Hence, future work on these filters should explore such alternative kernels, allowing more robust applications.

Chapter 5

Computing SIAC Filters

There are two main tasks involved during the implementation of SIAC filters, namely building the kernel itself and computing the filtering convolution. Since the values of the kernel coefficients (c_γ in equation (2.18) or (2.28)) are not given explicitly, the first step towards the implementation of SIAC filters is to compute such coefficients. The second and most important task is to actually find the numerical solution to the integral (*e.g.*, equation (3.9)) which involves several stages.

The kernel coefficients assign weights to each B-Spline through the filter polynomial reproduction property (equation (2.26)) and can be determined by solving a linear system of the type $Ac = b$. However, as the polynomial degree increases, matrix A has a large condition number. This usually implies that round-off errors will dominate and produce a singular system so the inverse no longer exists and it is not possible to find the solution. In [43], an alternative method was developed where such coefficients were characterized using Fourier analysis and avoided solving the linear system. This idea was later exploited in [47] and aided in deriving explicit formulas to calculate kernel coefficients using B-Splines with uniform and non-uniform knots [45]. For the purposes of this research, solving the linear system remains a safe approach since the numerical experiments use relatively low polynomial degrees. Hence, it will be shown how to create the matrix A for the system.

Despite the fact that calculating the integral of the convolution might seem straight forward, there are several computational challenges involved; to effectively solve the convolution, one has to find all the discontinuities within the integration region, use them to split the integral and finally apply a numerical method to approximate the integral at each piece. The authors in [40] provide algorithms to implement SIAC filters in multi-dimensions for triangular and quadrilateral meshes. However, they do not include the process of finding these discontinuities. Furthermore, they point out how this task becomes difficult for general non-uniform meshes. Later, it will be shown how the rotated kernel footprint on the DG mesh together with all its breaks results in a partition of the integral with a random structure (see Figure 5.9). Thus, the work carried out in this thesis made it necessary to develop an algorithm capable of searching for discontinuities without any assumptions on the underlying mesh. In addition, since the number of kernel breaks per element can be relatively high, a technique is presented to perform an appropriate polygon splitting in order to produce regions suitable for numerical integration. As a result, the implementation presented here is robust, admits either triangular or quadrilateral elements and does not rely on structured meshes.

This Chapter not only discusses the computational issues mentioned above but also explores the number of operations and simulation times required to post-process one point. By performing such a study, it is easy to extrapolate the cost of globally post-processing every point in the domain. The difference between the number of integration regions for quadrilateral and triangular meshes will be explored as well as what happens when the filters are applied to non-uniform meshes. This has direct impact on the performance of tensor product filters, requiring long simulation times, especially for triangular meshes. Line filters on the other hand, are minimally affected by the mesh properties so from a computational point of view, they have promising applications to unstructured triangular meshes.

5.1 Finding the Kernel Coefficients

In Chapter 2, the general SIAC kernel was defined by the formula

$$K^{(r+1,\ell)}(\eta) = \sum_{\gamma=0}^{r} c_\gamma \psi^{(\ell)}(\eta - x_\gamma(\lambda)) + c_{r+1} b^{(\ell)}(\underbrace{x - H\eta}_{=y}), \quad x_\gamma(\lambda) = -\frac{r}{2} + \gamma + \lambda, \quad (5.1)$$

Imposing $\lambda = 0$ in this equation and removing general B-Spline $b^{(\ell)}(\cdot)$ gives the original symmetric SIAC filter. Here, the computation of the kernel coefficients is discussed for the general kernel.

The polynomial reproduction property

$$K \star x^p = z^p, \; p = 0, \ldots, r,$$

implies

$$\sum_{\gamma=0}^{r} c_\gamma \underbrace{\int_{-\infty}^{\infty} \psi^{(\ell)}(x - y - x_\gamma(\eta)) y^p dy}_{=A(p,\gamma)} + c_{r+1} \underbrace{\int_{-\infty}^{\infty} b^{(\ell)}(x-y) y^p dy}_{=A(p,r+1)} = x^p, \; p = 0 \ldots, r+1.$$

One can obtain the coefficients c_γ's by solving the linear system

$$\begin{pmatrix} A(0,0) & \ldots & A(0,r+1) \\ \vdots & \ddots & \vdots \\ A(r+1,0) & \ldots & A(r+1,r+1) \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ c_{r+1} \end{pmatrix} = \begin{pmatrix} x^0 \\ \vdots \\ x^{r+1} \end{pmatrix} \quad (5.2)$$

Note 5.1.1. *Introducing a general spline adds an equation to the system. Hence, in order to be able to determine the coefficients, the filter is required to reproduce polynomials up to degree $r+1$ (rather than r). When using kernels $K^{(r,k+1)}$ without an additional general spline, it suffices to impose $p = 0, \ldots, r$.*

The entries of the matrix A can be computed using Gauss integration. Alternatively, one can solve these integrals analytically as it will be shown next [54].

Begin with the terms corresponding to the central B-Splines:

$$A(p,\gamma) = \int_{-\infty}^{\infty} \psi^{(\ell)}(z-y-x_\gamma(\eta))y^p dy, \quad \text{let } t = z-y-x_\gamma(\eta) \Rightarrow \begin{cases} y = z-t-x_\gamma(\eta) \\ dy = -dt \end{cases} \Rightarrow$$

$$A(p,\gamma) = -\int_{\infty}^{-\infty} \psi^{(\ell)}(t)(z-t-x_\gamma(\eta))^p dt.$$

Since the filter must reproduce **all** polynomials up to degree $r+1$ for any point z, take $z=0$. Then,

$$A(p,\gamma) = -\int_{\infty}^{-\infty} \psi^{(\ell)}(t)(-t-x_\gamma(\eta))^p dt.$$

Using the Binomial Theorem,

$$(-t-x_\gamma)^p = (-1)^p(t+x_\gamma(\eta)) = (-1)^p \sum_{i=0}^{p} \binom{p}{i} x_\gamma(\eta)^{(p-i)} t^i,$$

it is possible to write

$$A(p,\gamma) = (-1)^p \sum_{i=0}^{p} \binom{p}{i} x_\gamma(\eta)^{(p-i)} \int_{-\infty}^{\infty} \psi^{(\ell)}(t) t^i dt.$$

Integrating by parts and imposing the relation between the derivatives of the B-Splines and the divided differences:

$$\frac{d^{\ell-1}}{dt^{\ell-1}} \psi^{(\ell)}(t) = \partial^{\ell-1}\psi^{(1)}(t), \tag{5.3}$$

gives

$$A(p,\gamma) = (-1)^p \sum_{i=0}^{p} \binom{p}{i} x_\gamma(\eta)^{(p-i)} \frac{(-1)^{\ell-1}}{(i+1)\cdots(i+\ell)} \int_{-\infty}^{\infty} \eta^{i+(\ell-1)} \partial^{\ell-1}\psi^{(1)}(t) dt. \tag{5.4}$$

The integral term in equation (5.4) can be solved using the following formula for the divided differences:

$$\partial^{\ell-1}\psi^{(1)}(t) = \sum_{j=0}^{\ell-1} \binom{\ell-1}{j} (-1)^j \psi^{(1)}\left(t + \left(\frac{\ell-1}{2} - j\right)\right). \tag{5.5}$$

Since $\psi^{(1)}(t) = \chi_{[-1/2,1/2)}(t)$,

$$\int_{-\infty}^{\infty} t^{i+(\ell-1)} \partial^{\ell-1}\psi^{(1)}(t) dt = \sum_{j=0}^{\ell-1} \binom{\ell-1}{j} (-1)^j \left[\left(j - \frac{(\ell-1)-1}{2}\right)^{\ell+i} - \left(j - \frac{\ell}{2}\right)^{\ell+i} \right].$$

The matrix coefficients

$$A(p,r+1) = \int_{-\infty}^{\infty} b^{(\ell)}(z-y) y^p dy,$$

corresponding to the general spline are obtained in a similar way. Consider a left-sided filter, *i.e.* $\lambda = -\frac{r+\ell}{2} + \frac{\overline{x}-x_L}{H}$ with $\overline{x}$ being the post-processing point and x_L the domain left boundary. Let

$$t = z - y \Rightarrow \begin{cases} y = z - t \\ dy = -dt \end{cases} \Rightarrow A(p, r+1) = -\int_{\infty}^{-\infty} b^{(\ell)}(t)(z-t)^p dt$$
$$= \int_{\frac{\overline{x}-x_L}{h}-1}^{\frac{\overline{x}-x_l}{h}} \left(t - \left(\frac{\overline{x}-x_L}{h} - 1\right)\right)^k (-t)^p dt.$$

Imposing $z = 0$ and applying the binomial theorem:

$$A(p, r+1) = \sum_{i=0}^{\ell-1} \binom{\ell-1}{i} (-1)^{\ell-1+i} \left(\frac{\overline{x}-x_L}{h} - 1\right)^{\ell-1-i} \int_{\frac{\overline{x}-x_L}{h}-1}^{\frac{\overline{x}-x_L}{h}} t^i (-t)^p dt$$
$$= \sum_{i=0}^{k} \binom{\ell-1}{i} (-1)^{\ell-1-i+p} \left(\frac{\overline{x}-x_L}{h} - 1\right)^{\ell-1-i} \left(\frac{\frac{\overline{x}-x_L}{h}^{p+i+1} - (\frac{\overline{x}-x_l}{h} - 1)^{p+i+1}}{p+i+1}\right).$$

For the right case, it is easy to check that coefficient for c_{r+1} is given by:

$$A(p, r+1) = \sum_{i=0}^{k} \binom{k}{i} (-1)^{i+p} \left(\frac{\overline{x}-x_R}{h} + 1\right)^{k-i} \left(\frac{\left(\frac{\overline{x}-x_R}{h} + 1\right)^{p+i+1} - \left(\frac{\overline{x}-x_R}{h}\right)^{p+i+1}}{p+i+1}\right).$$

Note 5.1.2. *The kernel coefficients for the tensor product filters are computed individually along each direction.*

5.2 Implementation of SIAC Filters

As it was mentioned earlier, computing the filtering convolution requires several steps. In short, to successfully solve the integral, it is necessary to split the kernel support into integrable regions and then transform such regions into standard elements where numerical integration takes place. Here, two algorithms are given which were developed to solve equation (3.9) numerically. The first one consists of a routine that finds all the intersections between two overlapping meshes. This allows for determining the filter footprint in the mesh and provides the integral limits. The second algorithm is related to numerical integration over arbitrary regions using Gauss Quadrature rules.

5.2.1 The Intersection Algorithm

Since the filter has compact support, the integral in equation (3.9) is non-zero only in a small part of the DG mesh. However, the integration region contains two kinds of discontinuities, delimited by the mesh element boundaries and the kernel breaks; the natural discontinuous structure of DG produces a solution that is integrable only inside the elements. Furthermore, the kernel is built as a linear combination of B-Splines of degree k which means that it has only $k-1$ smoothness for each Spline. The kernel support is split into *kernel boxes* delimited by the kernel breaks which are given by the B-Splines knots. Just like DG for the mesh elements, the kernel function is only

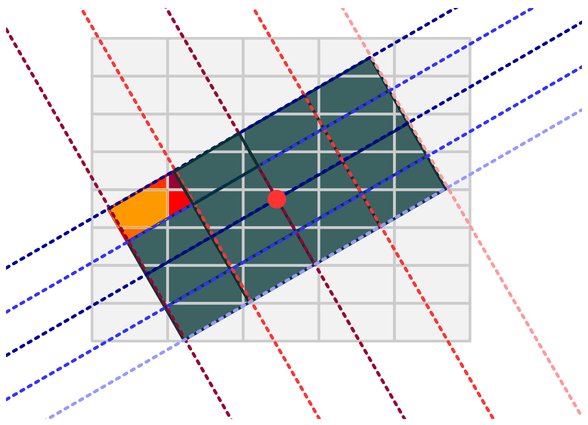


Figure 5.1: Footprint of a rotated filter using three B-Splines of order two in each direction. The grey quadrilaterals are the mesh elements. Dashed lines denote the kernel breaks and the rotated rectangles denote the “kernel boxes”. The multicoloured box illustrates the partition of the kernel box into integrable regions and the red circle denotes the post-processing point.

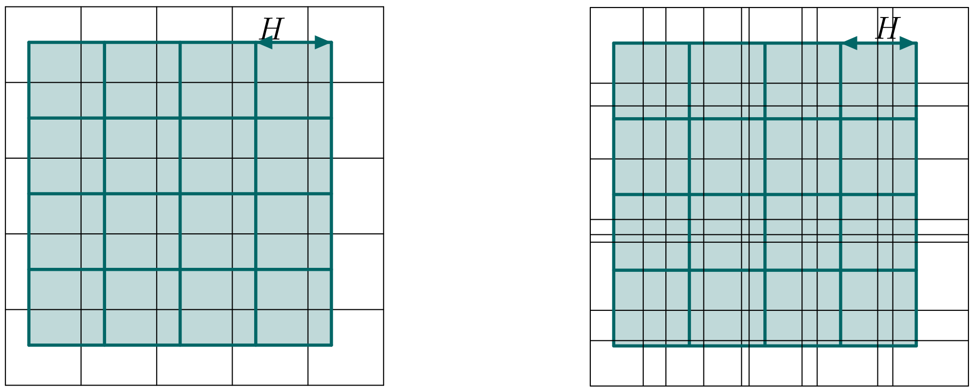


Figure 5.2: Footprint of an axis aligned filter highlighting the integration regions when using a uniform (left) and nonuniform (right) mesh.

integrable at the interior of the boxes. This is illustrated in Figure 5.1, where the footprint of a rotated filter is shown together with the element boundaries and the kernel boxes.

The footprint of a Cartesian axis filter over a uniform mesh can be predicted by taking advantage of the regular structures. However, this is not the case for non uniform meshes as show in Figure 5.2. Introducing a filter rotation has a similar effect. One of the most challenging and intense computations is to actually find and sort these regions. The kernel boxes together with the DG element boundaries produces a “random” structure as shown in Figure 5.1. Hence, for both a rotated filter and for an axis aligned filter over nonuniform meshes, one needs a tool that finds all the kernel-mesh intersections in a systematic manner. In the following, a rotated filter will be considered. The Cartesian axis aligned filter implementation can be identified with a rotated filter with angle $\theta = 0$.

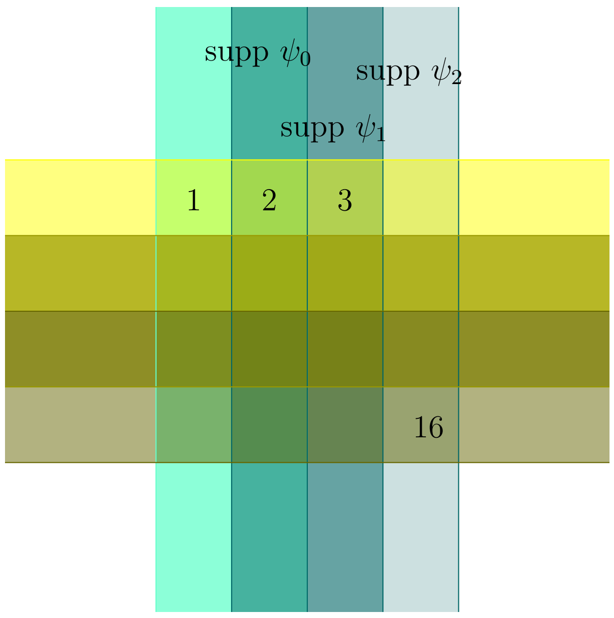


Figure 5.3: Example of a kernel consisting of three B-Splines of order two in each direction, producing 16 kernel boxes. Each vertical and horizontal line correspond to a B-Spline knot and the stripes represent the B-Spline support.

5.2.2 The Kernel Integral Delimiters

Define a rotated tensor product filter:

$$K_H^{(r+1,\ell)}(x', y') = K_{H_x}^{(r+1,\ell)}(\overline{x} - x') \otimes K_{H_y}^{(r+1,\ell)}(\overline{y} - y'), \tag{5.6}$$

where each kernel is symmetric. Recall that the central B-Spline of order ℓ has the knot sequence:

$$\boldsymbol{t} = t_i = -\frac{\ell}{2}, \frac{\ell-2}{2}, \dots, \frac{\ell}{2}. \tag{5.7}$$

The kernel breaks are determined by these knot sequences centred at each B-Spline node $x_\gamma = -\frac{r}{2} + \gamma,\ \gamma = 0, \dots r$:

$$b_{\gamma,\boldsymbol{t}} = x_\gamma + \boldsymbol{t}. \tag{5.8}$$

Finally, the kernel boxes are built using ordered kernel breaks as vertices:

$$V_{ij} = (b_{\gamma,t_i}, b_{\gamma',t_j}). \tag{5.9}$$

Example 5.2.1. *Let $k = 1$ ($r = 2k = 2,\ \ell = k + 1 = 2$), then the symmetric kernel $K_H^{(3,2)}(x', y')$ with B-Splines nodes $x_\gamma = -1, 0, 1$ has five kernel breaks in each direction:*

$$b = -2, -1, 0, 1, 2 \qquad (\boldsymbol{t} = -1, 0, 1)$$

producing the kernel boxes shown in Figure 5.3.

Note 5.2.1. *The general SIAC kernel with the parameter λ has the same kernel boxes but shifted towards one direction. Therefore, for simplicity, the symmetric kernel is used for the discussion.*

In order to locate the filter footprint, the kernel boxes are projected onto the DG mesh through the kernel vertices. Write

$$\eta_x = \frac{\overline{x} - x'}{H_x} \Rightarrow x' = \overline{x} - \eta_x H_x \qquad (y' = \overline{y} - \eta_y H_y)\,. \tag{5.10}$$

In the kernel basis $\mathcal{B}_2 = \{k_x, k_y\}$, the vertices have coordinates:

$$\begin{pmatrix} x'_i \\ y'_j \end{pmatrix} = \begin{pmatrix} \overline{x} - b_{\gamma,t_i} H_x \\ \overline{y} - b_{\gamma',t_j} H_y \end{pmatrix}. \tag{5.11}$$

Finally, the change-of-basis matrix (see (3.6)) gives the coordinates of the kernel vertices in the DG frame of reference:

$$\begin{pmatrix} x_i \\ y_j \end{pmatrix} = \frac{1}{det(\mathcal{P}_{\mathcal{B}_2 \leftarrow \mathcal{B}_1})} \begin{pmatrix} \sin\theta_y & -\cos\theta_y \\ -\sin\theta_x & \cos\theta_x \end{pmatrix} \begin{pmatrix} x'_i \\ y'_j \end{pmatrix}. \tag{5.12}$$

5.2.3 Finding all the Integral Regions

There are three types of points defining the integrable regions of the convolution in equation (5.6). The first type, the kernel boxes, dictate the main blocks in which the integral is split. Then, for each box, one has to find all the discontinuities arising from the DG mesh itself. These are collected in two groups: element vertices and element edge intersection points. Algorithm 1[1] describes how to find all the element vertices belonging to a particular kernel box. There is an important function call inside the

Algorithm 1 Collect Mesh Vertices

```
for $i = 0 : 3$ do
    $kv(i) \leftarrow$ kernel vertex
    $ID(i) \leftarrow$ get element id$(kv(i))$
end for
$B = \overline{\cup kv(i)}$   # bounding box
$(\min, \max) \leftarrow$ get min max$(ID)$
for $id \leftarrow \min : \max$ do
    while $nv <$ number of vertices in $(id)$ do
        if $point\ in\ polygon(vertex(nv), B)$ then
            collect coordinates and vertex id map
        else
            $++nv$
        end if
    end while
end for
```

algorithm: *point in polygon()*. This function solves the following problem: given a

[1]This routine assumes a mesh with ordered elements. In this case, the ID map is similar to the Nektar++ [7] structure, labelling from bottom to top and from left to right.

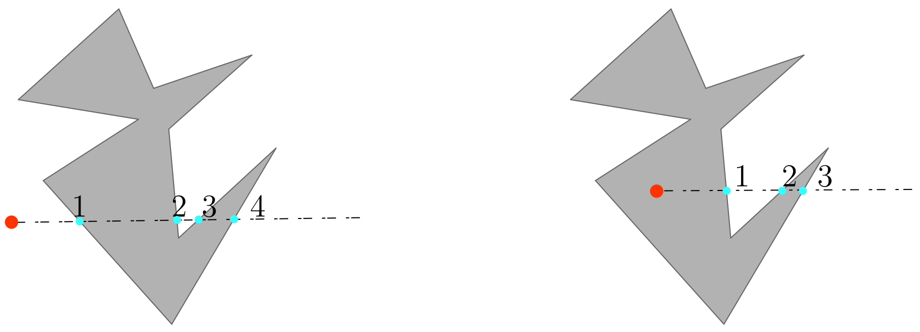


Figure 5.4: The Ray Casting Algorithm *even-odd* rule. The ray in the left figure crosses the polygon four times (even case) so the test point (red dot) is outside the polygon. The right figure shows a point inside the polygon and the ray crosses the polygon three times (odd case).

point and a polygon, determine whether the point lives inside or outside the polygon. Although the question is trivial for the human eye, the answer is not so immediate for a computer. The problem belongs to the branch of mathematics known as *computational geometry* and there is a long history on the development of algorithms for the point in polygon test [8, 25, 58]. Here a particular technique is briefly discussed: the *ray-casting algorithm*. More details can be found in [46, Ch. 7] and the code (in C) can be downloaded from [16].

The Ray Casting Approach

1. Check if the y-coordinate of the test point is between the polygon minimum and maximum y-value.
2. Draw a "semi-infinite" horizontal line from the test point.
3. Count the number of times the line crosses with the polygon. Each time the line intersects the polygon, the ray switches between the inside and outside regions as shown in Figure 5.4. If there are an odd number of switches, then the point lives inside the polygon. Otherwise, it is outside the polygon.

The proof of this algorithm can be done using a result from algebraic topology.

Definition 5.2.1. *A Jordan curve is a plane curve homeomorphic to the unit circle.*

Theorem 5.2.1. *(Jordan Curve Theorem [24]). Let S^n be the n^{th} dimensional sphere. A subspace of S^2 homeomorphic to S^1 separates S^2 into two complementary components.*

Note 5.2.2. *Identifying the Jordan curve with the polygon, this result can be interpreted as: "a simple closed curve divides the plane into an interior and exterior region".*

Finally, the last type of discontinuity is discussed: the element edge intersections. The idea is to loop around the kernel box and collect all the points where the trace crosses an element edge. The routine is described in Algorithm 2. The *get intersection*

Algorithm 2 Collect Element Edge Intersections

```
for $i = 0:3$ do
    $kv(i) \leftarrow$ kernel vertex
    $ID(i) \leftarrow$ get element id$(kv(i))$
end for
$kv(4) \leftarrow kv(0)$
for $n = 0:4$ do
    if $ID(n)! = ID(n+1)$ then
        while $kv(n)! = kv(n+1)$ do
            $s = \overline{kv(n)kv(n+1)}$   # segment
            do
                $e \leftarrow ID(n) \rightarrow$ get edge
            while $s \cap e = \emptyset$
            $kv(n) \leftarrow$ *get intersection point*$(s, e)$
            Collect coordinates and element id's
            $ID(n) \leftarrow$ get element id$(kv(n))$
        end while
    end if
end for
```

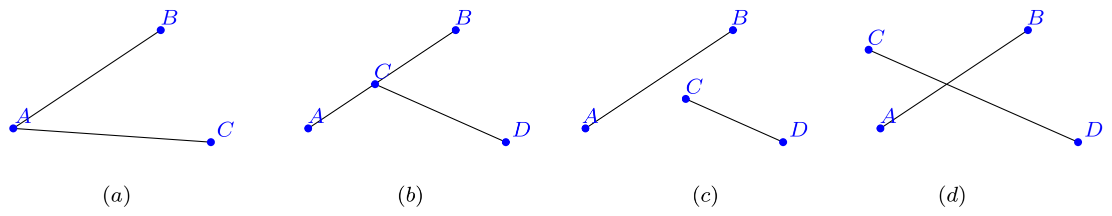


Figure 5.5: The four possible intersection types of two segments on the 2D plane.

point() function is evaluated in the following way. There are essentially four possible relative positions of the element edge and kernel break trace (see Figure 5.5). Each case is treated as follows:

- Case (a): Compare the coordinates of the four points and see if any two coincide.
- Case (b): Build the vectors AB, AC and BC and compute its norms:

$$C \in \overline{AB} \Leftrightarrow |AC| + |BC| = |AB|.$$

- Cases (c) & (d): Calculate the orientation of the triplets A, B, C and A, B, D. If the signs are different, the segments intersect. Figure 5.6 shows how the signs change according to the relative position of the points.

More details on the segment intersection algorithm can be found, for example, in [46, Ch. 1].

The previous algorithms allow gathering all the points needed to construct the regions where the integral will be computed. Now the question on how to construct

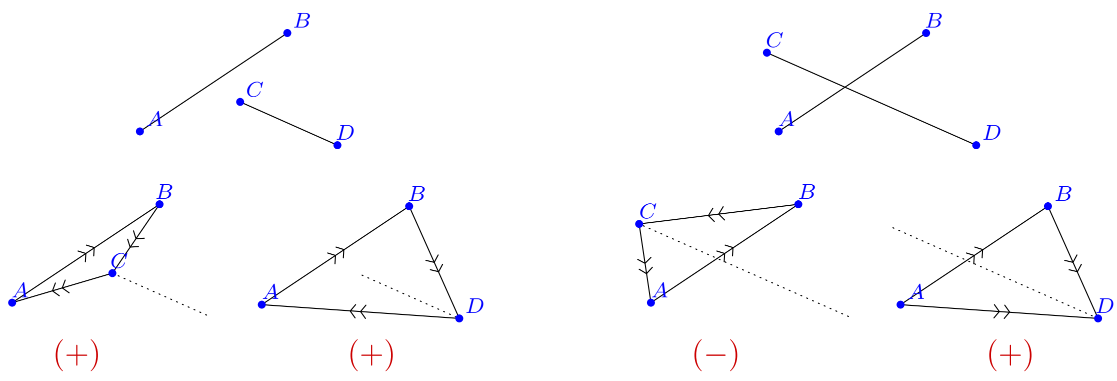


Figure 5.6: Segment intersection using orientations. The left group of diagrams illustrates the set of points $\{A, B, C\}$ and $\{A, B, D\}$ both with clockwise orientation (+ sign). In the right set of diagrams, the orientation changes and therefore the segments intersect.

integrable regions is discussed. During the intersection algorithm, for each point, the coordinates and element IDs are stored. Notice that edge intersection points belong to two adjacent elements, hence have two IDs. This allows sorting integration regions element by element in an efficient way by scanning and collecting all points with the same ID(s). However, it does not ensure that the points are ordered properly as shown in the left image of Figure 5.7. The resulting polygon linking $\{P_0P_1, \ldots, P_5\}$ is self-intersecting and is not suitable for numerical integration. The last routine presented here consists of sorting a set of randomly ordered points so that they form a convex polygon. The idea is borrowed from the famous Graham's Scan Algorithm [22], designed to solve the convex hull problem [46, Ch. 3,4].

Convex Polygon Technique

1. Find the point with lowest y-coordinate and draw a horizontal ray through the point.

2. Join that point to all the other vertices and calculate the angle with respect to the ray.

3. Sort the points by increasing angle size. The result is a counter-clockwise oriented polygon (see Figure 5.7).

5.2.4 Numerical Integration over Arbitrary Regions

The previous section explained how to find all the integration regions in which the filter has support. The remaining question is how to actually solve the integrals numerically. A detailed discussion on the existing numerical techniques for approximating integrals can be found in [62, Ch. 3] and [3, Ch. 4]. Here, the Gauss quadrature rules are discussed and it is shown how to apply them to effectively solve the 2D filtering convolution (equation (3.9)).

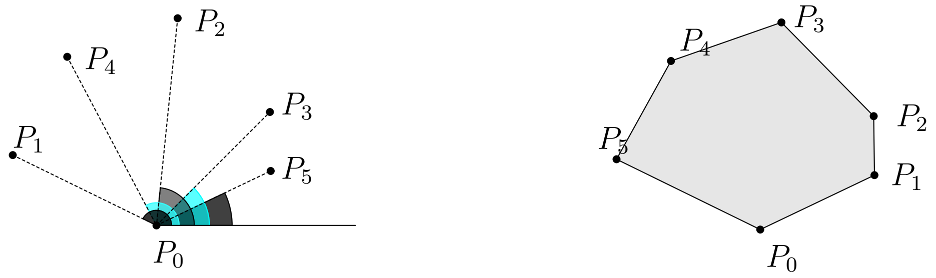


Figure 5.7: Convex polygon construction. The left image shows a set of points randomly organized and the angle with respect to the horizontal ray with origin P_0. The right figure is the result of ordering the points by increasing angles.

Theorem 5.2.2. *Gauss Quadrature. If $f \in \mathcal{C}^{2n}[a,b]$, then*

$$\int_a^b w(x)f(x)dx - \sum_{i=1}^{n} w_i f(x_i)u(\xi) = \frac{f^{(2n)}(\xi)}{(2n)!}(p_n, p_n), \quad \xi \in (a,b). \tag{5.13}$$

Here, p_n is the n^{th} orthogonal polynomial, x_i are its roots and w_i the associated weight functions [62, Ch. 3.6].

When the evaluating function is a polynomial, this technique is exact if enough quadrature points are used, *i.e.*, n points integrate exactly polynomials up to degree $2n-1$. Both the DG solution and the SIAC kernel have a polynomial representation. In fact, convolving a kernel $K_H^{(2k+1,k+1)}$ with a DG solution u_h of order k, gives a polynomial of degree $2k+2$. Hence, using $k+1$ quadrature points leads to exact integration.

Multidimensional Gaussian rules are calculated as a tensor product of univariate quadratures. For efficiency, the integral is computed over standard regions. In 2D, this corresponds to quadrilateral and triangular standard regions which are shown in Figure 5.8. For arbitrary polygons consisting of more than four vertices, the area is subdivided into quadrilateral and triangular subregions. The DG solutions used throughout this thesis were computed using Nektar++ software [7]. This DG scheme is implemented using a generalised tensorial bases. Taking advantage of this construction, the triangular standard region is written in the collapsed Cartesian system [31, Ch. 3]. This is nothing but the standard quadrilateral region with two collapsed vertices. It is a robust formulation suitable for Gauss integration [31, Ch. 4].

Integration over Quadrilateral Regions

Define the standard quadrilateral region by

$$\mathcal{Q}^2 = \{-1 \leq \xi_2,\ \xi_2 \leq 1\}. \tag{5.14}$$

The numerical integration over $\mathcal{Q}^2$ is defined as a product of two uninvariate integrals:

$$\int_{\mathcal{Q}^2} u(\xi_1,\xi_2)d\xi_1 d\xi_2 = \int_{-1}^{1}\left\{\int_{-1}^{1} u(\xi_1,\xi_2)|_{\xi_2} d\xi_1\right\} d\xi_2. \tag{5.15}$$

The Gaussian approximation to the integral is straight-forward:

$$\int_{Q^2} \sum_{i=0}^{Q_i-1} w_{1i} \left\{ \sum_{j=0}^{Q_j-1} u(\xi_{1i}, \xi_{2j}) w_{2j} \right\}, \tag{5.16}$$

where $\{\xi_{1i}, w_{1i}\}$ and $\{\xi_{1i}, w_{1i}\}$ denote the quadrature points and weights in each direction. The points distribution is shown in Figure 5.8 (left).

For general quadrilateral regions Ω_e with straight sides, define the mapping to the standard region via its vertices $\left\{X^A, X^B, X^C, X^D\right\}$, $X = (x_1, x_2)$:

$$x_i = x_i^A \frac{1-\xi_1}{2}\frac{1-\xi_2}{2} + x_i^B \frac{1+\xi_1}{2}\frac{1-\xi_2}{2} \tag{5.17}$$

$$+ x_i^D \frac{1-\xi_1}{2}\frac{1+\xi_2}{2} + x_i^C \frac{1+\xi_1}{2}\frac{1+\xi_2}{2}, \quad i = 1, 2. \tag{5.18}$$

The integral over the element Ω_e can be written by

$$\int_{\Omega^e} u(x_1, x_2) dx_1 dx_2 = \int_{\Omega_{st}} u(\xi_2, \xi_2) |J_{2D}| d\xi_1 d\xi_2 \tag{5.19}$$

with

$$J_{2D} = \begin{vmatrix} \frac{\partial x_1}{\partial \xi_1} \frac{\partial x_1}{\partial \xi_2} \\ \frac{\partial x_2}{\partial \xi_1} \frac{\partial x_2}{\partial \xi_2} \end{vmatrix} = \frac{\partial x_1}{\partial \xi_1}\frac{\partial x_2}{\partial \xi_2} - \frac{\partial x_1}{\partial \xi_2}\frac{\partial x_2}{\partial \xi_1}. \tag{5.20}$$

Hence the Gaussian Quadrature for a general quadrilateral region is given by the formula:

$$\int_{\Omega_e} u(x_1, x_2) dx_1 dx_2 \simeq \sum_{i=0}^{Q_i-1} w_{1i} \left\{ \sum_{j=0}^{Q_j-1} u(\xi_{1i}, \xi_{2j}) w_{2j} \begin{vmatrix} \frac{\partial x_1}{\partial \xi_1}(\xi_{1i}, \xi_{2j}) & \frac{\partial x_1}{\partial \xi_2}(\xi_{1i}, \xi_{2j}) \\ \frac{\partial x_2}{\partial \xi_1}(\xi_{1i}, \xi_{2j}) & \frac{\partial x_2}{\partial \xi_2}(\xi_{1i}, \xi_{2j}) \end{vmatrix} \right\} \tag{5.21}$$

Integration over Triangular Regions

Define the standard triangular region by

$$\mathcal{T}^2 = \{-1 \le \xi_1 \le \xi_2, \xi_2, \xi_1 + \xi_2 \le 0\}.$$

The *two dimensional collapsed coordinate system* is defined by the transformation:

$$\eta_1 = 2\frac{1+\xi_1}{1-\xi_2} - 1, \quad \eta_2 = \xi_2, \tag{5.22}$$

which has the inverse transformation

$$\xi_1 = \frac{(1+\eta_1)(1-\eta_2)}{2} - 1, \quad \xi_2 = \eta_2. \tag{5.23}$$

The new coordinates (η_1, η_2) define the standard triangular region:

$$\mathcal{T}^2 = \{(\eta_1, \eta_2) | -1 \le \eta_1, \eta_2 \le 1\},$$

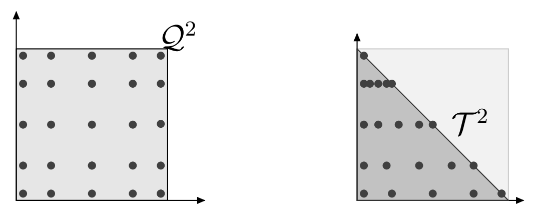


Figure 5.8: Quadrature points in the standard regions (quadrilateral and triangle) using Gauss-Legendre nodes in both directions.

and the integral is defined by:

$$\int_{\mathcal{T}^2} u(\xi_1,\xi_2)d\xi_1 d\xi_2 = \int_{-1}^{1}\int_{-1}^{-\xi_2} u(\xi_1,\xi_2)d\xi_1 d\xi_2 \tag{5.24}$$

$$= \int_{-1}^{1}\int_{-1}^{1} u(\eta_1,\eta_2)\left|\frac{\partial(\xi_1,\xi_2)}{\partial(\eta_1,\eta_2)}\right| d\eta_1 d\eta_2, \tag{5.25}$$

where the Jacobian can be expressed as

$$\frac{\partial(\xi_1,\xi_2)}{\partial\eta_1,\eta_2} = \frac{1-\eta_2}{2}.$$

In the new coordinate system, Gaussian quadrature is analogous to the standard quadrilateral case:

$$\int_{-1}^{1}\int_{-1}^{1} u(\eta_1,\eta_2)\frac{1-\eta_2}{2} d\eta_1 d\eta_2 \simeq \sum_{i=0}^{Q_i-1} w_{1i}\left\{\sum_{j=0}^{Q_j-1} u(\eta_{1i},\eta_{2j})w_{2j}\frac{1-\eta_{2j}}{2}\right\}. \tag{5.26}$$

Figure 5.8 shows the distribution of the quadrature points before and after one vertex being collapsed. For a general triangle with right sides and vertices

$$\{(x_1^A,x_2^A),(x_1^B,x_2^B),(x_1^C,x_2^C)\}.$$

the mapping to the standard region is given by:

$$x_i = x_i^A\frac{1-\eta_1}{2}\frac{1-\eta_2}{2} + x_i^B\frac{1+\eta_1}{2}\frac{1-\eta_2}{2} + x_i^C\frac{1+\eta_2}{2}, \quad i=1,2. \tag{5.27}$$

where C is the collapsed vertex.

The Gauss quadrature rules over general triangular regions are given by

$$\int_{\Omega_e} u(x_1,x_2)dx_1dx_2 \simeq \sum_{i=0}^{Q_i-1} w_{1i}\left\{\sum_{j=0}^{Q_j-1} u(\eta_{1i},\eta_{2j})w_{2j}\frac{1-\eta_{2j}}{2}\begin{vmatrix}\frac{\partial x_1}{\partial\eta_1}(\eta_{1i},\eta_{2j}) & \frac{\partial x_1}{\partial\eta_2}(\eta_{1i},\eta_{2j})\\ \frac{\partial x_2}{\partial\eta_1}(\eta_{1i},\eta_{2j}) & \frac{\partial x_2}{\partial\eta_2}(\eta_{1i},\eta_{2j})\end{vmatrix}\right\}.$$

5.2.5 Implementation of the SIAC Line Filter

The one-dimensional support of the SIAC Line filter results in a great reduction of the number of operations required to post-process each point. Therefore, both the simulation times and the level of difficulty of the implementation decrease. The support of tensor product filters requires searching for DG mesh vertices as well as classifying the points type (see Section 5.2.3). On the other hand, for the Line kernel support, it is only necessary to find if there are any element interfaces between two consecutive break points. Hence, the implementation is similar to SIAC filters for one-dimensional problems. The pseudo code for these filters is given in Algorithm 3.

Algorithm 3 Line Filtering Convolution

```
$N \leftarrow$ get number of kernel breaks
for $b = 0 : N - 1$ do
    $kv(b) \leftarrow$ kernel vertex
    $ID(b) \leftarrow$ get element id$(kv(b))$
end for
for $b = 0 : N - 2$ do
    if $ID(b) == ID(b + 1)$ then
        $Integral+ =$ evaluate convolution$(kv(b), kv(b + 1))$
    else
        while $ID(b)! = ID(b + 1)$ do
            $s = \overline{kv(b)kv(b + 1)}$
            do
                $e \leftarrow ID(b) \rightarrow$ get edge
            while $s \cap e = \emptyset$
            $p \leftarrow$ $get\ intersection\ point(s, e)$
            $Integral+ =$ evaluate convolution$(kv(b), p)$
            $kv(b) \leftarrow p$;
            $ID(b) \leftarrow$ get right element id
        end while
        if $kv(b)! = kv(b + 1)$ then
            $Integral+ =$ evaluate convolution$(kv(b), kv(b + 1))$
        end if
    end if
end for
```

5.3 Computational Study

The experiments presented next, study the filters from a computational point of view. Several types of filters and meshes were tested for the number of operations and simulation times that each filter requires to post-process a particular point. Figure 5.9 shows the footprints of three different tensor product filters and a Line filter highlighting the partition of the integral. Observe how the $\pi/6$ rotation produces a random partition compared to the repeated patterns in the other two 2D kernels. This is because the combination of the rotation angle and scaling results in a translation invariant space for the $\pi/4$ (and $3\pi/4$) rotations. The total number of integrals and quadrature sums are shown in Table 5.1. Line filters use one dimensional quadrature rules and the total number of integrals and quadrature sums match. On the other hand, the total number of sums using 2D filters is increased by a factor of n^2, where n denotes the total number of integrals. This equips line filters with excellent computational attributes; not only there are significantly less number of integration regions compared to tensor product filters but also such number does not grow when applying the numerical integration technique. Look for example at the highest degree Cartesian axis aligned 2D filter in Table 5.1. The convolution is split into 400 regions and it requires 160,000 quadrature sums. The Line filter on the other hand, divides the integral into 30 intervals only and performs 30 quadrature sums. Figure 5.10 shows the elapsed times required to post-process a single point using the filters from Figure 5.9 over two meshes. Notice that not only the computational times are significantly reduced when using a Line filter but also indicate that increasing the number and degree of the splines in the kernel slightly modifies such times. This represents a great advantage, as one important limiting factor on the applications of 2D SIAC filters is the long computational times of higher degree filters. Observing the plots, even for the highest order Line filters (using a $K^{(9,5)}$ kernel), the elapsed time is significantly lower than the one required to filter a point using the lowest degree tensor product kernel $K^{(3,2)}$. Figure 5.11 shows the footprint of a 2D filter and a Line filter applied to a nonuniform quadrilateral mesh. Although the number of integration regions increases, the values remain relatively close to those for the uniform case. Triangular elements, however, imply doubling the number of integration regions for tensor product filters. The Line filters indeed increase the number of integrals but the cut across the elements is similar to the quadrilateral meshes. This can be seen in Figure 5.12. Table 5.2 compares the number of integrals for the uniform and nonuniform quadrilateral meshes as well as the triangular mesh. From these experiments, one can see the clear computational advantages of Line filtering.

5.4 Discussion

From a computational perspective, the rotated filters have contributed towards the general application of SIAC filters; designing a filter that allows for post-processing in any direction and size produces a very robust algorithm. The methodology proposed

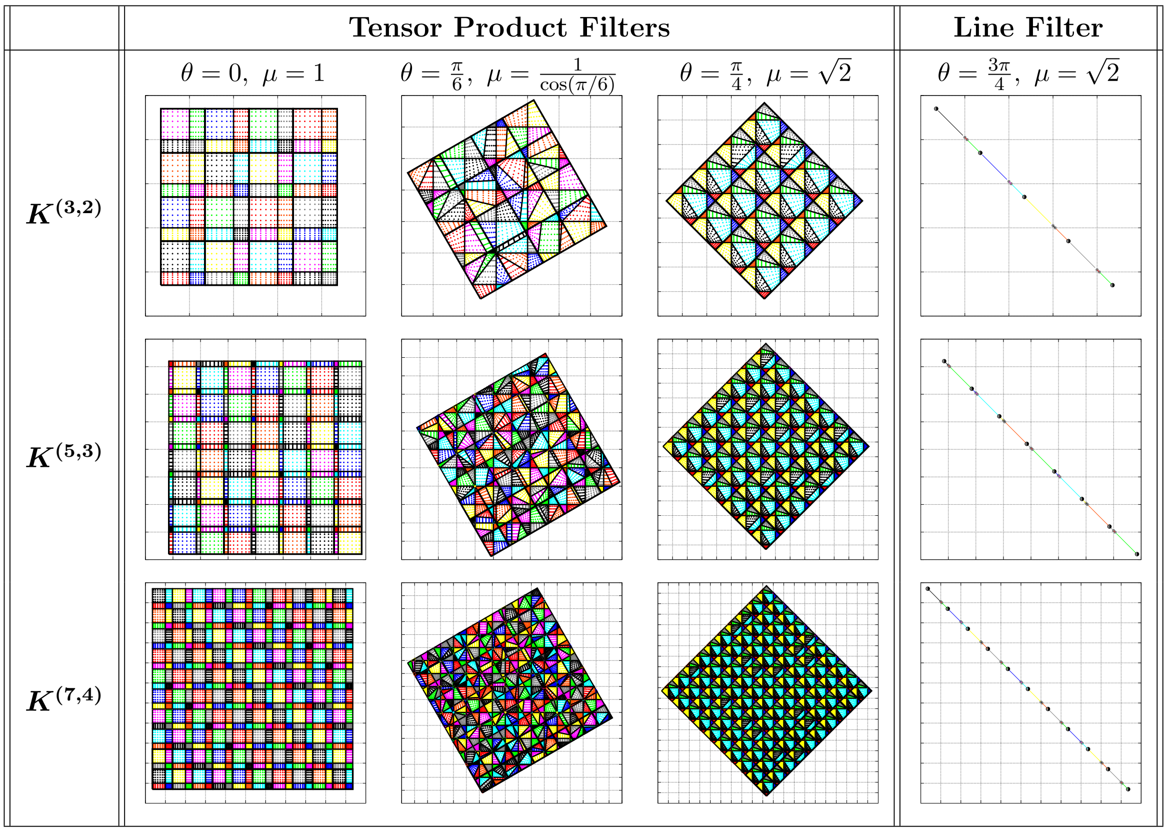

Tensor Product Filters
Line Filter
$\theta = 0,\ \mu = 1$
$\theta = \frac{\pi}{6},\ \mu = \frac{1}{\cos(\pi/6)}$
$\theta = \frac{\pi}{4},\ \mu = \sqrt{2}$
$\theta = \frac{3\pi}{4},\ \mu = \sqrt{2}$
$\boldsymbol{K^{(3,2)}}$
$\boldsymbol{K^{(5,3)}}$
$\boldsymbol{K^{(7,4)}}$

Figure 5.9: Integration regions for post-processing a single point applying different filters over an uniform mesh.

	Tensor Product Filters			Line Filter
Rotation angle	0	$\pi/6$	$\pi/4$	$3\pi/4$
$K^{(3,2)}$				
Intersection Scans	64	64	64	4
Integrals	64	115	144	12
Quadrature Sums	4096	13225	20736	12
$K^{(5,3)}$				
Intersection Scans	196	196	196	7
Integrals	196	337	441	21
Quadrature Sums	38416	113569	194481	21
$K^{(7,4)}$				
Intersection Scans	400	400	400	10
Integrals	400	699	900	30
Quadrature Sums	160000	488601	810000	30

Table 5.1: Summary of the number of operations required to compute the filtering convolution for the filters from Figure 5.9.

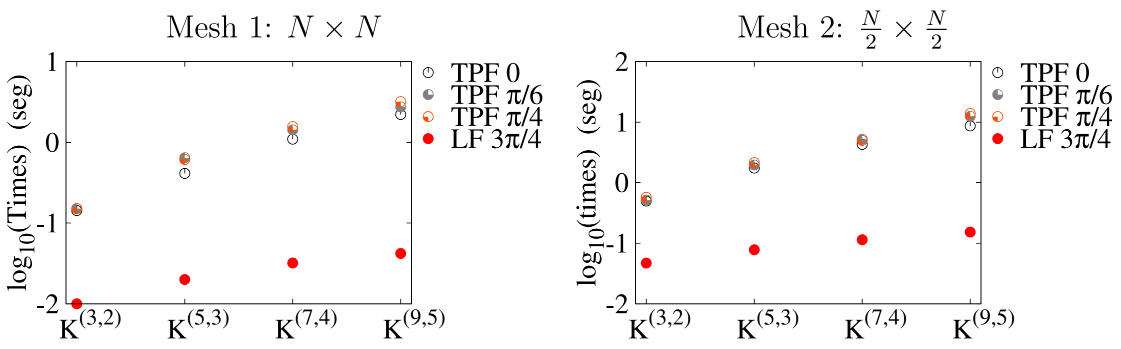


Figure 5.10: Computational times required to post-process one point applying different Tensor Product Filters (TPFs) and a Line Filter (LF) using kernels $K^{(2k+1,k+1)}$, $k = 1, \ldots, 4$.

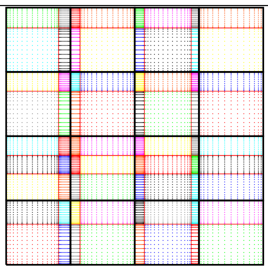
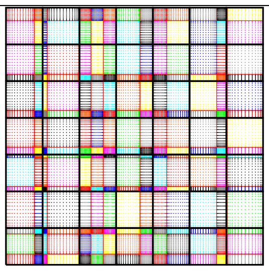
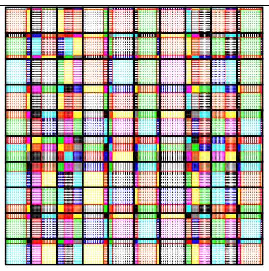
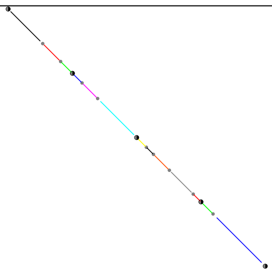
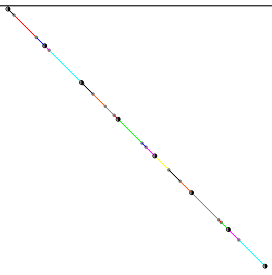
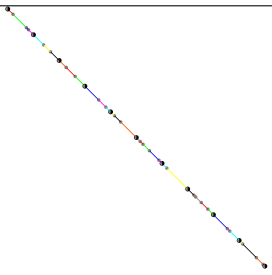
	$\boldsymbol{K^{(3,2)}}$	$\boldsymbol{K^{(5,3)}}$	$\boldsymbol{K^{(7,4)}}$
TPF $\theta = 0, \mu = 1$			
LF $\theta = \dfrac{3\pi}{4}, \mu = \sqrt{2}$			

Figure 5.11: Integration regions for post-processing a single point applying a Tensor Product Filter (TPF) and a Line Filter (LF) over a nonuniform mesh. The black lines in the top row denote the kernel boxes.

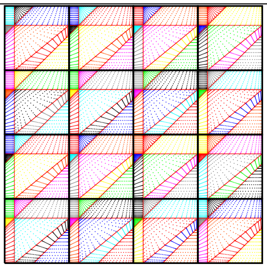
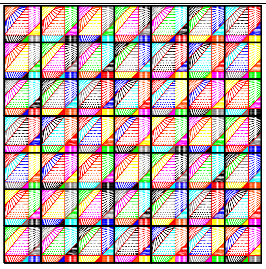
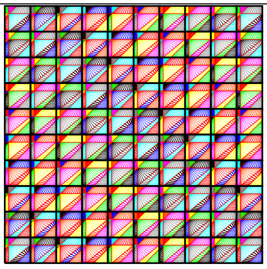
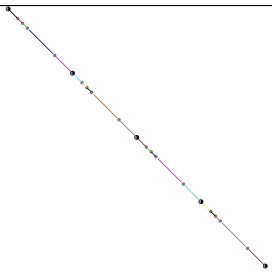
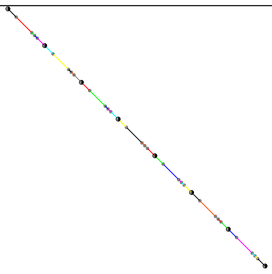
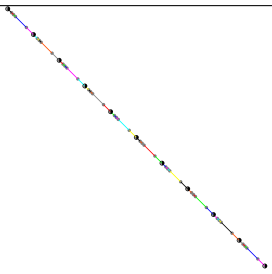
	$\boldsymbol{K^{(3,2)}}$	$\boldsymbol{K^{(5,3)}}$	$\boldsymbol{K^{(7,4)}}$
TPF $\theta = 0, \mu = 1$			
LF $\theta = \frac{3\pi}{4}, \mu = \sqrt{2}$			

Figure 5.12: Integration regions for post-processing a single point applying a Tensor Product Filter (TPF) and a Line Filter (LF) over a uniform triangular mesh. The black lines in the top row denote the kernel boxes.

	Uniform Mesh				Nonuniform Mesh	
	Quads		Triangles		Quads	
	TPF	LF	TPF	LF	TPF	LF
$K^{(3,2)}$						
Integrals	64	12	128	20	72	13
Quadrature Sums	4096	12	16384	20	5184	20
$K^{(5,3)}$						
Integrals	196	21	392	35	225	24
Quadrature Sums	38416	21	153664	35	50625	24
$K^{(7,4)}$						
Integrals	400	30	720	50	490	37
Quadrature Sums	160000	30	518400	50	240100	37
$K^{(9,5)}$						
Integrals	676	39	1352	65	754	45
Quadrature Sums	456976	39	1827904	65	568516	45

Table 5.2: Summary of the number of operations required to compute the filtering convolution for several mesh types using a Tensor Product Filter aligned with the Cartesian axis and a Line Filter along the $3\pi/4$ direction.

to implement tensor product and line filters here does not rely on any mesh geometry assumptions. The only restriction is that the Gauss Quadrature rules assumed elements with straight sides. However, this limitation can be easily overcome using for example, the mappings for general curvilinear elements given in [31, Ch. 4]. Thus, even for a zero rotation, this implementation is suitable to effectively apply the filters to general non-uniform meshes as shown in the examples from Section 5.3. Finally, the one-dimensional support of the Line filter presents great computational advantages. The algorithm design gives a relatively straightforward implementation which is similar that one used for one-dimensional problems. Furthermore, they lead to short simulation times, even for higher order kernels or triangular meshes, which is a promising tool for the visualization community.

Chapter 6

SIAC Filters and Streamline Visualisation

The goal of a numerical simulation is to provide an approximate solution of a model designed to understand a physical problem such as flow past an aircraft or weather forecasting. Hence, it is necessary to apply visualisation techniques that extract and evaluate the information from the numerical solution. Vector field visualisation through streamlines is a popular post-processing technique employed to understand fluid flow behaviour. Streamlines, curves everywhere tangent to the velocity field, are described by an Ordinary Differential Equation (ODE) and there are many numerical methods designed to solve ODEs such as the Runge-Kutta schemes [5]. However, the theoretical error estimates of these methods rely on Taylor series and therefore assume smooth field conditions [4, Ch. 3] [6]. Vector fields obtained through a DG method present constraints since the solution is only continuous inside each element. A suitable solver for computing streamlines over non-smooth fields has to be able to detect, locate and effectively step over a discontinuity [20]. This can be achieved through a Predictor-Corrector method [26,34] or by controlling the error through adaptive step size methods such as the Runge-Kutta-Fehlberg solvers [18, 19]. The downside of these methods is that they require intense computations since detecting and passing over a discontinuity implies increasing the number of evaluations per iteration. Alternatively, SIAC filters can be applied to obtain a local smooth solution where a relatively simple ODE solver can be implemented. Furthermore, since the filtered solution usually reduces the error from the DG approximation, the new filtered velocity field should lead to more accurate field lines.

Applying SIAC filters for flow visualisation implies combining different kernel types. For example, during streamline computations, since particles can move across the entire field, the filter has to be able to post-process points at the boundaries of the computational domain. In chapter 2, numerical results were given for the one dimensional boundary filters. The error plots (see Figure 2.4) suggested that these filters are not as effective as the symmetric filters when reducing the error from the DG approximation. Hence, in this chapter, boundary Line filters are included in the numerical experiments to give insight into how much accuracy is lost when applying such filters even though a suitable analysis has not been performed. On the other hand, the numerical results for Line filters presented in chapter 4 suggested that alternative orientations to those for which superconvergence can be proven still lead to error reduction. This was observed for the cases where the filter was oriented using the flow direction or when symmetries from the initial condition were used to choose the orientation. The results showed that for such alignments, the filtered solution presented lower errors than the original one. This can be exploited further near the boundaries, rotating the filter conveniently to fit a symmetric kernel, thus avoiding shifting its support.

This chapter investigates the potential of SIAC Line filters for accuracy enhancement during flow visualisation. A series of numerical experiments are presented where different kernel types have been implemented. The results are compared to traditional Tensor Product filtering in order to understand the trade-offs between computational performance and maximum accuracy resulting from reducing the dimension through Line filtering. Before presenting the numerical experiments, the next section provides a brief background on streamlines and ODE solvers.

6.1 Streamlines and ODE solvers

Let $U = (\overrightarrow{u_1}, \overrightarrow{u_2}, \ldots, \overrightarrow{u_n})$ be a vector field defined over the domain $\Omega \subset \mathbb{R}^n$. Streamlines are curves everywhere tangent to the vector field so streamline $\Gamma \subset \Omega$ satisfies

$$\Gamma' = (\overrightarrow{u_1}, \overrightarrow{u_2}, \ldots, \overrightarrow{u_n}),$$

where the sign $'$ above denotes the derivative. Consider a two dimensional field $U = (u(x,y), v(x,y))$. Then the streamlines are described by the first order ODE:

$$y'(x) = \frac{dy}{dx} = \frac{v(x,y)}{u(x,y)}. \tag{6.1}$$

Given a seed (initial condition), the solution to the streamline is found by solving the Cauchy problem:

$$\begin{cases} y'(x) = \dfrac{v(x,y)}{u(x,y)}, & (x,y) \in \Omega, \\ y_0 = y(x_0). \end{cases} \tag{6.2}$$

The following Theorem gives insight into how to develop numerical schemes that solve problem (6.2) numerically.

Theorem 6.1.1. *(Picard-Lindelöf [23, Ch. 2]). Consider the Cauchy Problem*

$$y' = f(x,y), \quad y(x_0) = y_0,$$

where f is a continuous function satisfying the Lipschitz condition:

$$|f(x,y_1) - f(x,y_2)| \leq L|y_1 - y_2| \tag{6.3}$$

in some open rectangle $R = \{(x,y) : \ a < x < b, \ c < y < b\}$ containing the point (x_0, y_0). Then the problem has unique solution in some closed interval $I = [x_0 - h, x_0 + h], \ h > 0$ and the Picard iteration

$$y_{n+1}(x) = y_0 + \int_{x_0}^{x_n} f(x, y_n(x))dx \tag{6.4}$$

produces a sequence of functions $y_n(x)$ that converges to this solution uniformly on I.

The differences between the types of ODE solvers are the number of steps employed to find the solution (single or multistep methods) and the way that the integral in equation (6.4) is approximated. For example, the explicit 2 stage Runge-Kutta (RK2) method is built in the following way. Assume that the function at time n, $f(x_n, y_n)$, is approximated by the midpoint of the interval $[x_n, x_{n+1}]$. Then,

$$\begin{aligned} y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} f(x, y(x))dx &\approx y_n + \int_{x_n}^{x_{n+1}} f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2} f(x_n, y_n)\right) dx \\ &= y_n + hf\left(x_n + \frac{h}{2}, y_n + \frac{h}{2} f(x_n, y_n)\right), \qquad h = x_{n+1} - x_n \end{aligned}$$

This can be written in two stages by

$$k_1 = f(x_n, y_n) \tag{6.5}$$

$$k_2 = f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2} k_1\right) \tag{6.6}$$

$$y_{n+1} = y_n + hk_2. \tag{6.7}$$

For streamline computations, identify f with $U = (u, v)$ (for the 2D case).

Adaptive methods control the error at each iteration and modify the stepsize to ensure that the error remains under a certain tolerance. For example, the RKF45 method uses a fourth order Runge-Kutta method as an estimator and computes the actual solution using a fifth order RK method [19]. The idea behind introducing the filter between the ODE solver and the DG field is to reduce the computational costs by using a lower order method, *e.g.* RK2 method, assuming that the filtering step is cheaper than adaptive error control. Previous work on tensor product filters for streamline visualisation implied post-processing the entire field prior to streamline computations [61]. The authors observed that for strict adaptive methods, filtering resulted in lower computational times. However, filtering the entire field adds unnecessary computational costs since streamlines only follow a particular region of the domain. Therefore, by only post-processing points which are used by the ODE solver, the computational times can already be improved. In addition, replacing the tensor product filter by a Line filter improves even further the efficiently of this post-processor. The following experiments study symmetric filters and compare the performance of the Line filters against the 2D filter aligned with the Cartesian axis.

6.2 Symmetric SIAC Filters

Theorem 4.3.1 shows how Line filters can extract superconvergence provided the appropriate kernel orientation and scaling are selected. Furthermore, the numerical experiments from Chapter 4 revealed that in addition to smoothness recovery, these filters reduce significantly the error from the DG approximation. This represents a great advantage during streamline integration, where error reduction is more desirable than extracting superconvergence. Hence, the following experiments investigate the potential of Line filters for accuracy enhancement. This study begins by applying symmetric kernels for different Line filters and comparing the results against Tensor Product filtering, both in terms of accuracy and computational costs.

The streamline experiments were done over complex analytic fields of the form:

$$z = x + iy, \tag{6.8}$$

$$u = Re(r), \tag{6.9}$$

$$v = -Im(r), \tag{6.10}$$

where the first field, CF1, was given by:

$$\begin{aligned} r =&(z - (0.74 + 0.35i))(z - (0.68 - 0.59i))(z - (-0.11 - 0.72i)) && (6.11)\\ &(\overline{z} - (-0.58 + 0.64i))(\overline{z} - (0.51 - 0.27i))(\overline{z} - (-0.12 + 0.84))^2, && (6.12)\end{aligned}$$

and the second field, CF2, by:

$$\begin{aligned} r =&(z - (0.74 + 0.35i))(z + (-0.68 - 0.19i))(z - (-0.11 - 0.72i)) && (6.13)\\ &(\overline{z} - (-0.58 + 0.64i))(\overline{z} - (0.51 - 0.27i)). && (6.14)\end{aligned}$$

These fields have been studied before for 2D symmetric filtering by [61] and for more general filters by [27, 35]. The computational domain used for the simulations corresponded to $\Omega = [-1, 1] \times [-1, 1]$, using two uniform quadrilateral meshes made of 40×40 and 80×80 elements respectively. The unfiltered solutions were obtained by performing the L^2-projection of each function (CF1 and CF2) which mimics a DG solution at initial time.

6.2.1 Line Kernels

Recall that the proof for superconvergence (Theorem 4.3.1) relies on choosing the rotation angle to be $\theta = \arctan\left(\frac{h_x}{h_y}\right)$. For a mesh made of uniform square elements, this implies $\theta = \pi/4$ or $\theta = 3\pi/4$. Hence, both rotations were considered despite the fact that the numerical experiments in chapter 4 indicated that applying a $3\pi/4$-Line filter resulted in greater (or same) error reduction. In addition, since other orientations (see for example Figure 4.8) also allowed for error reduction, filters oriented using flow information have also been implemented.

The first experiment was done over a DG approximation using $\mathbb{P}^1$ polynomials. The filtered solutions were obtained with a $K_\Gamma^{(3,2)}$ symmetric Line filter with scaling

$H = \sqrt{2}h$, h being the DG mesh size. The flow based filters were calculated in the following way: the first orientation (at streamline seed) was chosen to be $3\pi/4$. The rest of the points were post-processed using the direction given by the last two computed streamline points or its perpendicular direction. Both the unfiltered and filtered streamlines were computed using the RK2 method with time step $dt = 0.01$. The final time was determined by the exact streamline, corresponding to the last point inside the computational domain or when a streamline reached zero velocity. The exact streamlines were obtained by implementing the RK4 method with time step $dt = 1e^{-5}$ directly on the analytic velocity fields.

Figure 6.1 shows streamlines belonging to the velocity field CF1 (equation (6.11)) using four different filter orientations based on the underlying mesh ($\theta = \pi/4, 3\pi/4$) and the flow direction. Observe how the flow based filters produced a diverging streamline for lower seed (starting at a critical point) even after mesh refinement whereas the $\pi/4$ and $3\pi/4$ filtered streamlines converged towards the exact curve. Actually, the $\pi/4$ Line filter performs better since for a coarse mesh (40×40 elements), the filtered streamline moved away from the exact solution initially and eventually converged towards it. Figure 6.2 shows another set of streamlines corresponding to the second velocity field, CF2 (equation (6.13)), applying the same filters. In this case, both the $\pi/4$ and $3\pi/4$ Line filters were able to produce three converging streamlines for both meshes. Notice that for the 40×40 element mesh, the unfiltered streamlines diverged in two cases. On the other hand, the 80×80 element mesh suggests that it is not necessary to filter the streamlines since all the unfiltered curves have already converged towards the exact solution. The filter aligned with the flow had a worse performance than the one aligned tangent to it. This filter produced two diverging streamlines which follow the same path as the unfiltered solution. On the other hand, the filter aligned tangent to the flow produced similar curves as those obtained through the $\pi/4$ and $3\pi/4$ orientations.

Table 6.1 shows two error estimates; the first is a local maximum computed through the formula:

$$\max_{n=0:N} e_n = \max_{n=0:N} d(p_n, \tilde{p}_n), \tag{6.15}$$

where $d(p, \tilde{p})$ denotes the Euclidean distance, p_n and $\tilde{p}_n$ the exact and approximate solutions respectively and $t_N = T$, *i.e.*, the final time. The global error corresponds to the difference between the solutions at final time. The errors in this table show that even when both the filtered and unfiltered streamlines converge, the filtered solution has generally lower values. Regarding the differences between both filters, the numbers are very similar, especially after mesh refinement. The results suggest that both the $\pi/4$ and $3\pi/4$ orientations are suitable for effective post-processing. This orientations should be chosen (whenever possible) instead of flow aligned filters since the latter ones require longer and more complicated computations and do not seem to produce more accurate streamlines. Hence, for the rest of the experiments in this chapter, only the $\pi/4$ and $3\pi/4$ rotations will be considered.

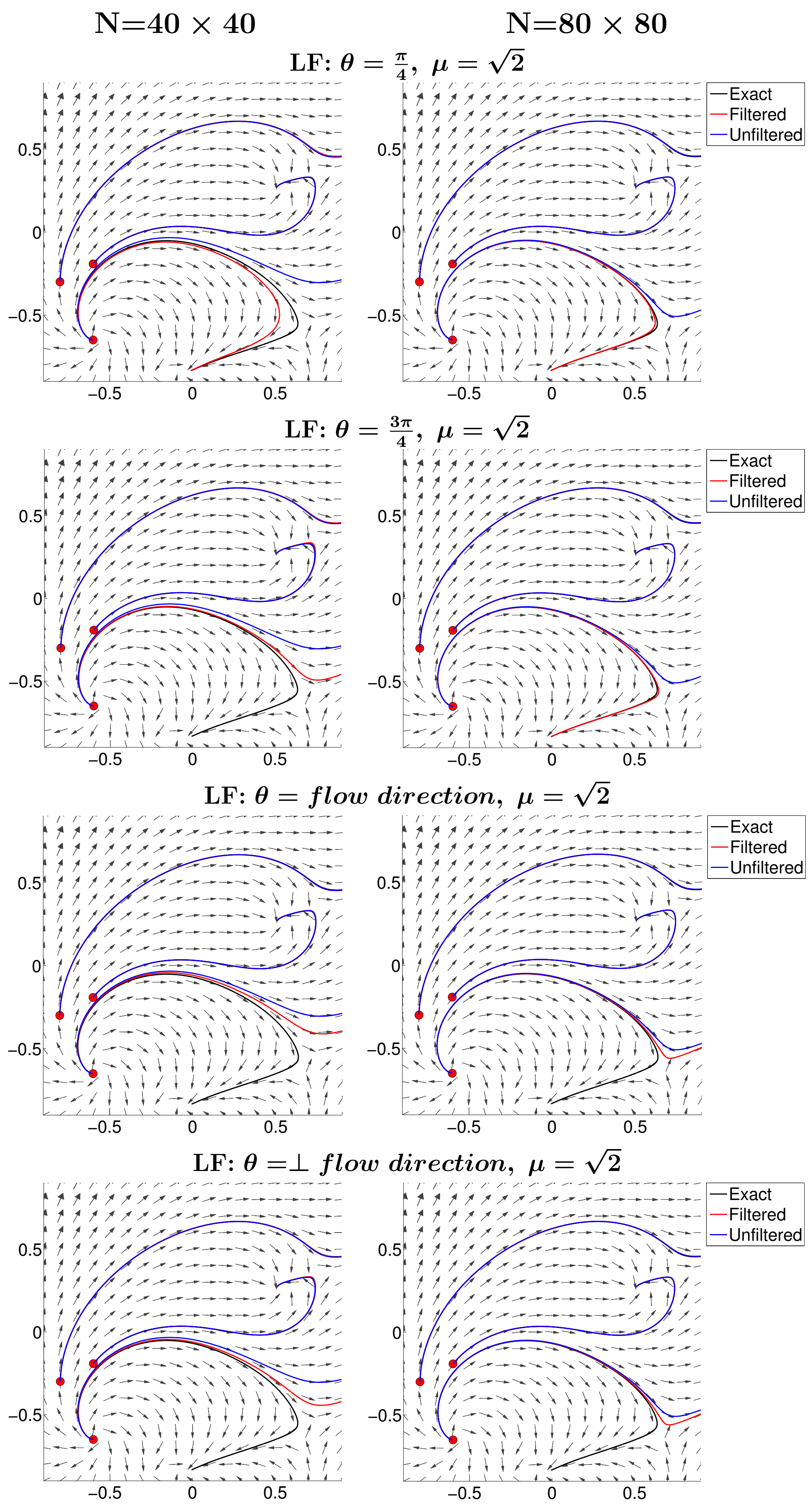


Figure 6.1: Streamlines along CF1 (equation (6.11)) for two meshes ($N = 40 \times 40$ and $N = 80 \times 80$) before and after applying different symmetric Line Filters (LFs) using the RK2 solver with $dt = 0.01$. $\perp$ denotes tangent direction. The plots where the exact curve cannot be seen is because it overlaps with the filtered streamline.

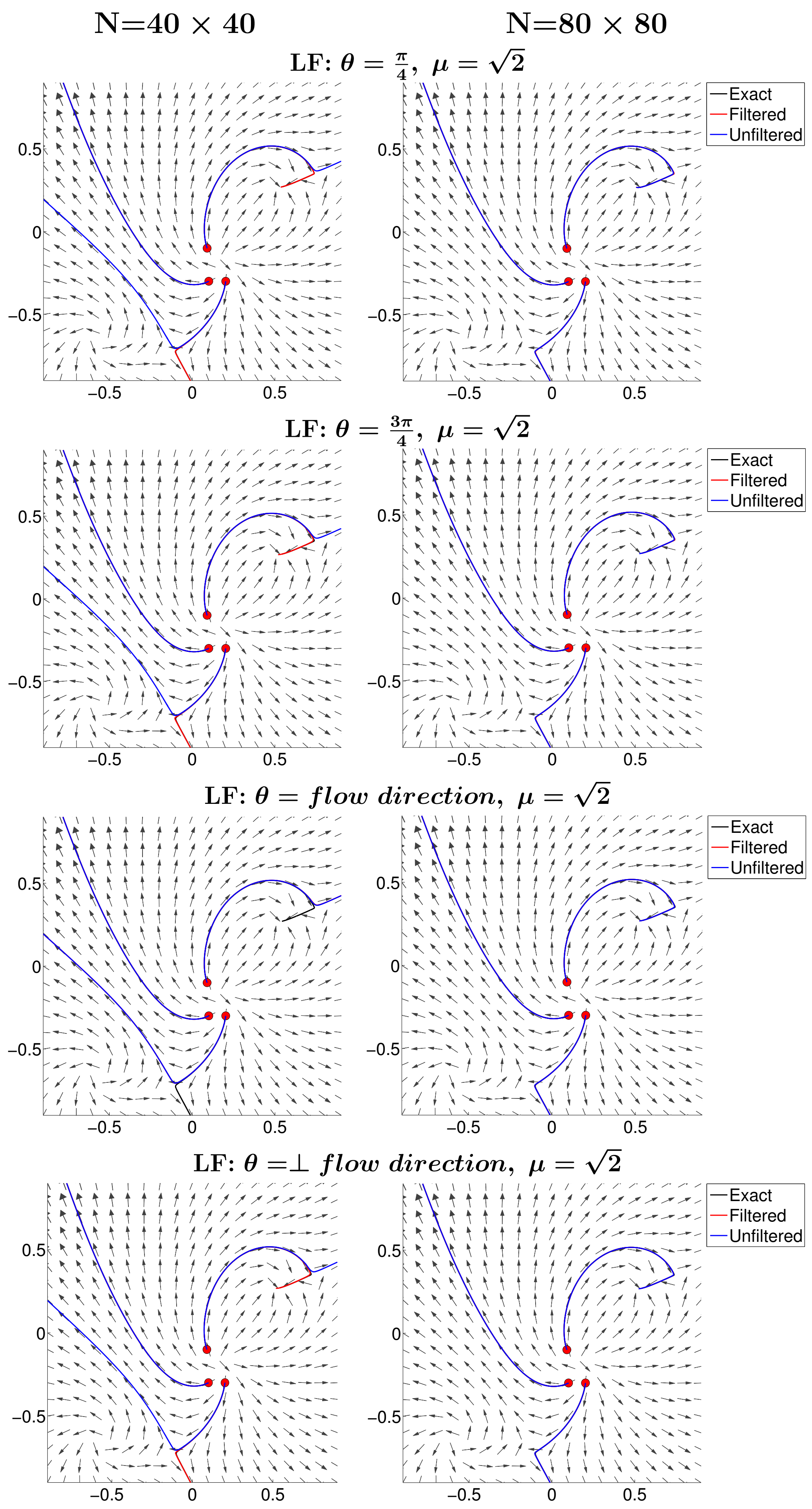


Figure 6.2: Streamlines along CF2 (equation (6.13)) for two meshes ($N = 40 \times 40$ and $N = 80 \times 80$) before and after applying different symmetric Line Filters (LFs) using the RK2 solver with $dt = 0.01$. $\perp$ denotes tangent direction. The plots where the exact curve cannot be seen is because it overlaps with the filtered streamline.

CF1						
	Unfiltered		**Line Filtering: $H = \sqrt{2}h$**			
			$\theta = \pi/4$		$\theta = \frac{3\pi}{4}$	
Seed	MD	GE	MD	GE	MD	GE
N= 40 × 40						
(-.6,-.651)	DIVERGED		2.1e-01	3.0e-02	DIV	DIV
(-.6,-.192)	4.0e-03	9.7e-04	6.1e-03	1.6e-04	3.1e-02	8.9e-05
(-.8,-.3)	4.1e-02	4.1e-02	3.8e-02	3.8e-02	2.2e-02	2.2e-02
N= 80 × 80						
(-.6,-.651)	DIVERGED		4.3e-02	7.3e-03	4.1e-02	7.7e-03
(-.6,-.192)	2.6e-03	1.2e-04	5.5e-04	9.6e-06	1.3e-03	5.2e-06
(-.8,-.3)	3.6e-02	3.6e-02	2.7e-02	2.7e-02	2.9e-02	2.9e-02

CF2						
	Unfiltered		**Line Filtering: $H = \sqrt{2}h$**			
			$\theta = \pi/4$		$\theta = \frac{3\pi}{4}$	
Seed	MD	GE	MD	GE	MD	GE
N= 40 × 40						
(0.202,-.3)	DIVERGED		1.8e-01	1.8e-01	2.7e-01	2.7e-01
(-.09,-.1)	DIVERGED		2.2e-02	7.0e-03	1.0e-01	2.3e-02
(.1,-.3)	5.4e-03	5.4e-03	7.1e-03	7.1e-03	8.0e-03	8.0e-03
N= 80 × 80						
(0.202,-.3)	2.2e-01	2.2e-01	1.2e-02	1.2e-02	2.5e-02	2.5e-02
(-.09,-.1)	7.6e-02	1.9e-02	7.0e-03	2.5e-03	1.8e-02	5.6e-03
(.1,-.3)	7.5e-03	7.5e-03	6.8e-03	6.8e-03	6.9e-03	6.9e-03

Table 6.1: Maximum Distance (MD) taken as the greatest point distance between each iteration and Global Errors (GE) measuring the point distance at final time comparing unfiltered and filtered streamlines for two different Line filters over two velocity fields (CF1 and CF2 (6.11)) and two different meshes ($N = 40 \times 40$ and $N = 80 \times 80$).

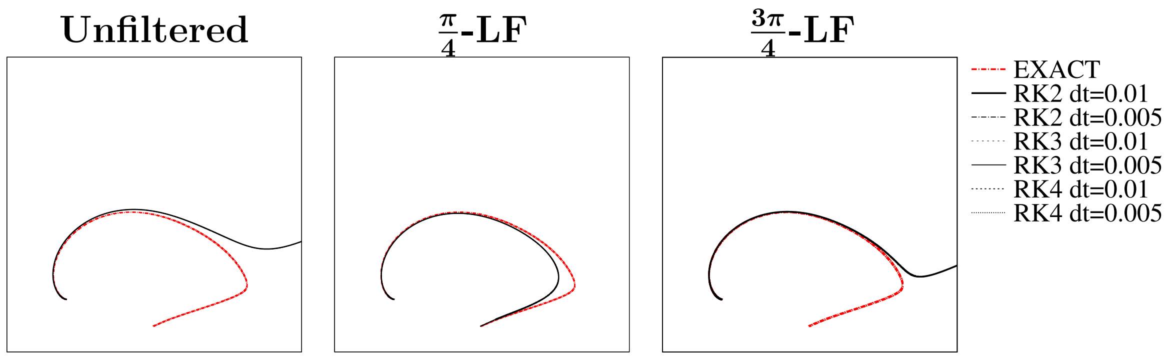


Figure 6.3: Unfiltered and filtered streamlines with seed near a critical point corresponding to the CF1 field (equation (6.11)) for different ODE solvers (RK2, RK3, RK4) and two time steps using 40×40 elements and $\mathbb{P}^1$ polynomials for the DG approximation.

Time Integrators and Polynomial Order

The previous experiments suggested that the DG mesh size plays a major role during post-processing. The next question was how much the solver type and time step affected the numerical results. Figure 6.3 studies the streamline from the first field (CF1) starting at the critical point. All the streamlines shown in the plots were computed over the 40×40 element mesh (notice that for the finer mesh, the RK2 method already gives satisfactory results) using three different solvers: RK2, RK3 and RK4 (explicit) and two different time steps: $dt = 0.01$, and $dt = 0.005$. Observe the overlap in all streamlines regardless the ODE solver type or time step. This indicates that implementing a higher order solver (RK3 or RK4) does not improve the streamline accuracy since the dominant errors come from the DG approximation to the velocity field.

The velocity fields used in the experiments were computed using a $\mathbb{P}^1$ polynomial basis. The results from Figures 6.1 and 6.2 show how $h-$refinement allows the filter to produce satisfactory streamlines. Hence, the same experiments were performed using a higher polynomial degree for the DG approximation ($p-$refinement). Figure 6.4 shows the same streamline from Figure 6.3 using $\mathbb{P}^2$ polynomials for the DG approximation and applying the RK2 method with different time steps. Notice that in this case, the time step $dt = 0.01$ already produces satisfactory streamlines for the unfiltered solution. On the other hand, a larger time step is not suitable for post-processing since all streamlines diverge, even for the 80×80 elements mesh. Figure 6.5 shows the same curve but using the RK3 solver. In this case, already at time $dt = 0.1$, even the unfiltered streamline converged towards the exact solution. This confirms that the reason why there was no improvement for the $\mathbb{P}^1$ polynomials after increasing the order of the ODE solver is due to dominant errors arising from computing the velocity field with too low order polynomial basis.

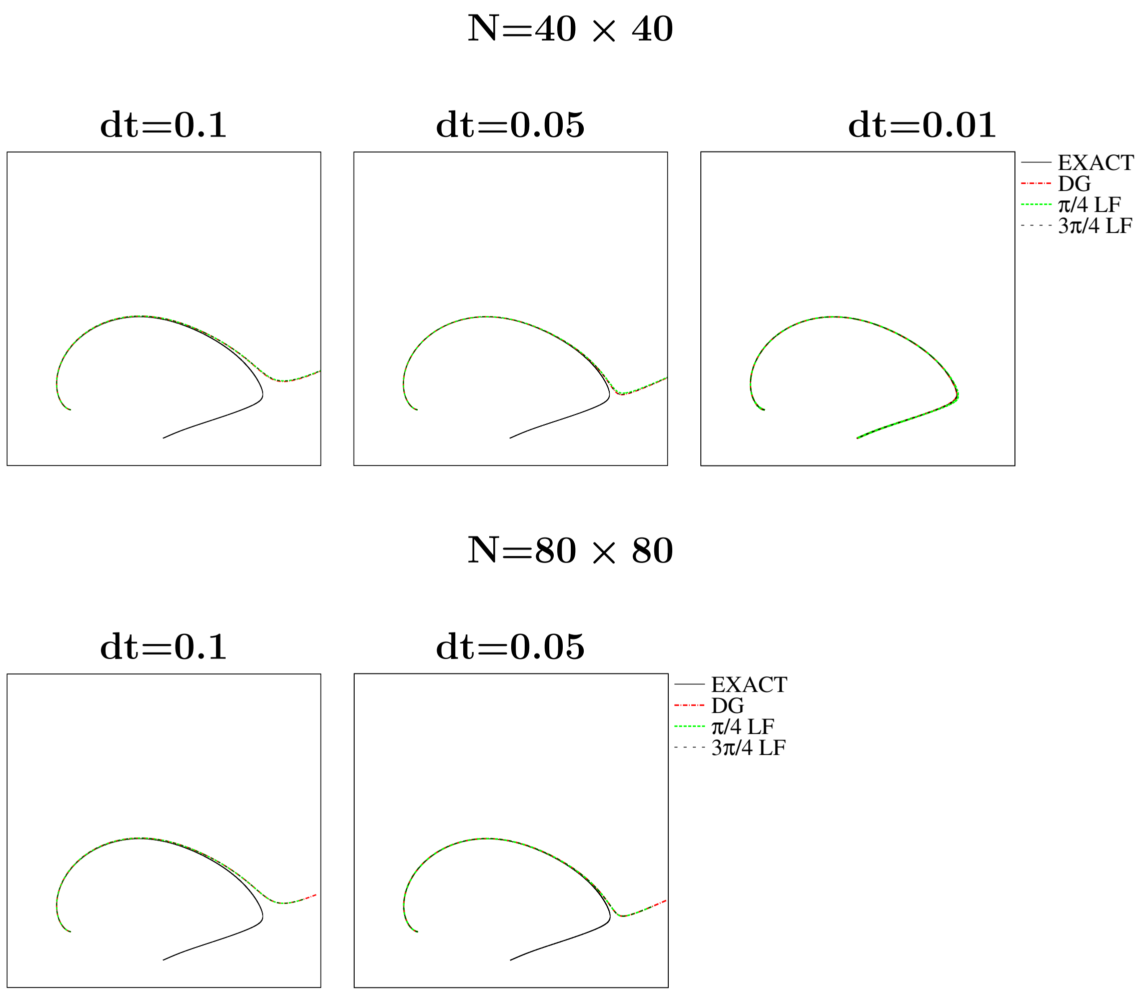
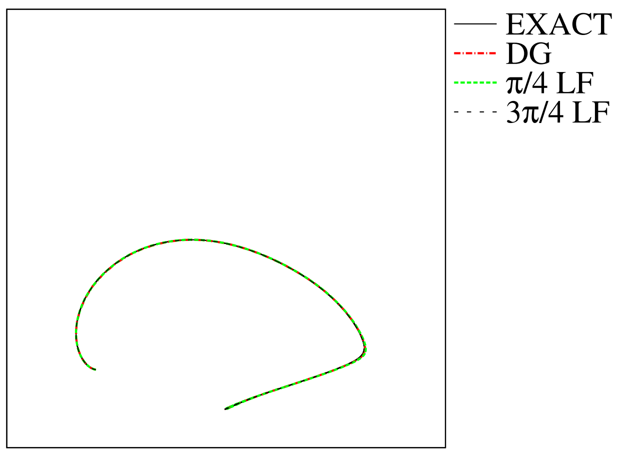


Figure 6.4: Unfiltered and filtered streamlines with seed near a critical point corresponding to the CF1 field (equation (6.11)) applying the RK2 method for three different time steps and two meshes using $\mathbb{P}^2$ polynomials for the DG approximation and $K_\Gamma^{(5,3)}$ kernels.

Figure 6.5: Unfiltered and filtered streamlines with seed near a critical point corresponding to the CF1 field (equation (6.11)) applying the RK3 method with time step $dt = 0.1$ using 40×40 elements and $\mathbb{P}^2$ polynomials for the DG approximation and $K_\Gamma^{(5,3)}$ kernels.



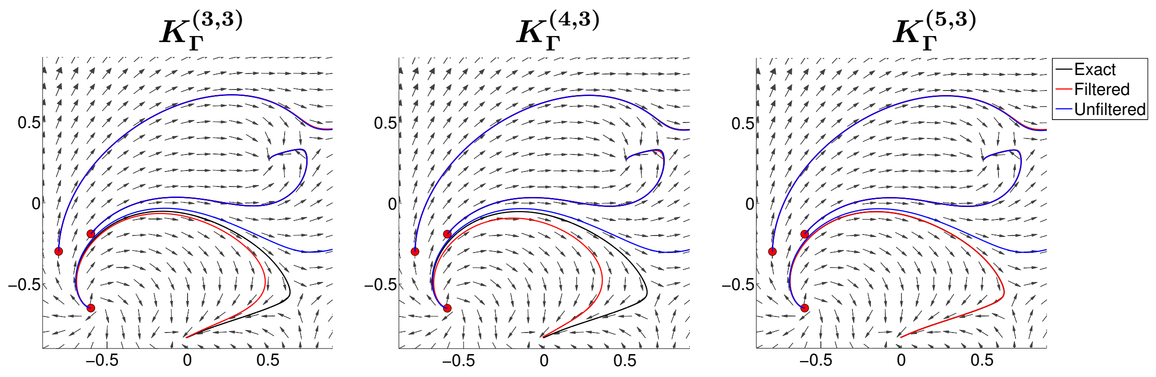


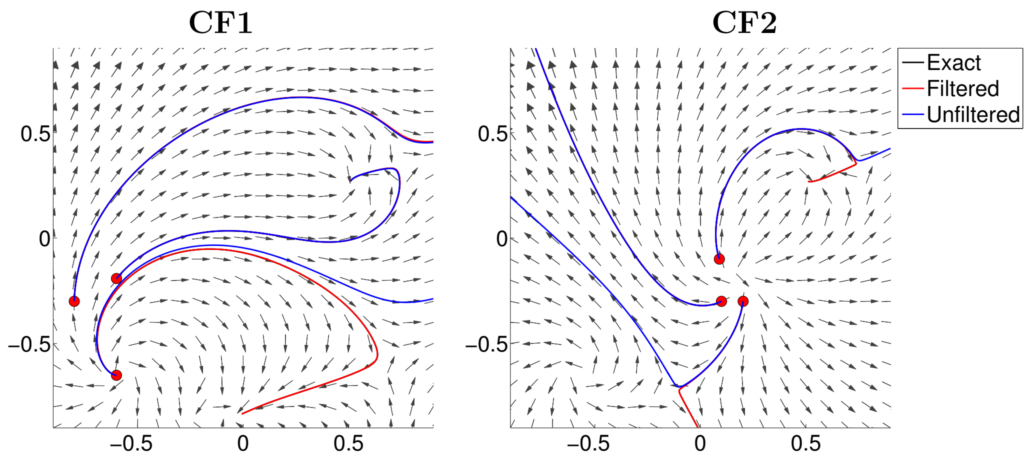
Figure 6.6: Streamlines along the field CF1 (equation (6.11)) over a 40×40 mesh before and after applying $\pi/4$ Line filters with kernels made of varying number (3,4 and 5 respectively) of B-Splines of order 2. The unfiltered solution was computed using $\mathbb{P}^1$ polynomials. The streamlines were computed with a RK2 solver using $dt = 0.01$. The plots where the exact curve cannot be seen is because it overlaps with the filtered streamline.

Higher Order Kernels

The last experiment in this section explores the order and number of splines used to build the kernel. In this case, the unfiltered solution was computed using $\mathbb{P}^1$ polynomials and the filtered solution was computed using the $K_\Gamma^{(5,3)}$ kernel, *i.e.*, a kernel typically employed for DG solutions belonging to the $\mathbb{P}^2$ space. The SIAC kernel, $K_\Gamma^{(2k+1,k+1)}$, is chosen according to the degree of the approximation space in order to ensure that the DG order is preserved. However, this does not represent an upper limit on the number or order of the splines that can be used, only the maximum order of accuracy that can be achieved.

Previously, it was shown that as soon as the degree of the approximation space increased, the unfiltered approximation could produce satisfactory streamlines at relatively large time steps and mesh size (see Figure 6.4 for $N = 40 \times 40$ and $dt = 0.01$). The plots in Figure 6.6 show the resulting streamlines before and after applying three different Line filters built with 3,4 and 5 B-Splines of order 3. Recall that the vector field belongs to the $\mathbb{P}^1$ space. The filtered solutions suggest that the best kernel is the one that uses more splines, *i.e.*, the $K_\Gamma^{(5,3)}$ kernel. Earlier experiments in this section showed that applying the $K_\Gamma^{(3,2)}$ kernel over coarse meshes (see Figure 6.1) produced unsatisfactory streamlines. Figure 6.7 shows the numerical results after the higher order kernel $K_\Gamma^{(5,3)}$ to the complex fields CF1 and CF2 over a 40×40 uniform mesh, using the RK2 method with time step $dt = 0.01$. Observe that both orientations, $\pi/4$ and $3\pi/4$ are able to produce six converging streamlines.

$$\boldsymbol{K_\Gamma^{(5,3)}} : \quad \boldsymbol{\theta = \tfrac{\pi}{4},\ \mu = \sqrt{2}}$$



$$\boldsymbol{K_\Gamma^{(5,3)}} : \quad \boldsymbol{\theta = \tfrac{3\pi}{4},\ \mu = \sqrt{2}}$$

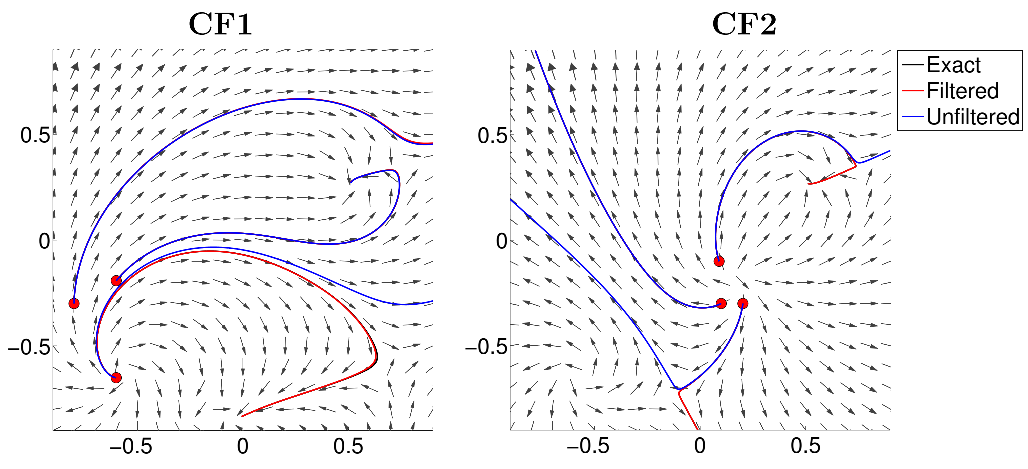


Figure 6.7: Streamline fields before and after applying two symmetric Line Filters with orientations $\theta = \pi/4,\ 3\pi/4$ using a kernel with five B-Splines of order 3 (degree 2). The unfiltered solution was computed using $\mathbb{P}^1$ polynomials using 40×40 elements. The streamlines were obtained through a RK2 solver with time step $dt = 0.01$. The plots where the exact curve cannot be seen is because it overlaps with the filtered streamline.

6.2.2 Line Kernels vs Tensor Product Kernels

From the previous experiments, it was concluded that Line filters should be aligned according to the mesh structure. Furthermore, the results suggested that the $\pi/4$ Line filter gave optimal results. Here, the performance of this filter was compared against the traditional 2D filter aligned with the Cartesian axis. Figure 6.8 shows streamlines corresponding to the CF1 and CF2 fields respectively using a $\mathbb{P}^1$ polynomial basis for the DG approximation and applying the SIAC kernel: $K^{(2k+1,k+2)}$, $k = 1$. The plots for the first field show that the Tensor Product Filter (TPF) handles the coarser meshes better than the Line Filter (LF). On the other hand, as soon as the mesh is refined, both filters have similar behaviour. Figure 6.9 shows streamlines filtered with the higher order Line kernel ($K_{\Gamma}^{(5,3)}$) compared to the Tensor Product kernel ($K^{(3,2)} \otimes K^{(3,2)}$) using a 40×40 mesh. In this case, the Line filter outperforms the tensor product filter. This is clear for the streamline from field CF1 with seed at the critical point. Table 6.2 shows two error estimates; the Maximum Distance was obtained through the formula (6.15) and the Global Error (GE) corresponds to the distance between the points at final time. For the case $N = 40 \times 40$, the higher order filter clearly had the best performance. Observe that the magnitude of the errors in field CF1 for such filter are significantly lower except for the last seed. The other Line filter produced similar results than the Tensor Product filter except for the first streamline (seed (-0.6,-0.651)). The second field shows similar results: the higher order filter has the greatest error reduction, especially for the 40×40 mesh. In all cases, the filters successfully increase the accuracy of the curves compared to the original streamlines.

The results in Table 6.3 show the computational times taken by each filter to post-process each streamline. The difference between the computational times between both meshes is not very large. Actually, for some streamlines, using a finer mesh resulted in faster computations as it can be seen for the times taken by each filer to post-process the last seed in both fields. On the other hand, the difference between the elapsed times taken by the Tensor Product filter compared to the Line filters is very large. Line filters use a one-dimensional convolution and this results in great reduction of the computational costs compared to 2D filters. The higher order Line filter requires longer simulation times compared to the other Line filter but still remain very low compared to Tensor Product filter. These type of filters increase the support since they use more splines and are of higher order. Therefore, they are less robust than the lower order Line filter in terms of the area of the domain where they can be employed. However, the results show that they are most effective when reducing the error. Whenever possible, higher-order filters should be implemented.

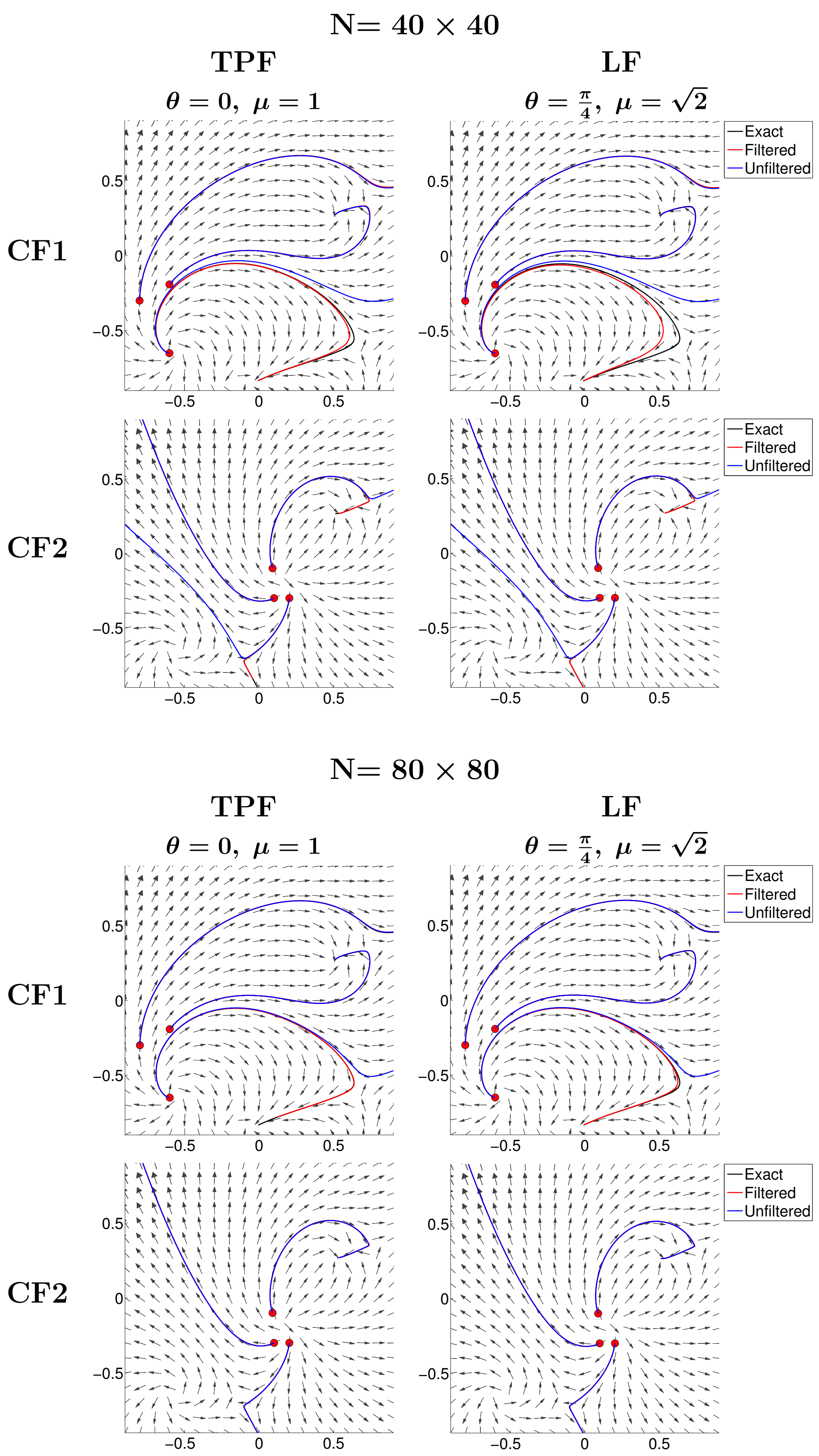


Figure 6.8: Comparison of the performance between the Cartesian axis Tensor Product Filter (TPF) and the $\pi/4$-Line Filter (LF) over two velocity fields (CF1, CF2) and two different meshes ($N = 40 \times 40$ and $N = 80 \times 80$) using $K^{(3,2)}$ kernels over a $\mathbb{P}^1$ DG solution. The cases where the black curve (exact) cannot be seen is because it overlaps with the filtered solution.

CF1								
	Unfiltered		$\boldsymbol{K^{(3,2)} \otimes K^{(3,2)}}$		$\boldsymbol{K_{\Gamma}^{(3,2)}}$		$\boldsymbol{K_{\Gamma}^{(5,3)}}$	
			$\boldsymbol{\theta = 0,\ \mu = 1}$		$\boldsymbol{\theta = \frac{\pi}{4},\ \mu = \sqrt{2}}$			
Seed	MD	GE	MD	GE	MD	GE	MD	GE
N= 40 $\times$ 40								
(-.6,-.651)	DIVERGED		9.1e-2	1.5e-2	2.1e-1	3.0e-2	3.3e-4	1.9e-4
(-.6,-.192)	4.0e-3	9.7e-4	3.8e-3	5.3e-5	6.1e-3	1.6e-4	2.2e-4	9.5e-6
(-.8,-.3)	4.1e-2	4.1e-2	4.1e-2	4.1e-2	3.8e-2	3.8e-2	5.4e-2	5.4e-2
N= 80 $\times$ 80								
(-.6,-.651)	DIVERGED		3.3e-3	5.8e-4	4.3e-2	7.3e-3	1.7e-3	1.1e-3
(-.6,-.192)	2.6e-3	1.2e-4	1.5e-5	4.2e-6	5.5e-4	9.6e-6	1.3e-4	7.5e-7
(-.8,-.3)	3.6e-2	3.6e-2	2.7e-2	2.7e-2	2.7e-2	2.7e-2	2.1e-2	2.1e-2
CF2								
	Unfiltered		$\boldsymbol{K^{(3,2)} \otimes K^{(3,2)}}$		$\boldsymbol{K_{\Gamma}^{(3,2)}}$		$\boldsymbol{K_{\Gamma}^{(5,3)}}$	
			$\boldsymbol{\theta = 0,\ \mu = 1}$		$\boldsymbol{\theta = \frac{\pi}{4},\ \mu = \sqrt{2}}$			
Seed	MD	GE	MD	GE	MD	GE	MD	GE
N= 40 $\times$ 40								
(0.202,-.3)	DIVERGED		1.8e-1	1.8e-1	1.8e-1	1.8e-1	3.8e-2	3.8e-2
(-.09,-.1)	DIVERGED		3.7e-2	1.1e-2	2.2e-2	7.0e-3	4.8e-3	4.8e-3
(.1,-.3)	5.4e-3	5.4e-3	7.7e-3	7.7e-3	7.1e-3	7.1e-3	6.9e-3	6.9e-3
N= 80 $\times$ 80								
(0.202,-.3)	2.2e-1	2.2e-1	9.5e-3	9.5e-3	1.2e-2	1.2e-2	1.0e-4	1.0e-4
(-.09,-.1)	7.6e-2	1.9e-2	6.1e-5	2.3e-5	7.0e-3	2.5e-3	2.9e-3	2.9e-3
(.1,-.3)	7.5e-3	7.5e-3	6.8e-3	6.8e-3	6.8e-3	6.8e-3	4.2e-3	4.2e-3

Table 6.2: Maximum Distance (MD) computed using equation (6.15) and Global Error (GE) corresponding to the distance between the points at final time comparing unfiltered and filtered streamlines along the fields CF1 and CF2 (equations (6.11) and (6.13)) using a Tensor Product and two Line filters over two different meshes ($N = 40 \times 40$ and $N = 80 \times 80$). The DG approximation was computed using $\mathbb{P}^1$ polynomials and all the streamlines were obtained through a RK2 method with time step $dt = 0.01$.

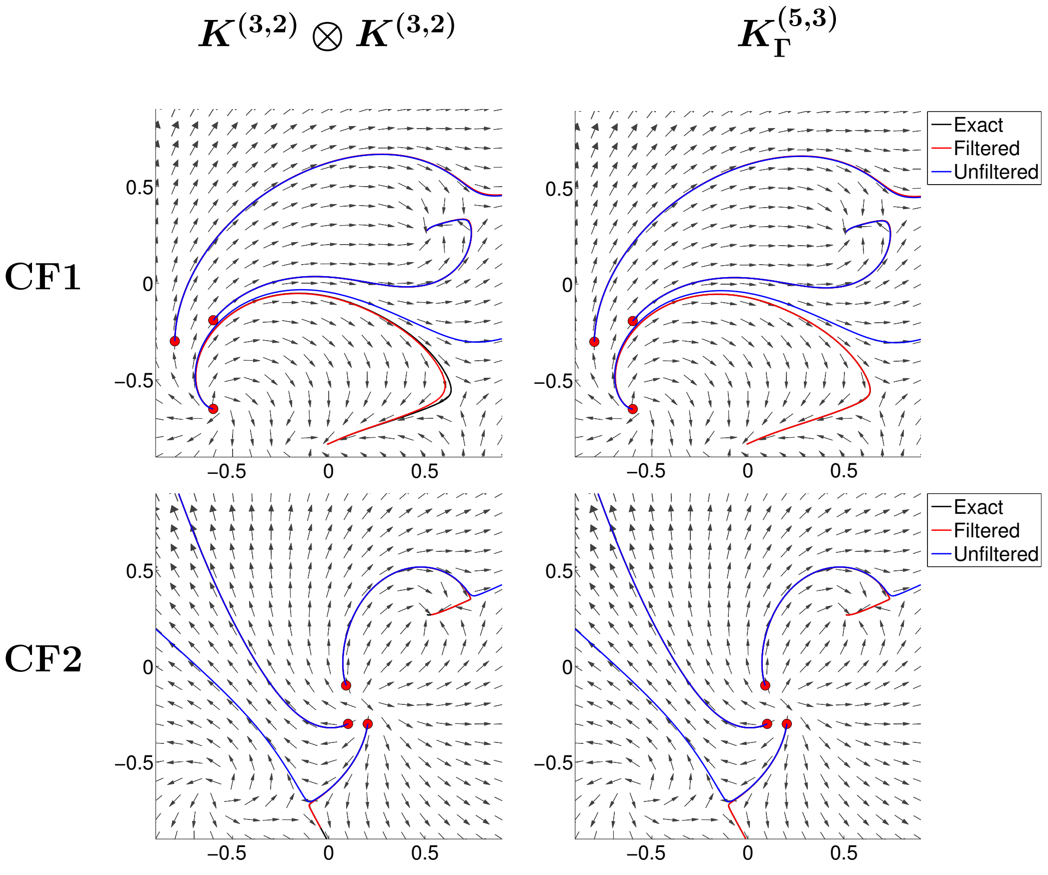


Figure 6.9: Streamlines before and after applying the 2D Cartesian axis aligned Filter $K^{(3,2)} \otimes K^{(3,2)}$ with the usual scaling $H = h$ and the $\pi/4$-Line Filter $K_\Gamma^{(5,3)}$ with scaling $H = \sqrt{2}h,$ h being the DG mesh size corresponding to the 40×40 elements mesh.

	$K^{(3,2)} \otimes K^{(3,2)}$		$K_{\Gamma}^{(3,2)}$		$K_{\Gamma}^{(5,3)}$	
Seed	40×40	80×80	40×40	80×80	40×40	80×80
CF1						
(-.6,-.651)	2504.5	2595.1	446.6	461.5	795.7	795.0
(-.6,-.192)	2066.3	2098.1	376.8	390.4	638.2	666.6
(-.8,-.3)	131.5	104.1	30.9	19.1	47.4	32.9
CF2						
(.202,-.3)	1329.4	1452.1	237.0	242.8	400.9	415.7
(-.09,-.1)	2434.4	2368.7	442.1	433.8	741.3	741.7
(.1,-.3)	776.7	509.2	150.9	93.1	243.5	155.3

Table 6.3: Computational times (seconds) taken to post-process each streamline for a Tensor Product filter and two $\pi/4$-Line filters of order 2 and 3 respectively.

6.3 Boundary Filters

As discussed earlier, symmetric filters may not be possible to implement everywhere in the domain. For example, for solutions over domains that do not assume periodic boundary conditions, such filters can not be implemented near the boundaries. Therefore, this section investigates one sided filters. In addition to boundary line filters and revisiting the idea from [64], filters along the streamline were also implemented. That is, boundary filters whose support spreads downstream along the curve. The experiments were performed along the same velocity fields than the symmetric kernels (equations (6.11) and (6.13)) and this study begins by investigating the first kind of filters: boundary line filters.

6.3.1 Boundary Line Filters

The following experiments were performed by implementing purely one-sided (boundary) filters, assuming every point was a boundary point. This means that the kernel has its support shifted totally towards one side and it is expected to produce the worst results (see Chapter 2). This situation is very unlikely to happen for all the post-processing points along a streamline curve and in practice, these filters should be position-dependent, allowing a transition towards symmetric kernels whenever possible. Nevertheless, in order to understand how much accuracy is lost, *all* the post-processing points were computed using a purely one-sided kernel.

The numerical results from the previous section together with the theoretical error estimates (Theorem 4.3.1) indicate that the filters should be aligned with the mesh, either at $\pi/4$ or $3\pi/4$. Consider a XLi left sided filter (see Figure 2.3). Then, there are four possible rotations: $\pi/4$, $3\pi/4$ and the opposite directions, *i.e.*, $5\pi/4$ and $7\pi/4$. Notice that these orientations could also be identified with right sided filters along the previous orientations. Figures 6.10 and 6.11 show the resulting streamlines before and after applying these filters over a DG solution using $\mathbb{P}^1$ polynomials and implementing the XLi kernel corresponding to $r = 2$, $\ell = 2$ in equation (2.28). The unfiltered and filtered streamlines were obtained through the RK2 solver with $dt = 0.01$. From these results, it is difficult to conclude whether there is an optimal orientation. The $3\pi/4$ orientation seems to produce the worst results since it is the only case where for the field CF1, the streamline starting at the critical point still diverges after mesh refinement. On the other hand, the $7\pi/4$ orientation produced highly accurate streamlines for that field. However, this filter had the worst performance for the 40×40 mesh and actually produced a diverging streamline which already converged towards the true solution before filtering. The $5\pi/4$ orientation lead to the best results for the second field, CF2. This filter produced the most accurate streamlines for the coarser mesh.

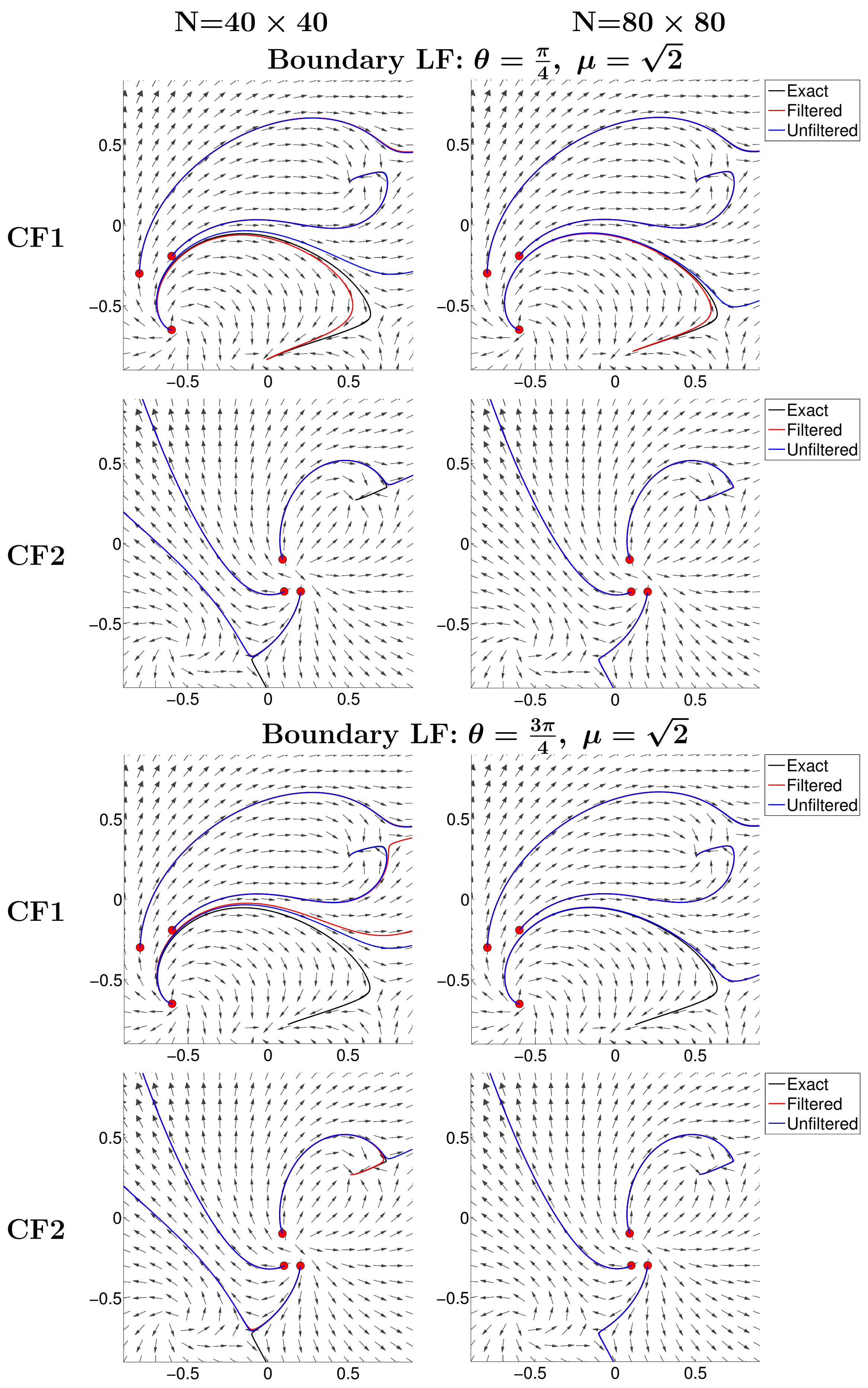


Figure 6.10: Streamlines along the velocity fields CF1 and CF2 (equations (6.11) and (6.13)) for two different meshes ($N = 40 \times 40$ and $N = 80 \times 80$) before and after applying boundary filters with orientation $\theta = \pi/4$ and $\theta = 3\pi/4$.

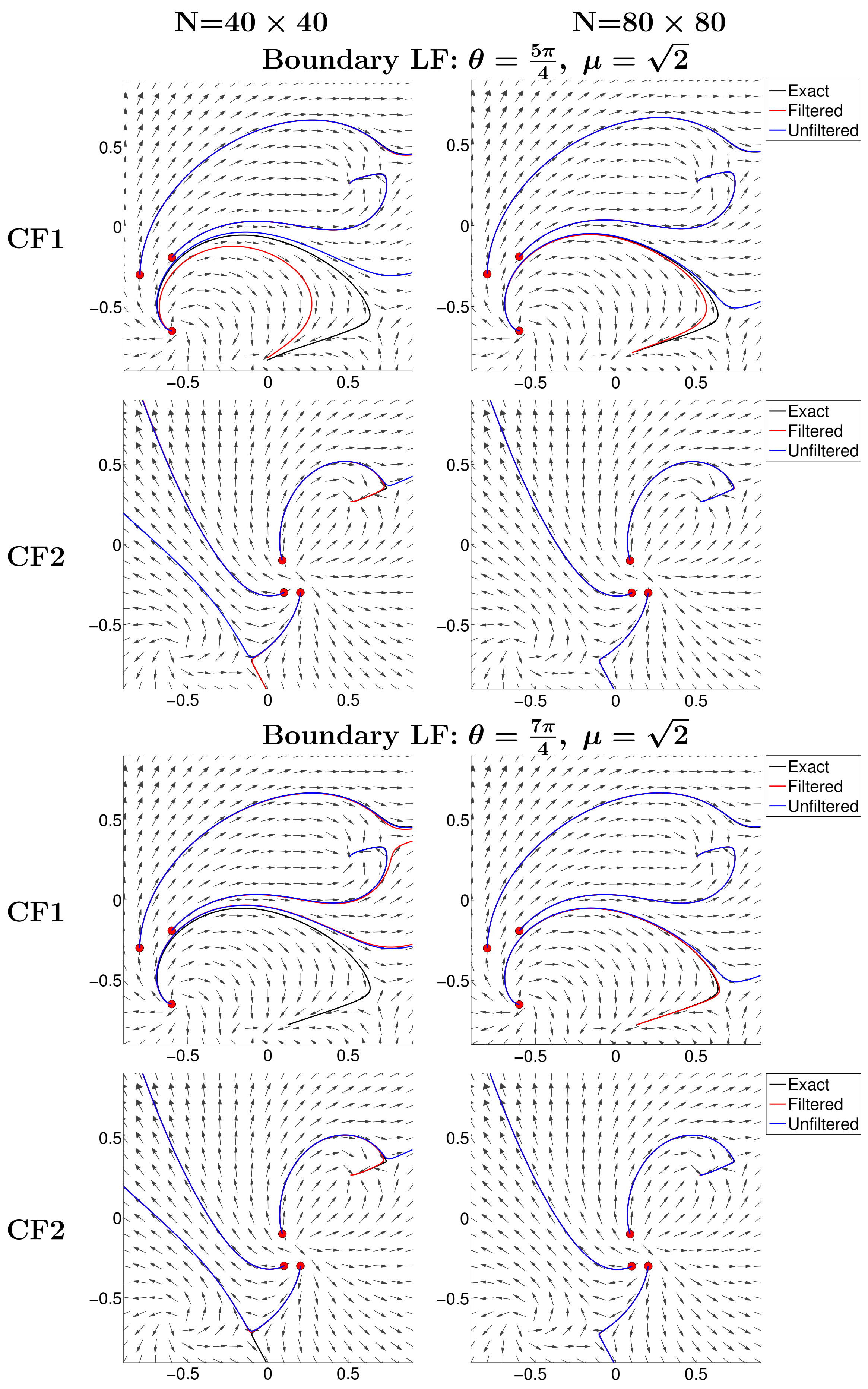


Figure 6.11: Streamlines along the velocity fields CF1 and CF2 ($N = 40 \times 40$ and $N = 80 \times 80$) before and after applying boundary filters with orientation $\theta = \pi/4$ and $\theta = 3\pi/4$.

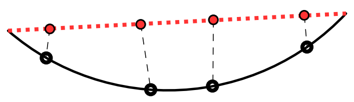


Figure 6.12: Location of the quadrature points using a linear reconstruction (red) compared to their exact location on the curve.

6.3.2 Filtering Along the Streamline

Filtering along the streamline curve implies spreading the kernel support along the curve. When using explicit ODE solvers, *i.e.*, schemes where each iteration step uses information only from previously computed points, a purely right sided filter should be applied. The implementation is very similar to Line-filters. In fact, a line filter could be seen as a particular case, when the streamline curves are straight lines. Let

$$\Gamma(t) = (\Gamma_x(t), \Gamma_y(t)), \quad \text{with } \Gamma(0) = (\overline{x, y}), \tag{6.16}$$

be the parametrization of the streamline by the arc-length parameter. Since this curve is unknown, it is reconstructed by interpolation of previously computed points. The filtering convolution is then given by:

$$u_h^\star(\overline{x, y}) = K \star u_h(\Gamma(t_0)) = \frac{1}{H}\int_{-\infty}^{\infty} K\left(\frac{-t}{H}\right) u_h\left(\Gamma(t)\right) \|\Gamma'(t)\| dt, \tag{6.17}$$

and Algorithm 4 illustrates how to solve this equation numerically. There are two functions in this algorithm, **get arc length(s_i, s_{i+1})** and **find kernel break coordinates (s_i, s_{i+1}, *local arc*)** , which depend on the type of curve reconstruction. Here, two types of interpolation will be discussed which were used to approximate the curve between every two consecutive points.

Curve Reconstruction

The curve from equation (6.16) consists of a union of curves given by streamline points. For simplicity and efficiency, the best way to reconstruct the streamline would be using linear interpolation. However, depending on the trajectory of the streamline, this could result in a excessive low order approximation. Since the convolution is solved using Gaussian Integration, the quadrature points where the kernel and field are evaluated can be far from the actual streamline as shown in Figure 6.12). Alternatively, the curve could be reconstructed by cubic interpolants using Hermite polynomials. These particular type of cubic spline curves use information from the derivative of the function. In a streamline, the values are readily available since they are used by the ODE solver when evaluating the velocity field. These points can be identified as interpolation nodes.

Let $x_1 < \ldots < x_N$ be an ordered knot sequence such that for a given function g, the following pairs

$$\{g(x_i),\ g'(x_i)\}_{i=1}^{N}$$

Algorithm 4 Arc length Convolution

```
$\{s_i\} \leftarrow$ collect sufficient streamline points to fit kernel support
$N \leftarrow$ Total streamline points
$scaling \leftarrow$ Get unscaled kernel scaling
$total\ arc = 0$
$break\ no = 1$
$Integral = 0$
$i = 0$
**while** $break\ no < total\ kernel\ breaks$ **do**
    $arc\ length =$ get arc length$(s_i, s_{i+1})$
    $++i$
    **if** $total\ arc + arc\ length > scaling$ **then**
        $local\ arc = (scaling - total\ arc)/(arc\ length)$
        $s_i \leftarrow$ find kernel break coordinates$(s_i, s_{i+1}, local\ arc)$
        $points\ matrix \leftarrow$ collect break
        $total\ arc = 0$
        $++break\ no$
    **else**
        $local\ arc+ = arc\ length$
    **end if**
    **while** $ID(s_i)! = ID(s_{i+1})$ **do**
        $s = \overline{(s_i, s_{i+1})}$
        **do**
            $e \leftarrow ID(s_i) \rightarrow$ get edge
        **while** $s \cap e = \emptyset$
        $points\ matrix \leftarrow$ get intersection point(s,e)
    **end while**
**end while**
**for** $p = 0 : dim(\ points\ matrix) - 1$ **do**
    $Integral+ =$ evaluate convolution$(points\ matrix(p), points\ matrix(p+1))$
**end for**
```

are known The cubic Hermite polynomial $H_3(x)$ that interpolates the function g is a combination of local cubic polynomials,

$$\{P_i(x), x \in [x_i, x_{i+1}],\ i = 1, \dots, N-1\}$$

satisfying the following two conditions:

$$\begin{aligned} P_i(x_i) = g(x_i),\ P_i(x_{i+1}) = g(x_{i+1}), \\ P_i'(x_i) = g'(x_i),\ P_i'(x_{i+1}) = g'(x_{i+1}). \end{aligned}$$

Each polynomial $P_i(x)$ can be obtained from its Newton form:

$$\begin{aligned} P_i(x) =& P_i(x_i) + (x - x_i)[x_i, x_i]P_i + (x - x_i)^2[x_i, x_i, x_{i+1}]P_i \\ &+ (x - x_i)^2(x - x_{i+1})[x_i, x_i, x_{i+1}, x_{i+1}]P_i. \end{aligned}$$

The following table gives the divided differences when these polynomials are applied to streamlines, identifying y with g and (u, v) with the velocity field.

	$[\ ]P_i$	$[\ ,\ ]P_i$	$[\ ,\ ,\ ]P_i$	$[\ ,\ ,\ ,\ ]P_i$
x_i	y_i			
		$v(x_i, y_i)$		
x_i	y_i		$\frac{\frac{y_{i+1}-y_i}{x_{i+1}-x_i} - v(x_i,y_i)}{x_{i+1}-x_i}$	
		$\frac{y_{i+1}-y_i}{x_{i+1}-x_i}$		$\frac{v(x_{i+1},y_{i+1}) + v(x_i,y_i) - 2\left(\frac{y_{i+1}-y_i}{x_{i+1}-x_i}\right)}{(x_{i+1}-x_i)^2}$
x_{i+1}	y_{i+1}		$\frac{v(x_{i+1},y_{i+1}) - \frac{y_{i+1}-y_i}{x_{i+1}-x_i}}{x_{i+1}-x_i}$	
		$v(x_{i+1}, y_{i+1})$		
y_{i+1}	y_{i+1}			

Assuming that $g \in C^{(4)}[x_1, x_N]$, the interpolation error of a Cubic Hermite Polynomial is bounded by

$$|g(x) - H_3(x)| \leq \left(\frac{x_{i+1} - x_i}{2}\right)^4 \max_{x_i \leq x \leq xi+1} \frac{|g^{(4)}(\xi_x)|}{4!}$$

More information on Hermite interpolation can be found in [17, Ch. 4].

Note 6.3.1. *The implementation given in Algorithm 4 computes intersection points assuming a linear parametrization of the streamline curve. The intersection algorithm described in chapter 5 assumes straight segments so the implementation should change for higher degree curves.*

Note 6.3.2. *The inter-points of the ODE solver are not included during curve reconstruction. For example, the two stage RK2 solver:*

$$k_1 = f(x_n, y_n) \tag{6.18}$$

$$k_2 = f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right) \tag{6.19}$$

$$y_{n+1} = y_n + hk_2 \tag{6.20}$$

requires evaluating the field at the point $y_n + h\frac{k_1}{2}$ which does not necessarily belong to the streamline. In practice, although that point is also filtered, curve reconstruction is done using the nodes y_n and y_{n+1} alone.

Numerical Results

The experiments performed on these filters included different types of curve reconstruction and kernel scalings. Since the kernel support follows the streamline curve, the scaling is no longer required to be $H = \sqrt{2}h$. Therefore, the simulations were also carried out using smaller scaling corresponding to $H = h$, h being the DG mesh size. The unfiltered streamline was computed using $\mathbb{P}^1$ polynomials and all the filters consisted of a XLi kernel made of 3 B-Splines and a general Spline of order 2, *i.e.*, equation (2.28) with $r = 2$, $\ell = 1$. All the streamlines were computed with the RK2 solver with time step $dt = 0.01$.

Figures 6.13 and 6.14 show streamlines before and after applying these filters using Hermite and Linear interpolants. Just like in previous experiments, the performance of these filters strongly depends on the mesh resolution. Observe how for the first field (Figure 6.13), when using $N = 40 \times 40$ elements, all the filtered streamlines starting at the critical point diverge from the exact curve and follow the path of the unfiltered streamline. On the other hand, increasing the order of the curve reconstruction seems to improve the solutions: in both fields, the filter implemented with Hermite interpolation produced more accurate streamlines. Regarding the scaling, the larger value, $\mu = \sqrt{2}$ produced better results with the exception of the second field (CF2) using the mesh $N = 40 \times 40$.

The previous results suggested that Hermite interpolation should be used over linear interpolation. On the other hand, this type of interpolation increases the computational costs. This is reflected in Table 6.5, which shows the computational times taken by this type of filter. Observe how using Linear interpolation implies significantly lower values than Hermite interpolation. However, the time step employed by the ODE solver was relatively large ($dt = 0.01$) so this types of curve reconstruction should be applied since it provides higher accurate streamlines.

In the final study, filtering along the streamline using Hermite interpolation was compared to Boundary Line filtering oriented along the flow direction and its tangent. Figure 6.15 shows the results from implementing these filters along the velocity field CF1 and CF2. The plots suggest that filtering along the streamline yields to better results than when applying a Boundary Line filter oriented along the same direction. On the other hand, the tangent flow aligned filter had a better performance. Observe how for the case $N = 80 \times 80$ and CF1, this filter is able to converge towards the exact streamline for the one starting at the critical point whereas the flow aligned filter diverges alongside the unfiltered curve. Figure 6.16 compares filtering along the streamline with the Boundary Line Filters from section 6.3 that gave best results, *i.e.*, the $5\pi/4$ and $7\pi/4$ rotations. For the finest mesh, the filter along the streamline behaves

	Unfiltered		**$\frac{\pi}{4}$ SLF**		**$\perp$flow-BLF**		**$\frac{7\pi}{4}$ BLF**		**FAS-H**	
Seed	MD	FD	MD	FD	MD	FD	MD	FD	MD	FD
CF1										
(-.6,-.651)	DIVERGED		4.3e-2	7.3e-3	3.7e-2	1.6e-2	3.9e-2	1.6e-2	2.5e-3	1.6e-3
(-.6,-.192)	2.6e-3	1.2e-4	5.5e-4	9.6e-6	1.9e-3	1.5e-4	6.0e-4	2.2e-4	6.9e-3	2.6e-3
(-.8,-.3)	3.6e-2	3.6e-2	2.7e-2	2.7e-2	1.7e-2	1.7e-2	2.7e-2	2.7e-2	3.0e-2	3.0e-2
CF1										
(0.202,-.3)	2.2e-1	2.2e-1	1.2e-2	1.2e-2	7.0e-2	7.0e-2	1.9e-1	1.9e-1	2.5e-1	2.5e-1
(-.09,-.1)	7.6e-2	1.9e-2	7.0e-3	2.5e-3	2.1e-2	2.1e-2	1.7e-2	1.7e-2	2.9e-2	2.9e-2
(.1,-.3)	7.5e-3	7.5e-3	6.8e-3	6.8e-3	7.2e-3	7.2e-3	2.7e-3	2.7e-3	3.1e-2	3.1e-2

Table 6.4: Maximum Distance (MD) taken as the greatest point distance from all the iterations and Global Errors (GE) measuring the point distance at final time comparing unfiltered and filtered streamlines along the fields CF1 and CF2 (equations (6.11) and (6.13)) using a 80×80 mesh. The filtered solutions were obtained through a Symmetric Line Filter (SLF), two Boundary Line Filters (BLFs) oriented along the $7\pi/4$ direction and tangent to the flow respectively and a Filter Along the Streamline using Hermite interpolation. The streamlines were computed using the RK2 solver with $dt = 0.01$.

similarly to the $7\pi/4$ Boundary Line filter, outperforming the $5\pi/4$ rotation. On the other hand, applying the filters on the mesh made of 40×40 suggests that the latter rotation, $5\pi/4$ gives the best results. Finally, Table 6.4 shows the errors comparing these filters with the symmetric kernel for the mesh $N = 80 \times 80$. The values show how although in general, applying symmetric filters leads to better post-processing, accuracy enhancement is still possible for the one-sided filters.

	Linear Interpol.		**Hermite Interpol.**	
Seed	40×40	80×80	40×40	80×80
CF1				
(-.6,-.651)	137.7	179.6	295.0	798.3
(-.6,-.192)	290.9	241.1	960.3	731.7
(-.8,-.3)	17.8	18.0	34.3	36.1
CF2				
(.202,-.3)	77.7	65.9	416.8	321.5
(-.09,-.1)	347.0	280.8	1124.4	828.9
(.1,-.3)	17.57	28.2	47.5	81.7

Table 6.5: Computational times (seconds) taken to post-process each streamline using a filter along the streamline with Hermite and linear interpolation respectively.

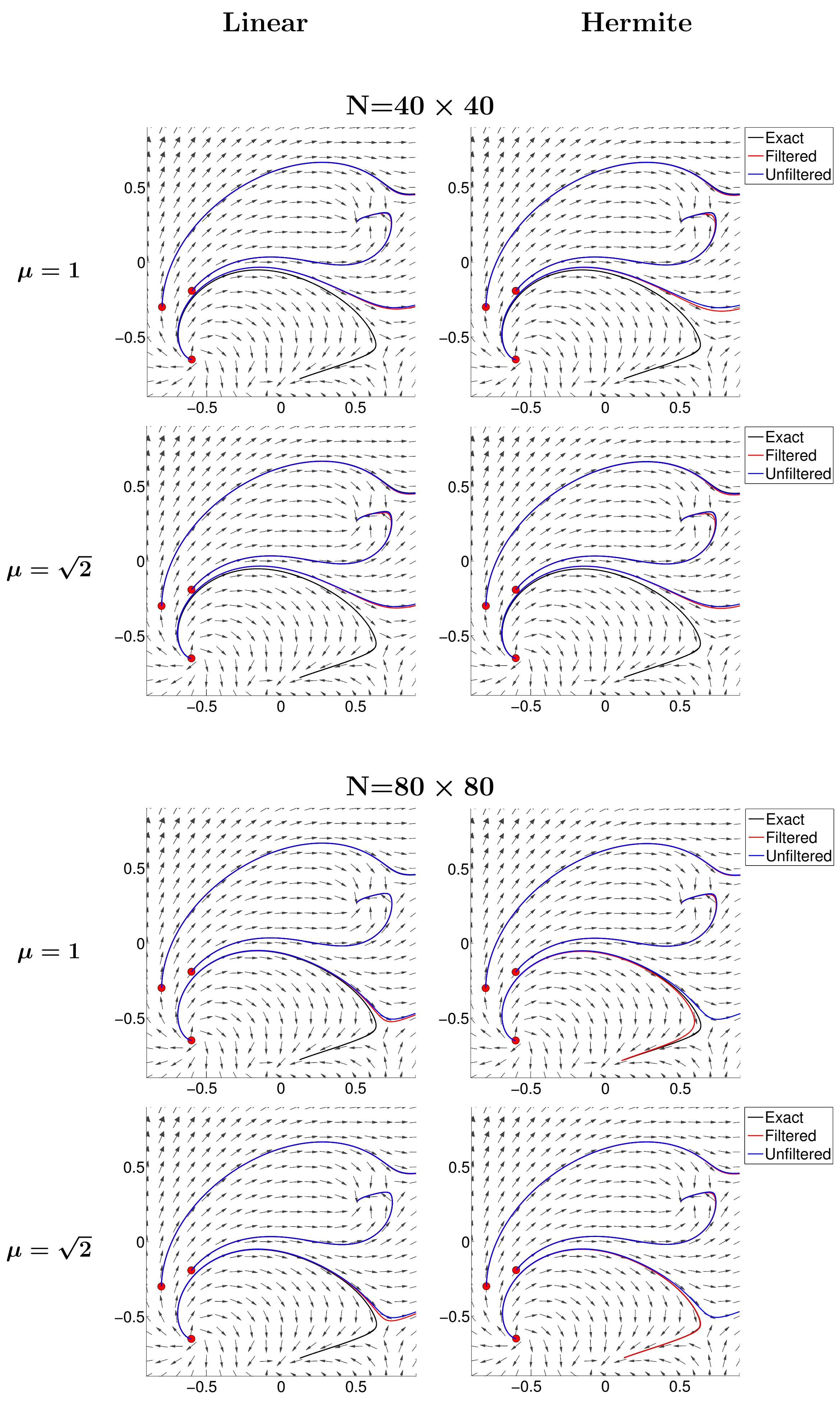


Figure 6.13: Streamlines along the first velocity fields (CF1, equations (6.11)) for two different meshes before and after filtering along the streamlines using linear and Hermite interpolation for the curve reconstruction.

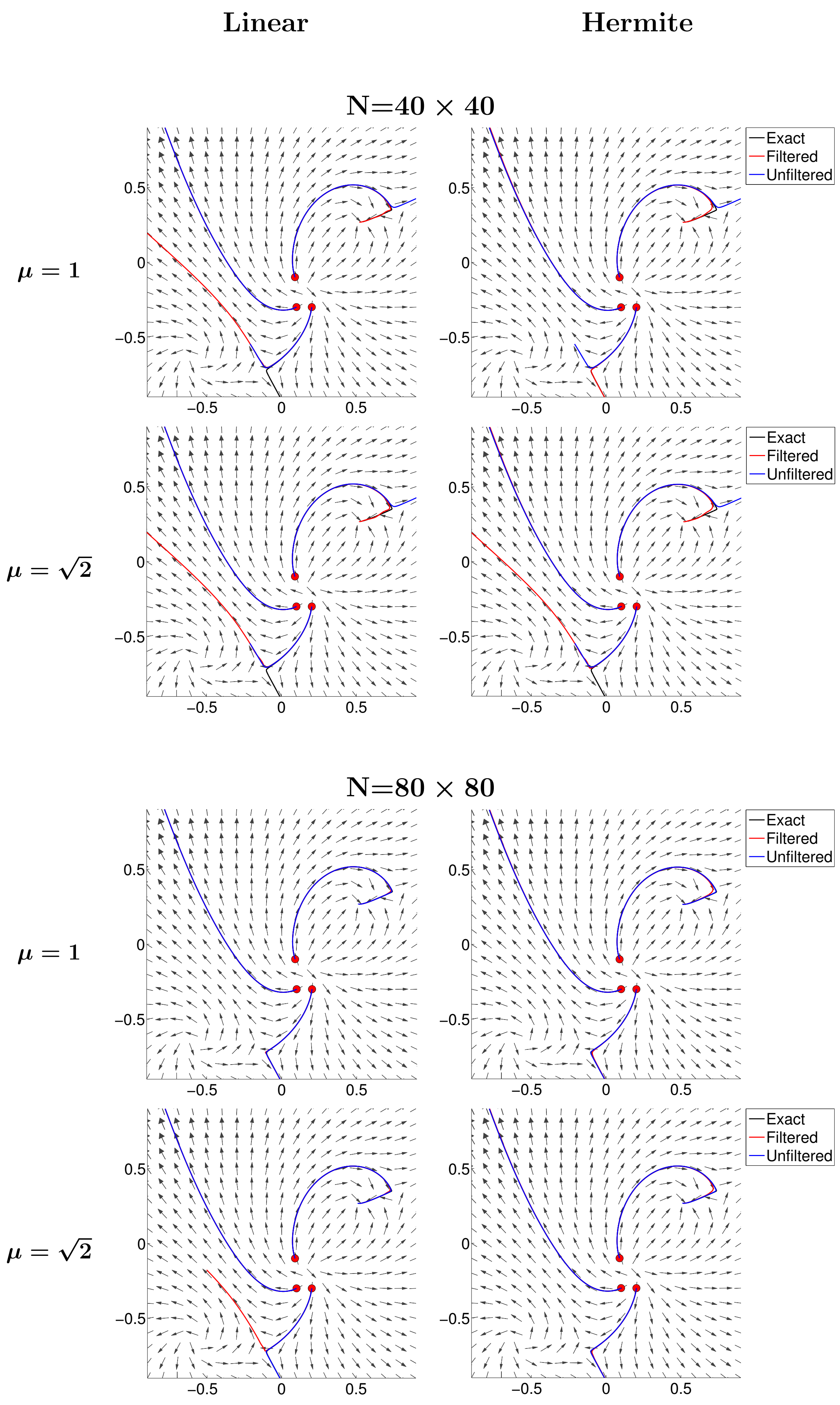


Figure 6.14: Streamlines along the second velocity fields (CF2, equations (6.13)) for two different meshes before and after filtering along the streamlines using linear and Hermite interpolation for the curve reconstruction.

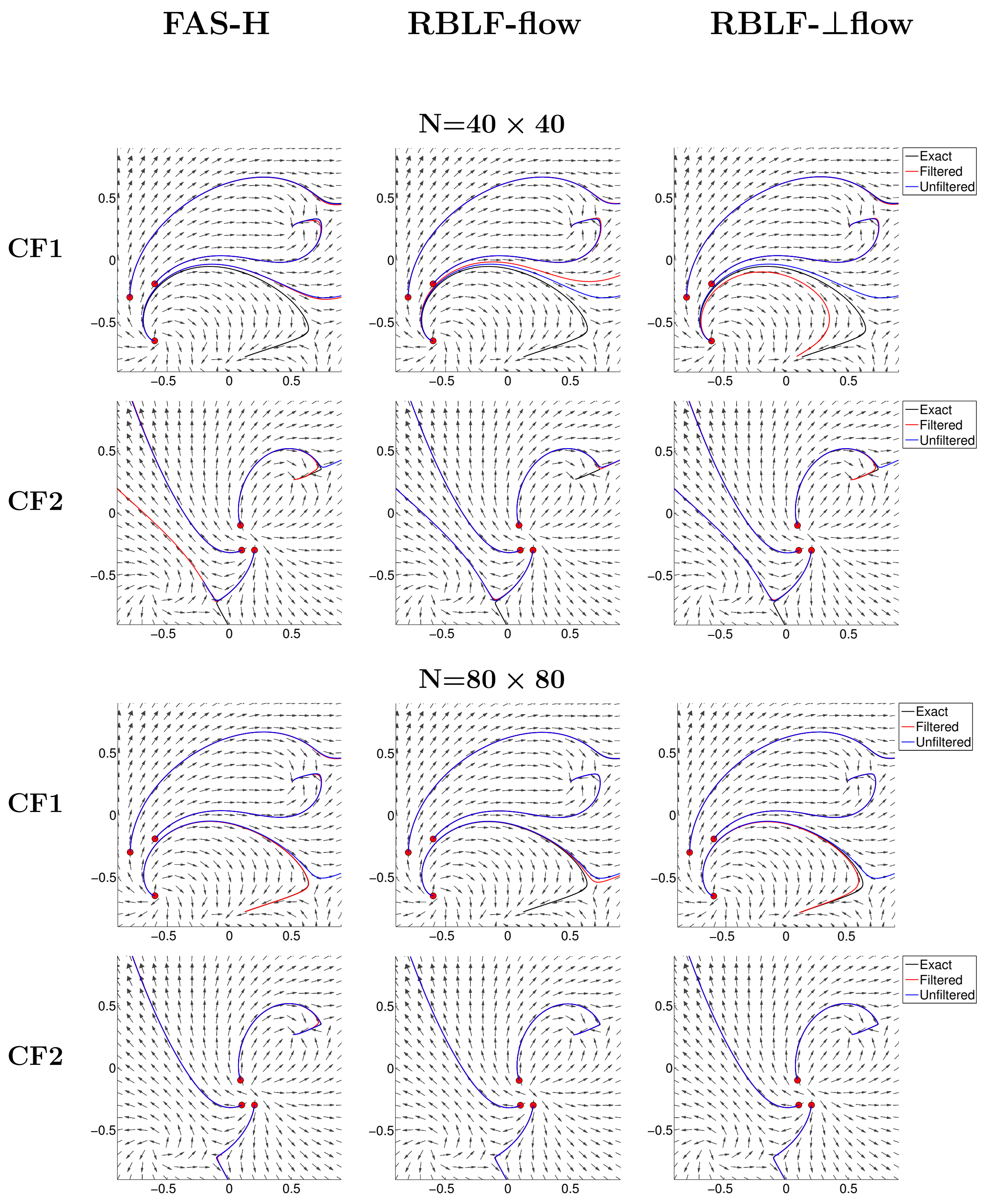


Figure 6.15: Streamlines along two velocity fields (CF1 and CF2) before and after applying a Filter Along the Streamline using Hermite reconstruction (FAS-H) compared to Right Boundary Line Filters (RBLFs) aligned with the flow (middle) and tangent to the flow (right). All the filters were computed with the scaling $H = \sqrt{2}h$.

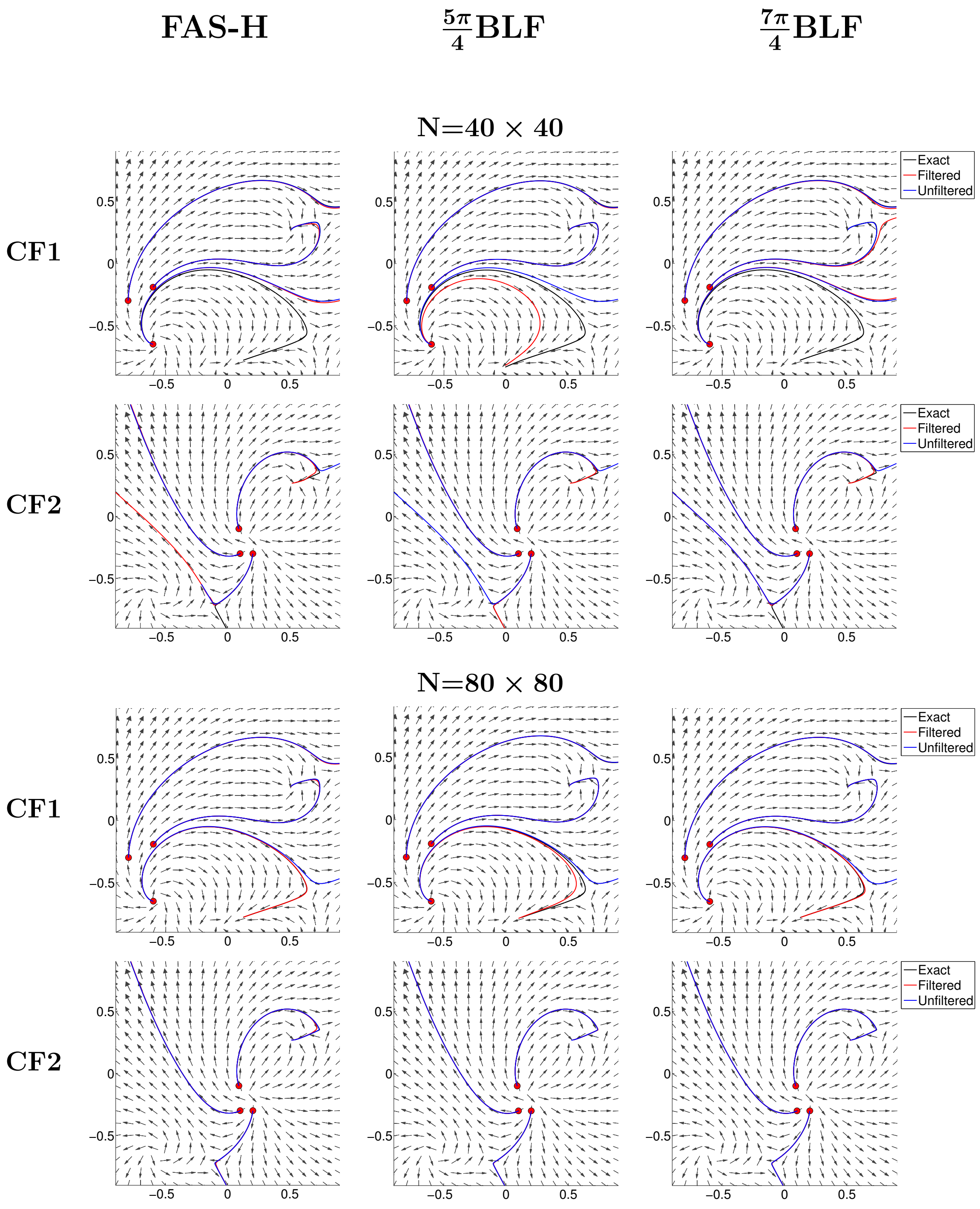


Figure 6.16: Streamlines along two velocity fields (CF1 and CF2) before and after applying Filter Along the Streamline using Hermite reconstruction (FAS-H) compared to Boundary Line Filters (BLFs) using $5\pi/4$ and $7\pi/4$ orientations. All the filters were computed with the scaling $H = \sqrt{2}h$.

6.4 Closest Point Approach

The error estimates given in Tables 6.1, 6.2 and 6.4 were taken as the maximum difference between the exact and approximated streamline at each iteration. For unsteady flows, the distance between the points at each iteration (time step) is important since the solution changes with time. However, for steady flows, alternative error measurements could be taken. Figure 6.17 shows a streamline where the error at final time would be very large. However, if the final time for computing the filtered streamline was extended, an overlap between the curves overlap as shown in Figure 6.18. The plots in this figure were computed using the domain boundaries as the stop criteria, *i.e.*, streamline points were computed until they exited $\Omega = [-1, 1] \times [-1, 1]$. Based on this approach, the following estimate was proposed in order to quantify the error between the streamline curves.

Denote by A and B the exact and approximated curves. Given a point $b \in B$, define the distance from the point to the other curve by:

$$d(b, A) = \inf_{a \in A} d(a, b),$$

where $d(a, b)$ is the Euclidean distance. The following error estimate:

$$e_B(A) = \sup_{b \in B} d(b, A), \tag{6.21}$$

gives the maximum distance between the streamlines, ignoring the instant distance between the points. Although this measurement is not suitable for time dependent flows, for steady flows it gives a more accurate estimate of the actual distance between the curves. The idea is based on the Hausdorff distance, which is defined by:

$$d(B, A) = \sup \{d_B(A), d_A(B)\}, \tag{6.22}$$

except that is only computed from one curve to the other, *i.e.* imposing:

$$d_A(B) \leq d_B(A).$$

Otherwise, applying the Hausdorff distance of the streamlines shown in Figure 6.17 would give the distance between the curves at final time. Table 6.6 shows error estimates before and after filtering with a line filter and a filter along the streamline using Hermite interpolation. For each streamline, two errors were computed using equations (6.15) and (6.6). Observe that this closest point approach gives lower errors for all cases. This type of measurement could be employed as an error estimate since it provides information regarding how close the streamlines remain remain with respect the exact solution.

6.5 Discussion

This chapter explored the applications of Line filtering during flow visualisation. Earlier numerical studies (Chapter 4) demonstrated the ability of these filters to recover

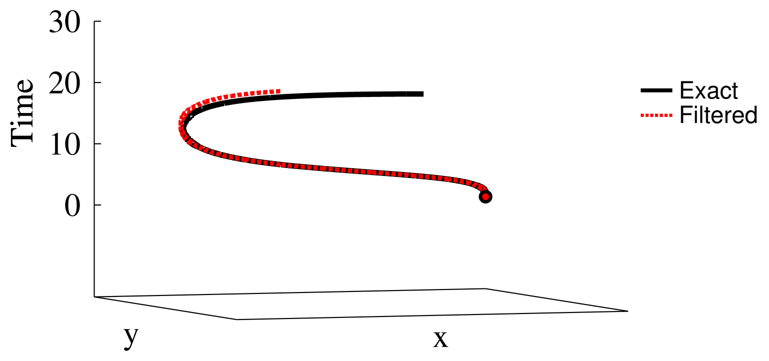


Figure 6.17: Streamlines corresponding to the second field CF2 with seed at (.202,-.3) adding the time dimension. Observe how the exact streamline has a longer trajectory and that although the filtered streamline is delayed, it remains close to the exact solution.

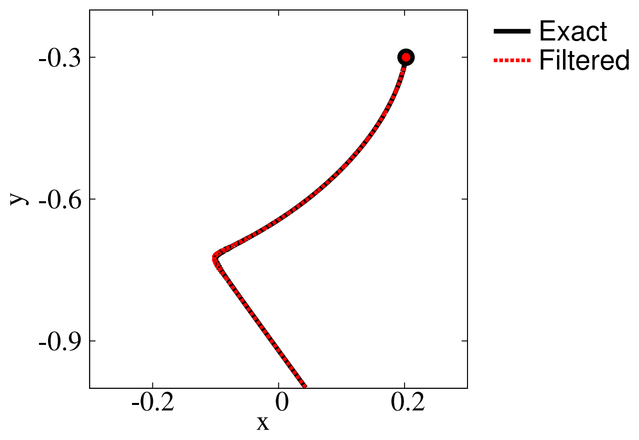


Figure 6.18: Streamline from Figure 6.17 (without the time variable) with different final times, allowing a larger time for the filtered streamline showing the overlap between the curves.

CF1						
	Unfiltered		**Filtered**			
			$\frac{\pi}{4}$-SLF		**FAS-H**	
Seed	$\max(e_n)$	$d_B(A)$	$\max(e_n)$	$d_B(A)$	$\max(e_n)$	$d_B(A)$
N= 40 × 40						
(-.6,-.651)	DIVERGED		2.1e-1	1.0e-1	DIVERGED	
(-.6,-.192)	4.0e-3	2.4e-3	6.1e-3	1.9e-3	3.2e-2	1.8e-2
(-.8,-.3)	4.1e-2	4.6e-3	3.8e-2	3.8e-3	1.8e-2	1.1e-2
N= 80 × 80						
(.202,-.3)	DIVERGED		1.8e-1	1.9e-3	DIVERGED	
(-.09,-.1)	DIVERGED		2.2e-2	7.0e-3	3.2e-2	2.8e-2
(.1,-.3)	5.4e-3	5.4e-3	7.1e-3	7.1e-3	6.6e-2	3.4e-2
CF2						
	Unfiltered		**Filtered**			
			$\frac{\pi}{4}$-SLF		**FAS-H**	
Seed	$\max(e_n)$	$d_B(A)$	$\max(e_n)$	$d_B(A)$	$\max(e_n)$	$d_B(A)$
N= 40 × 40						
(-.6,-.651)	DIVERGED		4.3e-2	1.1e-2	2.5e-3	2.5e-3
(-.6,-.192)	2.6e-3	9.8e-4	5.5e-4	4.1e-4	6.9e-3	6.9e-3
(-.8,-.3)	3.6e-2	6.2e-3	2.7e-2	3.6e-3	3.0e-2	3.3e-3
N= 80 × 80						
(.202,-.3)	2.2e-1	2.8e-3	1.2e-2	7.1e-4	2.5e-1	6.0e-3
(-.09,-.1)	7.6e-2	1.9e-2	7.0e-3	2.1e-4	2.9e-2	2.9e-2
(.1,-.3)	7.5e-3	7.5e-3	6.8e-3	6.8e-3	3.1e-2	3.1e-2

Table 6.6: Error estimates using the greatest point distance from all the iterations ($\max(e_n)$) given by equation (6.15) and using the maximum point-to-curve distance ($d_B(A)$) given by equation (6.21). The filtered solutions were computed with a symmetric line filter (sfl) and a filter along the streamline using Hermite reconstruction (FAS-H). The DG approximation was computed using $\mathbb{P}^1$ polynomials and all the streamlines were obtained through a RK2 method with time step $dt = 0.01$.

smoothness and increase the accuracy from the original DG solution. Here, the filters have been applied to more general vector fields (including singularities) and once again, the results suggest that this post-processor enhances the accuracy from the original solution, leading in this case, to more accurate streamlines.

Line filters using the symmetric kernel showed excellent performance and the low computational times associated with them make them great candidates for engineering applications. In general, the streamlines computed with these filters had the same accuracy than those obtained through a Tensor Product filter, which requires a more complicated implementation and larger computational times. However, the experiments suggested that mesh resolution has a stronger effect on Line filters, showing how over coarse meshes, applying Tensor Product filters results in more accurate streamlines. On the other hand, this limitation was overcome by increasing the order and number of B-Splines employed to build the Line kernel, producing a solution that matched the quality of the one obtained through the Tensor Product filter.

Boundary Line filters resulted in less effective post-processing. This was expected since previous numerical results on one-sided filters already showed that they cannot reduce the error as much as the symmetric kernel. However, the Boundary Line Filters were included as preliminary work, to give an idea of their behaviour. The length of this thesis allowed the development of theoretical estimates of symmetric filtering alone and one-sided filtering was left as future work. The experiments suggest that these filters (in particular filters along the streamline) can be suitable for accuracy enhancement during flow visualisation. In many cases where the unfiltered streamline diverged from the exact solution, the filtered curve converged back towards the exact curve. From this study it was not possible to conclude whether a particular orientation or type of boundary filter could give optimal results because there are many possible configurations. For instance, there is no guarantee that the kernel scalings used here $(h = h,\ H = \sqrt{2}h)$ are the appropriate ones for flow aligned filters. In fact, it could be possible that since the rotation angle changes, the kernel scaling should also vary within the filter location.

The ODE solver employed for most simulations corresponded to the RK2 method. This is a relatively low order method for streamline computations which are generally done using higher order solvers (RK3, RK4) in order to ensure convergence. However, the theory for ODE solvers assumes analytic fields and does not account for the error introduced by the numerical method that was employed to compute the vector field. Hence, applying the RK3 or RK4 solver over a field obtained with low degree polynomials can result ineffective since no accuracy is gained from using a higher order solver. For example, the RK4 method uses information from higher order derivatives which are not available when producing a DG solution with a $\mathbb{P}^1$ or $\mathbb{P}^2$ polynomial basis. In addition, since the filter has to be applied at each stage, lower number of stages is desirable. For example, implementing the RK4 method would double the number of filtering points per time step. This usually can be counter-balanced by enlarging the time step but the value used in the experiments $(h = 0.01)$ is already relatively large, so the RK2 solver seems suitable for these filters.

Chapter 7

Conclusions and Future Research

The theory and applications of SIAC filters in multidimension has traditionally employed a tensor product structure constructed using one-dimensional kernels. In addition, the tensor product has always been done along the Cartesian axis, resulting in a filter whose support has fixed shape and orientation. This thesis has challenged these assumptions, leading to the investigation of rotated filters: tensor product filters with variable orientation. Furthermore, combining this approach with previous experiments on lower-dimension filtering, a new and efficient type of post-processor has been developed: SIAC Line filters. These filters transform the integral of the convolution into a line integral, reducing significantly the computational times and complexity of the algorithm design. A solid theoretical background for SIAC Line filters has been developed and in Theorem 4.3.1, superconvergent error estimates similar to those for tensor product filtering were proven. Using this one-dimensional approach, SIAC filters can be applied to multidimensional fields in an efficient way, becoming an attractive tool for the scientific community.

The development of SIAC Line filters began with the idea of 2D rotated filters. The numerical results suggested that rotated filters preserve the properties of SIAC filtering in terms of superconvergence and smoothness recovery. Compared to the Cartesian axis aligned filter, however, these filters resulted in less effective error reduction. In practice, smoothness recovery and error reduction are more relevant features. Hence, it was concluded that the original Cartesian axis aligned filter was already optimal. Nevertheless, implementing these filters has contributed towards the general application of SIAC filters; designing a filter that allows for post-processing in any direction has lead to the development of a very robust algorithm.

As a starting point, rotated filters were applied to very simple models and imposing uniform squared meshes. Previous studies have shown that the Cartesian axis aligned filter looses accuracy as the mesh becomes less uniform or when applied to non-linear problems. This thesis employed uniform meshes only but the rotated filters should be investigated over more general arbitrary meshes. It still remains a question whether the mesh conditions or the nature of the hyperbolic problem would imply that alternative orientations or kernels with non-orthogonal inner axis would be more suitable.

Theorem 4.3.1 provides superconvergent error estimates for SIAC Line filters. In addition, the numerical results revealed that, in general, the filtered solution has lower error than the original DG solution. This was first studied globally and locally over DG solutions alone. Later, during flow visualisation, the filters were applied to general velocity fields (containing singularities) with a view to improve the field conditions where streamlines were being computed. The results that the filter performs efficiently for this type of post-processor and produces highly accurate streamlines in cases where the unfiltered streamline diverged from the exact solution.

During the investigations of SIAC Line filters for visualisation, several types of kernels were implemented. Furthermore, the numerical results were compared against tensor product filtering so that the impact from reducing the dimension could be estimated. The experiments suggested that Line filters with symmetric kernels produced streamlines as accurate as those computed using a Tensor Product filter (aligned with the Cartesian axis). However, it was observed that the mesh resolution had a stronger effect on Line filters. The studies over coarse meshes showed that applying Tensor Product filters consistently resulted in more accurate streamlines. This limitation was overcome by increasing the degree and number of splines employed to build the Line kernel. As a result, streamline computations using higher degree Line filters matched (if not improved) the results from the 2D filter. Although increasing the number and order of the B-Splines enlarges the support, the computational times still remain very low compared to tensor product filtering. Thus, Line filtering becomes a promising alternative for post-processing in multidimensions.

In contrast, boundary Line filters resulted in less effective post-processing. This was studied for streamline visualisations and only experimentally. The results suggested that these filters (in particular filters along the streamline) enhance the field conditions but the kernel design should be improved. For all the cases where the original DG solution diverged, a particular boundary filter was able to produce a solution which converged back towards the exact curve. However, it was not possible to determine a suitable filter for *all* streamlines. The length of this thesis allowed the development of theoretical estimates for symmetric filtering and boundary filtering was left as future work. From the numerical studies alone, it is not possible to conclude whether a particular orientation or filter type could give optimal results. Furthermore, these type of filters offer many possible configurations. For instance, there is no guarantee that the kernel scalings used here ($H = h,\ \sqrt{2}h$) are the appropriate ones for flow aligned filters. In fact, it is possible that since the rotation angle changes, perhaps the kernel scaling should also vary within the filter location.

Finally, the experiments confirmed that Line filters can be combined with relatively simple ODE solvers. Streamline computations are typically done through higher order solvers such as the RK4, which is fourth order. However, most simulations performed here employed the RK2 solver which using two stages, only gives second order. Since the filter has to be applied at each stage, this affords a great advantage. For example, if one wishes to implement the RK4 method, this implies doubling the number of filtering points per time step. Although higher order solvers allow for larger time steps, the streamline experiments presented here were computed using $dt = 0.01$, which is already a relatively large step. In addition, it was observed that the RK3 and RK4 solvers were inefficient for DG fields computed using low degree polynomials. For such cases, dominant errors come from the field data itself and implementing the RK3 or RK4 method gave exactly the same streamlines than the RK2 method. Therefore, it was concluded that the solver should require as low number of stages as possible. Furthermore, for low mesh resolution problems, rather than increasing the order of the solver, the kernel should increase the order and number of B-Splines.

7.1 Future Research

This thesis provides a solid background on a family of SIAC filters with variable support orientation. The investigations were performed over tensor product kernels as well as line kernels. The length of this research has allowed for development of the mathematical formulation of these filters together with the implementation but has limited the study to linear hyperbolic problems over uniform meshes. In order for these filters to to become a suitable tool for CFD applications, investigations on non-linear problems involving unstructured meshes should be carried out.

SIAC filters have already been applied to a wide variety hyperbolic problems such as the advection equation with variable coefficient as well as the non-linear Burgers equation. However, for non-linear problems, error estimates for the divided differences of the DG solution do not exist, so the theory for SIAC filtering strongly relies on results from the linear case. On the other hand, it has been shown computationally how the filter increases the smoothness and accuracy order from the DG solution, even in the presence of a shock. Hence, the next step for the rotated filters is to change the type of hyperbolic problem as well as introduce solutions containing shocks.

Moreover, the original tensor product filter has already been implemented over triangular elements in 2D and tetrahedral elements in 3D. For structured triangular meshes and linear hyperbolic problems, it is possible to prove that the filtered solution achieves $2k + 1$ order, both theoretically and computationally. The numerical experiments performed on the Line filters were limited to quadrilateral elements. Based on the existing proofs for tensor product filters, theoretical and computational work should be done over triangular elements. This would provide a solid foundation for extending the applications of Line filters to solutions over general meshes, including curvilinear elements.

One of the most challenging questions around SIAC filtering is the problem of finding the optimal scaling that allows for greatest error reduction. This presents big limitations when the filters are applied over non-uniform meshes. The existing theoretical error estimates over such meshes suggest a scaling that in practice, is not optimal in terms of the magnitude of the error. Furthermore, this choice is based on the underlying mesh itself and does not consider flow features. Several numerical examples shown in this thesis suggested that, leaving superconvergence aside, in terms of error reduction, Line filters can obtain satisfactory results for alternative scaling choices based on the mesh as well as the flow. This could help in designing the right scaling when the filter is applied together with a ODE solver during streamline computations. Furthermore, the rotated filters using tensor product kernels were tested over uniform quadrilateral meshes. For such structures, a kernel aligned with the Cartesian axis is also aligned with the elements. On the other hand, introducing unstructured triangular meshes destroys this alignment. Rotated filters should be tested over such meshes since the Cartesian axis aligned would no longer be under optimal conditions. This could lead to cases where a rotated kernel outperformed the original axis aligned one.

Flow features such as vorticity, involve performing computations on the field itself as well as its derivatives. The investigations on SIAC filters have already extended to derivative filtering and there are available theoretical and numerical results. Line filters revealed that when using the appropriate rotation and scaling, it is possible to recover smoothness in any direction even though the support is fixed. Therefore, in the future, it should be explored whether it is possible to obtain similar results on the field derivatives. In addition, flow visualisation experiments were done only over streamlines. The velocity fields employed during the experiments represented steady flows but the filter should not be limited for such cases alone. The applications of Line filters should extend to unsteady flows, introducing the post-processing step parallel to vector field computations, enhancing the field conditions where for example, streaklines are being calculated.

The notion of directional divided difference discussed in chapter 4 is not necessarily restricted to the 2D space. Although the theoretical error estimates given in Theorem 4.3.1 were done for the two dimensional case, they should extend to higher dimensions. The goal of Line filtering is to be possible to post-process in three dimensions so that this technique can be applied in real-world problems. However, since adding a dimension implies a whole new possible set of filtering orientations, due to lack of time, the theory and experiments were limited to the 2D case, leaving higher dimensions for future research. In addition, the theoretical error estimates for the Line filters presented here have only considered symmetric kernels. During streamline visualisation, one-sided kernels were also introduced but these type of kernels were only studied experimentally. Applying Line filters to physical problems requires post-processing over computational domains that do not necessarily involve periodic boundary conditions. Therefore, this technique should ensure effective post-processing near the boundary. With the existing proofs for one-sided filtering, theoretical estimates for boundary Line filters should be developed. This would lead to a robust and suitable post-processor extending their potential to a wider set of problems.

Regarding the implementation of the filter, the point-scan algorithm should be improved; during the computation of the kernel breaks and mesh intersection points, the algorithm uses lines to join the points building the kernel support. Although this computation is exact for line filtering, the same does not apply for the filters implemented along the streamline since their support expands along the curve. It was observed that applying Hermite interpolation for the curve reconstruction resulted in more accurate streamlines compared to those filtered using Linear interpolation. However, the interpolation nodes location corresponding to kernel or DG mesh breaks is not exact, since at the moment, that step is done joining the points by straight segments. In the future, the algorithm should include curve intersection routines that would allow for determining more accurately the location of the integral breaks.

Finally, SIAC Line filters should not be restricted to DG methods alone. The applications of these filters should be extended for example, to Finite Element or Finite Difference Methods. This will bring the filter closer to applications to turbulent models obtained through RANS or LES.

Bibliography

[1] Arnold, D. N. and Brezzi, F. and Cockburn, B. and Marini, L. D. Unified Analysis of Discontinuous Galerkin Methods for Elliptic Problems. *SIAM Journal of Numerical Analysis 39*, 5 (May 2001), 1749–1779.

[2] Bramble, J. H. and Schatz, A. H. Higher Order Local Accuracy by Averaging in the Finite Element Method. *Mathematics of Computation 31*, 137 (1977), pp. 94–111.

[3] Burden, R. L. and Faires, J. D. *Numerical Analysis: 4th Edition.* PWS Publishing Co., Boston, MA, USA, 1989.

[4] Butcher, J. C. *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods.* Wiley-Interscience, New York, NY, USA, 1987.

[5] Butcher, J. C. A History of Runge-Kutta Methods. *Applied Numerical Mathematics 20*, 3 (1996), 247 – 260.

[6] Butcher, J. C. and Johnston, P. B. Estimating Local Truncation Errors for Runge-Kutta Methods. *Journal of Computational and Applied Mathematics 45*, 1 (1993), 203 – 212.

[7] Cantwell, C. D. and Moxey, D. and Comerford, A. and Bolis, A. and Rocco, G. and Mengaldo, G. and De Grazia, D. and Yakovlev, S. and Lombard, J.-E. and Ekelschot, D. and Jordi, B. and Xu, H. and Mohamied, Y. and Eskilsson, C. and Nelson, B. and Vos, P. and Biotto, C. and Kirby, R. M. and Sherwin, S. J. Nektar++: an Open-Source Spectral/ Element Framework. *Computer Physics Communications 192* (2015), 205 – 219.

[8] Chong-Wei, H. and Tian-Yuan, S. On the Complexity of Point-in-Polygon Algorithms. *Computers & Geosciences 23*, 1 (1997), 109 – 118.

[9] Cockburn, B. *An Introduction to the Discontinuous Galerkin Method for Convection-Dominated Problems.* Springer Berlin Heidelberg, 1998, pp. 151–268.

[10] Cockburn, B. and Dong, B. and Guzmán, J. Optimal Convergence of the Original DG Method for the Transport-Reaction Equation on Special Meshes. *SIAM Journal on Numerical Analysis 46*, 3 (2008), 1250–1265.

[11] Cockburn, B. and Karniadakis, G. E. and Shu, C-W. *The Development of Discontinuous Galerkin Methods.* Springer Berlin Heidelberg, 2000, pp. 3–50.

[12] Cockburn, B. and Lin, S-Y. and Shu, C-W. . TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws III: One-Dimensional Systems. *Journal of Computational Physics 84*, 1 (1989), 90–113.

[13] Cockburn, B. and Luskin, M. and Shu, C-W. and Süli, E. Enhanced Accuracy by Post-Processing for Finite Element Methods for Hyperbolic Equations. *Mathematics of Computation 72*, 242 (Apr. 2003), 577–606.

[14] Cockburn, B. and Shu, C-W. Runge-Kutta Discontinuous Galerkin Methods for Convection-Dominated Problems. *Journal of Scientific Computing 16*, 3 (2001), 173–261.

[15] Curtis, S. and Kirby, R. M. and Ryan, J. K. and Shu, C-W. Postprocessing for the Discontinuous Galerkin Method over Nonuniform Meshes. *SIAM Journal on Scientific Computing 30*, 1 (2008), 272–289.

[16] Darel, R. F. Point-in-Polygon Algorithm-Determining Whether a Point is Inside a Complex Polygon. `http://alienryderflex.com/polygon/`, 2007. [Online; accessed 16-August-2016].

[17] De Boor, C. *A Practical Guide to Splines.* No. v. 27 in Applied Mathematical Sciences. Springer-Verlag New York, 1978.

[18] Fehlberg, E. New High-Order Runge-Kutta Formulas with Step Size Control for Systems of First-and Second-Order Differential Equations. In *Zeitschrift fur Angewandte Mathematik und Mechanik*, vol. 44. Wiley-V CH Verlag Gmbh Muhlenstrasse 33-34, D-13187 Berlin,Germany, 1964, p. T17.

[19] Fehlberg, E. Low-Order Classical Runge-Kutta Formulas with Stepsize Control and their Application to Some Heat Transfer Problems. Tech. rep., NASA, 1969.

[20] Gear, C. W. and Osterby, O. Solving Ordinary Differential Equations with Discontinuities. *ACM Transactions on Mathematical Software 10*, 1 (Jan. 1984), 23–44.

[21] Gottlieb, S. and Ketcheson, D. I. and Shu, C-W. *Strong Stability Preserving Runge-Kutta and Multistep Time Discretizations.* World Scientific Publishing Co., 2011.

[22] Graham, R. L. An Efficient Algorith for Determining the Convex Hull of a Finite Planar Set. *Information processing letters 1*, 4 (1972), 132–133.

[23] Hartman, P. *Ordinary Differential Equations.* New York ; John Wiley, 1964, 1964.

[24] Hatcher, A. *Algebraic Topology.* Cambridge University Press, 2002.

[25] Hormann, K. and Agathos, A. The Point in Polygon Problem for Arbitrary Polygons. *Computational Geometry 20*, 3 (2001), 131 – 144.

[26] Hull, T. E. and Creemer, A. L. Efficiency of Predictor-Corrector Procedures. *Journal of the ACM (JACM) 10*, 3 (1963), 291–301.

[27] Ji, L. and Van Slingerland, P. and Ryan, J. K. and Vuik, K. C. Superconvergent Error Estimates for Position-Dependent Smoothness-Increasing Accuracy-Conserving (SIAC) Post-Processing of Discontinuous Galerkin Solutions. *Mathematics of Computation 83*, 289 (2014), 2239–2262.

[28] Ji, L. and Xu, Y. and Ryan, J. K. Accuracy-Enhancement of Discontinuous Galerkin Solutions for Convection-Diffusion Equations in Multiple-Dimensions. *Mathematics of Computation 81*, 280 (2012), 1929–1950.

[29] Ji, L. and Xu, Y. and Ryan, J. K. Negative-Order Norm Estimates for Nonlinear Hyperbolic Conservation Laws. *Journal of Scientific Computing 54*, 2 (2013), 531–548.

[30] Johnson, C. and Pitkäranta, J. An Analysis of the Discontinuous Galerkin Method for a Scalar Hyperbolic Equation. *Mathematics of computation 46*, 173 (1986), 1–26.

[31] Karniadakis, G. E. and Sherwin, S. *Spectral/hp Element Methods for Computational Fluid Dynamics: Second Edition.* Numerical Mathematics and Scientific Computation. OUP Oxford, 2005.

[32] King, J. and Mirzaee, H. and Ryan, J. K. and Kirby, R. M. Smoothness-Increasing Accuracy-Conserving (SIAC) Filtering for Discontinuous Galerkin Solutions: Improved Errors Versus Higher-Order Accuracy. *Journal of Scientific Computing 53*, 1 (2012), 129–149.

[33] Kirby, R. M. and Karniadakis, G. E. Selecting the Numerical Flux in Discontinuous Galerkin Methods for Diffusion Problems. *Journal of Scientific Computing 22*, 1 (2005), 385–411.

[34] Klopfenstein, R. W. and Millman, R. S. Numerical Stability of a One-Evaluation Predictor-Corrector Algorithm for Numerical Solution of Ordinary Differential Equations. *Mathematics of Computation 22*, 103 (1968), 557–564.

[35] Li, X. *Smoothness-Increasing Accuracy-Conserving Filters for Discontinuous Galerkin Methods: Challenging the Assumptions of Symmetry and Uniformity.* PhD Thesis in Applied Mathematics, Delft University of Technology, Numerical Analysis Department, The Netherlands, 2015. ISBN 9789461868007.

[36] Li, X. and Ryan, J. K. and Kirby, R. M. and Vuik, C. K. Smoothness-Increasing Accuracy-Conserving (SIAC) Filters for Derivative Approximations of Discontinuous Galerkin (DG) Solutions over Nonuniform Meshes and near Boundaries. *Journal of Computational and Applied Mathematics 294* (2016), 275 – 296.

[37] Meng, X. and Ryan, J. K. Discontinuous Galerkin Methods for Nonlinear Scalar Hyperbolic Conservation Laws: Divided Difference Estimates and Accuracy Enhancement. *Numerische Mathematik* (2016), 1–47.

[38] Mirzaee, H. and King, J. and Ryan, J. K. and Kirby, R. M. Smoothness-Increasing Accuracy-Conserving Filters for Discontinuous Galerkin Solutions over Unstructured Triangular Meshes. *SIAM Journal on Scientific Computing 35*, 1 (2013), A212–A230.

[39] Mirzaee, H. and Ryan, J. K. and Kirby, R. M. Quantification of Errors Introduced in the Numerical Approximation and Implementation of Smoothness-Increasing Accuracy-Conserving (SIAC) Filtering of Discontinuous Galerkin (DG) Fields. *Journal of Scientific Computing 45*, 1 (2010), 447–470.

[40] Mirzaee, H. and Ryan, J. K. and Kirby, R. M. Efficient Implementation of Smoothness-Increasing Accuracy-Conserving (SIAC) Filters for Discontinuous Galerkin Solutions. *Journal of Scientific Computing 52*, 1 (7 2012), 85–112.

[41] Mirzaee, H. and Ryan, J. K. and Kirby, R. M. Smoothness-Increasing Accuracy-Conserving (SIAC) Filters for Discontinuous Galerkin Solutions: Application to Structured Tetrahedral Meshes. *Journal of Scientific Computing 58*, 3 (2014), 690–704.

[42] Mirzaee, H. and Ryan, J. K. and Kirby, R. M. Smoothness-Increasing Accuracy-Conserving (SIAC) Postprocessing for Discontinuous Galerkin Solutions over Structured Triangular Meshes. *Journal of Scientific Computing 58*, 3 (2014), 690–704.

[43] Mirzargar, M. and Ryan, J. K. and Kirby, R. M. Smoothness-Increasing Accuracy-Conserving (SIAC) Filtering and Quasi-Interpolation: A Unified View. *Journal of Scientific Computing 67*, 1 (2016), 237–261.

[44] Mock, M. S. and Lax, P. D. The Computation of Discontinuous Solutions of Linear Hyperbolic Equations. *Communications on Pure and Applied Mathematics 31*, 4 (1978), 2423–430.

[45] Nguyen, D-M and Peters, J. Nonuniform Discontinuous Galerkin Filters via Shift and Scale. *SIAM Journal on Numerical Analysis 54*, 3 (2016), 1401–1422.

[46] O'Rourke, J. *Computational Geometry in C.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998.

[47] Peters, J. General Spline Filters for Discontinuous Galerkin Solutions. *Computers & Mathematics with Applications 70*, 5 (2015), 1046–1050.

[48] Poole, D. *Linear Alebra. a Modern Introduction, 2nd Edition .* Thomson Brooks/Cole, Canada, 2006.

[49] Reed, W. H. and Hill, T. R. Triangular Mesh Methods for the Neutron Transport Equation. Tech. rep., Los Alamos Scientific Lab., N. Mex.(USA), 1973.

[50] Richter, G. R. An Optimal-Order Error Estimate for the Discontinuous Galerkin Method. *Mathematics of Computation 50*, 181 (1988), 75–88.

[51] Ryan, J. K. Local Derivative Post-Processing: Challenges for a Non-Uniform Mesh. Tech. Rep. 10-18, Delft University of Technology, Delft, Netherlands, 2010.

[52] Ryan, J. K. *Exploiting Superconvergence through Smoothness-Increasing Accuracy-Conserving (SIAC) Filtering.* Springer International Publishing, Cham, 2015, pp. 87–102.

[53] Ryan, J. K. and Cockburn, B. Local Derivative Post-Processing for the Discontinuous Galerkin Method. *Journal of Computational Physics 228*, 23 (2009), 8642–8664.

[54] Ryan, J. K. and Kirby, R. M. Efficient Computation of B-Spline Weighting Coefficients for SIAC Filters. 2011.

[55] Ryan, J. K. and Li, X. and Kirby, R. M. and Vuik, K. C. One-Sided Position-Dependent Smoothness-Increasing Accuracy-Conserving (SIAC) Filtering Over Uniform and non-Uniform Meshes. *Journal of Scientific Computing 64*, 3 (2015), 773–817.

[56] Ryan, J. K. and Shu, C-W. On a One-Sided Post-Processing Technique for the Discontinuous Galerkin Methods. *Methods and Applications of Analysis 10*, 2 (06 2003), 295–308.

[57] Ryan, J. K. and Shu, C-W. and Atkins, H. Extension of a Postprocessing Technique for the Discontinuous Galerkin Method for Hyperbolic Equations with Application to an Aeroacoustic Problem. *SIAM Journal on Scientific Computing 26*, 3 (2005), 821–843.

[58] Salomon, K. B. An Efficient Point-in-Polygon Algorithm. *Computers & Geosciences 4*, 2 (1978), 173 – 178.

[59] Schumaker, L. *Spline Functions: Basic Theory, 3rd Edition.* Cambridge University Press, 2007. Cambridge Books Online.

[60] Shu, C-W. Discontinuous Galerkin Methods: General Approach and Stability. In *Numerical Solutions of Partial Differential Equations*, Bertoluzza, S. and Falletta, S. and Russo, G. and Shu, C.W., Ed., Advanced Courses in Mathematics - CRM Barcelona. Birkhäuser Basel, 2008, ch. 3, pp. 149–201.

[61] Steffen, M. and Curtis, S. and Kirby, R. M. and Ryan, J. K. Investigation of Smoothness-Increasing Accuracy-Conserving Filters for Improving Streamline Integration through Discontinuous Fields. *IEEE Transactions on Visualization and Computer Graphics 14*, 3 (2008), 680–692.

[62] Stoer, J. and Bartels, R. and Gautschi, W. and Bulirsch, R. and Witzgall, C. *Introduction to Numerical Analysis.* Texts in Applied Mathematics. Springer New York, 2002.

[63] van Slingerland, P. and Ryan, J. K. and Vuik, K. C. Position-Dependent Smoothness-Increasing Accuracy-Conserving (SIAC) Filtering for Improving Discontinuous Galerkin Solutions. *SIAM Journal on Scientific Computing 33*, 2 (2011), 802–825.

[64] Walfisch, D. and Ryan, J. K. and Kirby, R. M. and Haimes, R. One-Sided Smoothness-Increasing Accuracy-Conserving Filtering for Enhanced Streamline Integration through Discontinuous Fields. *Journal of Scientific Computing 38*, 2 (2009), 164–184.