

# THE CLOSEST POINT METHOD FOR TIME-DEPENDENT PROCESSES ON SURFACES

by

Colin B. Macdonald

B.Sc., Acadia University, 2001

M.Sc., Simon Fraser University, 2003

A THESIS SUBMITTED IN PARTIAL FULFILLMENT  
OF THE REQUIREMENTS FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY  
IN THE  
DEPARTMENT OF MATHEMATICS

© Colin B. Macdonald 2008  
SIMON FRASER UNIVERSITY  
Summer 2008

Some rights reserved. This work may not be reproduced in whole or in part, by photocopy or other means, without permission of the author, except for scholarly or other non-commercial use for which no further copyright permission need be requested.

## APPROVAL

**Name:** Colin B. Macdonald  
**Degree:** Doctor of Philosophy  
**Title of thesis:** The Closest Point Method for Time-dependent Processes on Surfaces

**Examining Committee:** Dr. JF Williams  
Chair

---

Dr. Steven J. Ruuth  
Senior Supervisor

---

Dr. Robert D. Russell  
Supervisor

---

Dr. James H. Verner  
Supervisor

---

Dr. Torsten Möller  
Internal Examiner

---

Dr. Hong-Kai Zhao  
External Examiner

**Date Approved:**

---

# Abstract

This thesis concerns the numerical solution of time-dependent partial differential equations (PDEs) on general surfaces using a recent technique known as the Closest Point Method. The Closest Point Method represents surfaces with a closest point representation which leads to great flexibility with respect to surface geometry, among other advantages. The computation itself alternates between two steps: first, an explicit time step is performed using standard finite difference techniques on a narrow band of grid points surrounding the surface embedded in a higher dimension; second, a closest point extension is used to maintain consistency with the original surface PDE.

The Closest Point Method is applied to the important problem of interface motion on surfaces by using level set equations posed on surfaces. New weighted essentially non-oscillatory (WENO) interpolation schemes are derived to perform the necessary closest point extensions. This approach, in combination with standard Hamilton–Jacobi WENO finite difference schemes and explicit time stepping, gives high-order results (up to fifth-order) on a variety of test problems. Example computations are performed on a sphere, torus, triangulated human hand and Klein bottle to demonstrate the flexibility of the method.

A new implicit Closest Point Method is presented for surface PDEs which are stiff, for example, because of diffusion terms. The method uses implicit time-stepping schemes to allow large steps but retains the flexibility with respect to surface geometry of the original explicit Closest Point Method. Numerical convergence studies on the heat equation and a fourth-order biharmonic problem demonstrate the accuracy of the method and a variety of example computations demonstrate its effectiveness. These include an image processing example of blurring on triangulated surfaces, heat diffusion on a surface consisting of multiple components connected by a thin filament and Turing pattern formation on surfaces using implicit–explicit (IMEX) time stepping.

A class of time-stepping methods known as diagonally split Runge–Kutta (DSRK) methods is investigated. These methods appear promising because they offer both high-order convergence and unconditional contractivity (a nonlinear stability property). However, numerical computations and analysis of stage-order demonstrates that unconditionally contractive DSRK methods suffer from order reduction which severely limits their practical application.

**Keywords:**

Closest Point Method; level set methods; surface computation; implicit surfaces; partial differential equations; WENO schemes; time stepping; implicit time stepping; biharmonic operator; diagonally split Runge–Kutta methods; strong stability preserving

# Acknowledgments

The research behind this thesis was supported financially by an NSERC postgraduate scholarship, NSERC grants, the C.D. Nelson Memorial scholarship and the PIMS Graduate Fellowship.

I thank my supervisor Steve Ruuth for introducing me to the Closest Point Method and for his enthusiasm. Special thanks also to my collaborator Sigal Gottlieb.

Finally, I thank my wife and my family for their support.

# Contents

<b>Approval</b>	<b>ii</b>
<b>Abstract</b>	<b>iii</b>
<b>Acknowledgments</b>	<b>v</b>
<b>Contents</b>	<b>vi</b>
<b>List of Tables</b>	<b>x</b>
<b>List of Figures</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Numerical Partial Differential Equations . . . . .	1
1.1.1 The method of lines . . . . .	1
1.1.2 Linear problems . . . . .	2
1.1.3 Hamilton–Jacobi equations . . . . .	3
1.1.4 Hamilton–Jacobi WENO . . . . .	4
1.2 Time Stepping . . . . .	4
1.2.1 Runge–Kutta methods . . . . .	5
1.2.2 Implicit linear multistep methods . . . . .	6
1.2.3 IMEX schemes . . . . .	7
1.3 Consistency, Stability and Convergence . . . . .	8
1.4 Strong Stability Preserving Time Stepping . . . . .	9
1.5 Interpolation . . . . .	9
1.5.1 1D polynomial interpolation . . . . .	10

1.5.2	Barycentric Lagrange interpolation . . . . .	11
1.5.3	Higher dimensional interpolation . . . . .	11
1.5.4	Nonlinear interpolation . . . . .	11
1.6	Surfaces . . . . .	12
1.6.1	Partial differential equations on surfaces . . . . .	12
1.6.2	Representations of surfaces . . . . .	13
1.6.3	Closest point representation of surfaces . . . . .	13
1.6.4	Equivalence of gradients . . . . .	15
1.7	The Closest Point Method . . . . .	17
1.7.1	The explicit Closest Point Method . . . . .	17
1.7.2	Closest point extension and interpolation . . . . .	19
1.7.3	Banding and bandwidth . . . . .	19
1.8	Outline of Thesis Contributions . . . . .	20
<b>2</b>	<b>Level Set Equations on Surfaces</b>	<b>22</b>
2.1	Level Set Methods . . . . .	22
2.1.1	Computing level sets on surfaces . . . . .	24
2.2	WENO Interpolation . . . . .	25
2.2.1	One-dimensional WENO interpolation . . . . .	26
2.3	Banded Calculations . . . . .	30
2.3.1	Bandwidth upper bounds . . . . .	31
2.3.2	The stencil set approach . . . . .	32
2.3.3	Implementation . . . . .	34
2.4	Numerical Results . . . . .	34
2.4.1	Passive transport: flow under a specified velocity field . . . . .	34
2.4.2	Normal flow . . . . .	36
2.4.3	Signed distance and reinitialization . . . . .	37
2.4.4	WENO interpolation on marginally resolved surfaces . . . . .	39
2.4.5	Triangulated surfaces . . . . .	40
2.4.6	Klein bottle . . . . .	42
2.4.7	Forest fires . . . . .	43

<b>3</b>	<b>The Implicit Closest Point Method</b>	<b>46</b>
3.1	Introduction . . . . .	47
3.2	Matrix Formulation of the Closest Point Method . . . . .	48
3.2.1	The discrete extension operator . . . . .	49
3.2.2	The discrete differential operator . . . . .	50
3.2.3	Stabilizing the Closest Point Method matrix . . . . .	53
3.2.4	Re-examining the explicit Closest Point Method . . . . .	54
3.3	Matrix Properties . . . . .	56
3.3.1	Spectra, spectral radius and condition number . . . . .	56
3.3.2	Sparsity . . . . .	57
3.3.3	Diagonal dominance . . . . .	60
3.3.4	Positive definiteness . . . . .	60
3.3.5	Summary of matrix properties . . . . .	61
3.4	Error Analysis . . . . .	61
3.5	Constructing the Computational Band . . . . .	62
3.5.1	Forming the $\Delta_h$ matrix . . . . .	63
3.5.2	Forming the $E$ matrix . . . . .	64
3.6	Numerical Results . . . . .	64
3.6.1	Numerical convergence studies . . . . .	64
3.6.2	Heat equation on surfaces . . . . .	67
3.6.3	Biharmonic problems . . . . .	70
3.6.4	Pattern formation . . . . .	74
3.7	Nonlinear Problems . . . . .	76
<b>4</b>	<b>Diagonally Split Runge–Kutta Methods</b>	<b>78</b>
4.1	Introduction . . . . .	79
4.1.1	Strong stability preserving time stepping . . . . .	79
4.1.2	Implicit strong stability preserving time stepping . . . . .	80
4.2	Diagonally Split Runge–Kutta Methods . . . . .	82
4.2.1	Dissipative systems and contractivity . . . . .	83
4.2.2	DSRK schemes . . . . .	84
4.2.3	Numerical implementation of DSRK . . . . .	86
4.3	Numerical Results . . . . .	87



4.3.1	Convection-driven problems . . . . .	87
4.3.2	Diffusion driven problems . . . . .	92
4.3.3	The Black–Scholes equation . . . . .	92
4.3.4	Hyperbolic conservation laws: Burgers’ equation . . . . .	97
4.4	Stage Order and Order Reduction . . . . .	99
4.4.1	The van der Pol equation . . . . .	99
4.4.2	DSRK schemes with higher underlying stage order . . . . .	100
4.4.3	DSRK schemes with higher stage order . . . . .	101
<b>5</b>	<b>Conclusions</b>	<b>104</b>
<b>A</b>	<b>Implicit Closest Point Method Algorithms</b>	<b>106</b>
A.1	Construction of the Computational Band . . . . .	106
A.1.1	Pass one . . . . .	106
A.1.2	Pass two . . . . .	107
<b>B</b>	<b>WENO Schemes</b>	<b>108</b>
B.1	The Hamilton–Jacobi WENO Procedure . . . . .	108
<b>C</b>	<b>Explicit Strong Stability Preserving Runge–Kutta Schemes</b>	<b>110</b>
C.1	Common SSP Runge–Kutta Schemes . . . . .	110
	<b>Bibliography</b>	<b>112</b>
	<b>Index</b>	<b>119</b>

# List of Tables

2.1	Sufficient bandwidths for Closest Point Method . . . . .	32
2.2	Numerical verification of bandwidths for a Closest Point Method calculation .	33
2.3	Numerical convergence study for passive transport . . . . .	35
2.4	Numerical convergence study for constant normal flow . . . . .	37
2.5	Numerical convergence study for reinitialization equation . . . . .	39
3.1	Spectral radius and condition number for $I - \Delta t M$ . . . . .	57
3.2	Sparsity properties of the matrix $M$ . . . . .	58
3.3	Diagonal dominance of $A_{BE} = I - \Delta t M$ . . . . .	60
3.4	Unsymmetric positive definiteness of $A_{BE} = I - \Delta t M$ . . . . .	61
3.5	Properties of the computation band . . . . .	65
4.1	The 14 order conditions for fourth-order DSRK schemes . . . . .	82
4.2	Numerical convergence study for the linear advection of a sine wave . . . . .	89
4.3	Total variation of the solution for the advection of a square wave . . . . .	89
4.4	Numerical convergence study for the heat equation with smooth initial conditions . . . . .	93
4.5	Black–Scholes numerical convergence study . . . . .	97
4.6	Burgers’ equation numerical convergence study . . . . .	98
4.7	Initial conditions for the van der Pol problem . . . . .	99

# List of Figures

1.1	Example of 1D interpolation with linear and cubic interpolants . . . . .	10
1.2	Choice of grid points for various degrees of 1D interpolating polynomials . . .	10
1.3	Using a 1D interpolation scheme to build a 2D interpolation routine . . . . .	12
1.4	Example of the closest point function for a curve . . . . .	14
1.5	Example of the closest point function for a circle . . . . .	14
1.6	An example of closest point extensions . . . . .	20
1.7	Example of the computational band . . . . .	20
2.1	Level set representation of an interface in 2D and on a surface . . . . .	23
2.2	The one-dimensional WENO interpolation grid and three candidate stencils .	26
2.3	Values of the ideal WENO weights $C_i(x)$ . . . . .	27
2.4	Example contrasting Lagrange and WENO interpolation . . . . .	30
2.5	The minimum bandwidths involved in a Closest Point Method computation .	31
2.6	Passive transport of a circular interface on a sphere . . . . .	35
2.7	Unit normal flow on a torus . . . . .	37
2.8	Lagrange and WENO interpolation on closely spaced surfaces . . . . .	40
2.9	Lagrange and WENO interpolation on increasingly closely spaced surfaces . .	41
2.10	Unit normal flow on “Laurent’s Hand” . . . . .	42
2.11	Reinitialization on a Klein bottle . . . . .	44
2.12	Forest fire computation on the surface of a mountain . . . . .	45
3.1	Example of the sets $L_{\text{evolve}}$ and $L_{\text{ghost}}$ . . . . .	49
3.2	Second-order stencils for the Laplacian in various dimensions . . . . .	52
3.3	Fourth-order stencils for the Laplacian in various dimensions . . . . .	52
3.4	Stable and unstable solutions of the in-surface heat equation. . . . .	53

3.5	Spectra of the $\tilde{\mathbf{M}}$ and $\mathbf{M}$ matrices . . . . .	55
3.6	Examples of the spectrum of $\mathbf{M}$ for various surfaces . . . . .	56
3.7	Spectral radii for Jacobi and Gauss–Seidel iteration matrices . . . . .	58
3.8	Sparsity structure of the $\mathbf{M}$ , $\Delta_h$ and $\mathbf{E}$ matrices . . . . .	59
3.9	Numerical convergence study for the in-surface heat equation using second-order finite differences . . . . .	67
3.10	Numerical convergence study for the in-surface heat equation using fourth-order finite differences . . . . .	68
3.11	Blurring on the surface of Laurent’s Hand by solving the in-surface heat equation . . . . .	68
3.12	A pig-wire-sphere domain with heat source . . . . .	69
3.13	Pig-wire-sphere solution . . . . .	71
3.14	Second-order stencil for the biharmonic operator in 2D . . . . .	72
3.15	Inconsistent biharmonic numerical convergence study . . . . .	72
3.16	Comparison of the sparsity structure of $\mathbf{M}$ and $\mathbf{M}_{\text{bi}}$ . . . . .	73
3.17	Biharmonic numerical convergence studies in 2D and 3D . . . . .	74
3.18	Various patterns of the Brusselator Bunny . . . . .	76
4.1	Oscillations from Crank–Nicolson for advection of a square wave . . . . .	81
4.2	Advection of a square wave after two time steps . . . . .	90
4.3	Numerical convergence study for linear advection of a sine wave . . . . .	91
4.4	Numerical convergence study for linear advection of a square wave . . . . .	91
4.5	Numerical convergence studies for the heat equation with smooth initial conditions . . . . .	93
4.6	Numerical convergence study for heat equation with discontinuous initial conditions . . . . .	94
4.7	Computational domain and initial conditions for the Black–Scholes problem . . . . .	94
4.8	Numerical solutions of the Black–Scholes problem . . . . .	96
4.9	Burgers’ equation with Crank–Nicolson and DSRK2 . . . . .	98
4.10	Numerical convergence study on the van der Pol equation . . . . .	100
4.11	Stage order numerical convergence study for linear advection of a sine wave . . . . .	103
4.12	Stage order numerical convergence study on the van der Pol equation . . . . .	103

# Chapter 1

## Introduction

The Closest Point Method is a new general technique for the numerical solution of partial differential equations and other processes on surfaces. The method was introduced in [RM08, MR07], and this thesis makes substantial contributions to its development.

The Closest Point Method uses three basic techniques from numerical analysis: the method of lines, interpolation and time stepping. This chapter begins by reviewing these three concepts, before introducing the Closest Point Method itself.

### 1.1 Numerical Partial Differential Equations

Most of this thesis deals with the numerical solution of partial differential equations (PDEs) on surfaces. In this section we review some important concepts that form the basis for the techniques introduced in later sections

#### 1.1.1 The method of lines

The method of lines is a widely used technique for approximating partial differential equations by discretizing in all but one dimension to obtain a large system of ordinary differential equations (ODEs). Often the spatial dimensions are discretized, resulting in a system of ODEs in time; this process is known as a spatial semi-discretization. A numerical solution to the PDE is then obtained by computing a numerical solution of the system of ODEs using a time-stepping scheme.

Consider, for example, the nonlinear PDE

$$u_t - f(u, \nabla u, \Delta u) = 0, \quad (1.1)$$

where  $\nabla u$  and  $\Delta u$  are respectively the gradient and Laplacian of  $u$ , with initial conditions  $u(t, 0) = u^0$  and solution  $u(t, \mathbf{x}) : \mathbb{R} \times \mathbb{R}^d \rightarrow \mathbb{R}$ . A possible method-of-lines approach begins by approximating  $u(t, \mathbf{x}) \in \mathbb{R}$  with a vector  $\mathbf{u}(t) \in \mathbb{R}^m$  of values  $u_i$  on a grid of points  $\mathbf{x}_i \in \mathbb{R}^d$ ,  $i = 1, \dots, m$ . Finite difference schemes [Tho95] (or other techniques) are then used to approximate the spatial derivatives over these spatial grid points. For example, one possible semi-discretization of the above PDE in 1D at each interior point  $x_i$  would be

$$\frac{\partial}{\partial t} u_i(t) = f \left( u_i, \frac{u_{i+1} - u_i}{\Delta x}, \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} \right), \quad i = 2, \dots, m-1, \quad (1.2)$$

combined with some boundary condition-dependent treatment for the end points  $u_1$  and  $u_m$ . The grid spacing  $\Delta x$  depends on how many spatial grid points are used; in this thesis, equispaced grids are used so  $\Delta x$  will be constant. In a semi-discretization like (1.2), the ODEs for each  $u_i$  are coupled together by the approximation of the spatial derivatives. Thus the original PDE is approximated by the coupled system of ODEs

$$\frac{\partial}{\partial t} \mathbf{u}(t) = \mathbf{f}(\mathbf{u}(t)), \quad (1.3a)$$

$$\mathbf{u}(t_0) = \mathbf{u}^0, \quad (1.3b)$$

where  $\mathbf{f}(\mathbf{u}(t))$  approximates  $f(u, \nabla u, \Delta u)$  at each of the spatial grid points.

### 1.1.2 Linear problems

If both the PDE and the spatial discretization are linear then instead of (1.3), we obtain the linear system of ODEs

$$\frac{\partial}{\partial t} \mathbf{u}(t) = \mathbf{M}\mathbf{u}(t), \quad (1.4a)$$

$$\mathbf{u}(t_0) = \mathbf{u}^0, \quad (1.4b)$$

where  $\mathbf{M}$  is a constant  $m \times m$  matrix. For example, in 1D on a periodic domain, a semi-discretization of the heat equation  $u_t = u_{xx}$  in the form of (1.4) might have

$$\mathbf{M} = \frac{1}{\Delta x^2} \begin{bmatrix} -2 & 1 & & & 1 \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ 1 & & & 1 & -2 \end{bmatrix}. \quad (1.5)$$

### 1.1.3 Hamilton–Jacobi equations

An important class of time-dependent PDEs are the Hamilton–Jacobi equations of the form

$$\phi_t + H(t, x, \nabla \phi) = 0,$$

where the *Hamiltonian*  $H$  is a function of the gradient  $\nabla \phi$ . Hamilton–Jacobi equations occur in the motion of interfaces such as the level set equations in Chapter 2, and for these problems we will typically use  $\phi$  instead of  $u$  for consistency with the literature [OF03, Set99]. The solutions of Hamilton–Jacobi equations can develop nonsmooth solutions with kinks (discontinuous first derivatives) even from smooth initial conditions. This property and the hyperbolic nature of the equations mean that care must be taken in their discretization.

An example of a particular Hamilton–Jacobi equation is the 3D linear advection equation

$$\phi_t + \mathbf{V} \cdot \nabla \phi = 0, \quad (1.6)$$

which has  $H(\nabla \phi) = \mathbf{V} \cdot \nabla \phi$  and describes flow under a specified velocity field  $\mathbf{V} = [v_1(t, \mathbf{x}), v_2(t, \mathbf{x}), v_3(t, \mathbf{x})]^\top$ . A spatial semi-discretization of (1.6) can be obtained using *upwind differencing* in a dimension-by-dimension fashion as follows. For each grid point  $\mathbf{x}_{ijk}$  we compute the forward difference

$$\phi_x^+ = \frac{\phi_{i+1,j,k} - \phi_{ijk}}{\Delta x},$$

and backward difference

$$\phi_x^- = \frac{\phi_{ijk} - \phi_{i-1,j,k}}{\Delta x}.$$

If  $v_1$ , the first component of  $\mathbf{V}$ , is positive then we approximate  $\phi_x$  with  $\phi_x^-$ ; otherwise, we approximate  $\phi_x$  with  $\phi_x^+$ . The same procedure is repeated to approximate  $\phi_y$  and  $\phi_z$ . We

then use these values to approximate  $\mathbf{V} \cdot \nabla \phi = v_1 \phi_x + v_2 \phi_y + v_3 \phi_z$ , thus completing the spatial discretization.

The discretization of a more general Hamiltonian typically involves using the forward and backward differences  $\phi_x^+$ ,  $\phi_x^-$ ,  $\phi_y^+$ ,  $\phi_y^-$ ,  $\phi_z^+$  and  $\phi_z^-$  to evaluate a *numerical Hamiltonian* such as the Lax–Friedrichs or Godunov scheme [OF03]. Several examples are presented in Section 2.4.

#### 1.1.4 Hamilton–Jacobi WENO

To obtain a more accurate spatial discretization, we can use higher order finite difference schemes for the derivatives  $\phi_x^+$  and  $\phi_x^-$ . One approach is to use Hamilton–Jacobi weighted essentially non-oscillatory (WENO) schemes. These schemes use a dynamic combination of candidate stencils to approximate the derivatives based on the local smoothness properties of the solution. When the solution is smooth, the Hamilton–Jacobi WENO procedure produces high-order accurate derivatives; when the solution is nonsmooth or even discontinuous, the procedure uses smaller stencils biased away from the nonsmooth behaviour. This process is inherently nonlinear and even for linear PDE like (1.6), Hamilton–Jacobi WENO results in a nonlinear system of ODEs.

See Appendix B.1 for the Hamilton–Jacobi WENO procedure to be used in Chapter 2.

## 1.2 Time Stepping

Consider the system of ordinary differential equations (ODEs)

$$\frac{\partial}{\partial t} \mathbf{u}(t) = \mathbf{f}(\mathbf{u}(t)), \quad (1.7a)$$

$$\mathbf{u}(t_0) = \mathbf{u}^0, \quad (1.7b)$$

which, in our context, will typically arise from the spatial semi-discretization of a PDE, with  $\mathbf{u} \in \mathbb{R}^m$  where  $m$  is the number of points in the spatial grid. A numerical time-stepping scheme approximates  $\mathbf{u}(t)$  with a sequence of solution values  $\mathbf{u}^n$  at discrete steps  $n = 0, 1, 2, \dots$  with  $\mathbf{u}^n \approx \mathbf{u}(t_n)$ . Several classes of time discretizations will be used in this thesis; they are outlined over the next several sections.



### 1.2.1 Runge–Kutta methods

A simple numerical method for approximating the solution to (1.7) is the *forward Euler* method which computes the solution at the next step value  $\mathbf{u}^{n+1}$  in terms of the current step value  $\mathbf{u}^n$  as

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t \mathbf{f}(\mathbf{u}^n). \quad (1.8)$$

The result is a solution which is first-order accurate in time. Runge–Kutta methods can construct a higher order accurate numerical solution by employing intermediate *stage* values to approximate combinations of higher order terms in the Taylor series of the exact solution [HNW93, But03]. One possible representation of an explicit  $s$ -stage Runge–Kutta scheme is the Shu–Osher form [SO88]

$$\begin{aligned} \bar{\mathbf{u}}^{(0)} &= \mathbf{u}^n, \\ \bar{\mathbf{u}}^{(i)} &= \sum_{k=0}^{i-1} \left( \alpha_{ik} \bar{\mathbf{u}}^{(k)} + \Delta t \beta_{ik} \mathbf{f}(\bar{\mathbf{u}}^{(k)}) \right), \quad i = 1, \dots, s, \\ \mathbf{u}^{n+1} &= \bar{\mathbf{u}}^{(s)}, \end{aligned} \quad (1.9)$$

which expresses the next step of the numerical solution  $\mathbf{u}^{n+1}$  in terms of the solution at the current step  $\mathbf{u}^n$  and  $s$  intermediate stage values  $\bar{\mathbf{u}}^{(i)}$ . The values of the coefficients  $\alpha_{ik}$  and  $\beta_{ik}$  must be chosen to satisfy certain *order conditions*<sup>1</sup> for the Runge–Kutta method to achieve higher order accuracy [HNW93, But03].

In Chapter 2, the following Runge–Kutta method, originally by Fehlberg [Feh70] and now known as SSP(3,3), will be used.

#### SSP(3,3): three-stage, third-order Runge–Kutta scheme

$$\begin{aligned} \bar{\mathbf{u}}^{(1)} &= \mathbf{u}^n + \Delta t \mathbf{f}(\mathbf{u}^n), \\ \bar{\mathbf{u}}^{(2)} &= \frac{3}{4} \mathbf{u}^n + \frac{1}{4} \bar{\mathbf{u}}^{(1)} + \frac{1}{4} \Delta t \mathbf{f}(\bar{\mathbf{u}}^{(1)}), \\ \mathbf{u}^{n+1} &= \frac{1}{3} \mathbf{u}^n + \frac{2}{3} \bar{\mathbf{u}}^{(2)} + \frac{2}{3} \Delta t \mathbf{f}(\bar{\mathbf{u}}^{(2)}). \end{aligned}$$

This scheme has an additional *strong stability preserving* (SSP) property as discussed in Chapter 4. In fact it is the optimal three-stage third-order strong stability preserving

---

<sup>1</sup>These order conditions are generally presented for the Butcher tableau representation of the Runge–Kutta method, which is closely related to the Shu–Osher form [Hig05, FS05, KMG08].

Runge–Kutta method (see Section 4.1) [GS98, SO88]. Note, the scheme is explicit because each stage of the computation depends only on previously computed values. In contrast, in implicit Runge–Kutta methods, the computation of each stage in general depends on all the other stages. Implicit Runge–Kutta methods will be discussed further in Chapter 4. We next consider implicit methods in the class of *linear multistep methods*.

### 1.2.2 Implicit linear multistep methods

Instead of computing intermediate *stage* values, linear multistep methods use a weighed *linear* combination of the current and previous step values and function evaluations thereof, to advance to the next step. Chapter 3 uses implicit linear multistep methods for linear problems, that is, in the special case where the ODE system takes the form (1.4). In particular, the implicit Closest Point Method of Chapter 3 takes this form for linear PDEs. If  $\mathbf{M}$  is a spatial discretization of a diffusive operator such as the Laplacian or the biharmonic operator, then the system may be stiff and explicit time stepping may be inefficient due to the small time-step sizes required to ensure stability [Tho95]. In this case, implicit schemes—with their typically relaxed stepsize restrictions for stability [BF01, HW96]—will often be more efficient. Stability is discussed further in Section 1.3. The following implicit multistep schemes will be applied to linear ODEs of the form (1.4) in this thesis.

**Backward Euler** The simplest implicit linear multistep method is known as *backward Euler* and is also the implicit Runge–Kutta method

$$[\mathbf{I} - \Delta t \mathbf{M}] \mathbf{u}^{n+1} = \mathbf{u}^n, \quad (1.10)$$

where  $\mathbf{I}$  represents the  $m \times m$  identity matrix. The backward Euler method is implicit because a linear system solve is required to advance from the current step value  $\mathbf{u}^n$  to the next step  $\mathbf{u}^{n+1}$ . The scheme is first-order, has excellent stability properties and is self-starting.

**Crank–Nicolson** A second-order self-starting scheme known as Crank–Nicolson is

$$\left[ \mathbf{I} - \frac{\Delta t}{2} \mathbf{M} \right] \mathbf{u}^{n+1} = \mathbf{u}^n + \frac{\Delta t}{2} \mathbf{M} \mathbf{u}^n. \quad (1.11)$$

**BDF-2** The second-order backward difference formula (BDF-2) [Gea71] uses the current step value and one previous step value, and is given by

$$\left[ \mathbf{I} - \frac{2}{3} \Delta t \mathbf{M} \right] \mathbf{u}^{n+1} = \frac{4}{3} \mathbf{u}^n - \frac{1}{3} \mathbf{u}^{n-1}. \quad (1.12)$$

We note that this scheme must have  $n \geq 1$  and so this method is not self-starting. One solution is to compute the first step  $\mathbf{u}^1$  with the Crank–Nicolson scheme. Alternatively, one can use several smaller sub-steps of the backward Euler scheme.

**BDF-3** The third-order backward difference formula (BDF-3) uses the current step value and two previous step values, and is given by

$$\left[ \mathbf{I} - \frac{6}{11}\Delta t \mathbf{M} \right] \mathbf{u}^{n+1} = \frac{18}{11}\mathbf{u}^n - \frac{9}{11}\mathbf{u}^{n-1} + \frac{2}{11}\mathbf{u}^{n-2}. \quad (1.13)$$

Again the method is not self-starting, and now we require two starting steps  $\mathbf{u}^1$  and  $\mathbf{u}^2$ . In Chapter 3.6, the first step is computed with Crank–Nicolson followed by a step of BDF-2. We note that this starting procedure is only second-order; however a globally third-order solution is expected and may still be observed because only a small constant number of steps of the second-order methods are used.

Each of these multistep methods perform one inversion per step, solving a single linear system of the form  $\mathbf{A}\mathbf{u} = \mathbf{b}$  with  $\mathbf{A} = \mathbf{I} - \gamma\Delta t \mathbf{M}$  for some value of  $\gamma$  that depends on the particular method. Time-stepping matrices corresponding to the above methods are denoted as  $\mathbf{A}_{BE}$ ,  $\mathbf{A}_{CN}$ ,  $\mathbf{A}_{BDF2}$  and  $\mathbf{A}_{BDF3}$ .

### 1.2.3 IMEX schemes

Implicit-explicit (IMEX) time-discretization schemes can be applied to general problems involving a combination of stiff and nonstiff terms including the particular class of problem

$$\frac{\partial}{\partial t}\mathbf{u}(t) = \mathbf{f}(\mathbf{u}(t)) + \mathbf{M}\mathbf{u}(t),$$

which arises, for example, from the spatial discretization of the prototype problem

$$u_t = f(u) + u_{xx},$$

where the diffusion term  $u_{xx}$  often makes the problem stiff. An IMEX scheme treats the nonlinear but presumably nonstiff  $\mathbf{f}(\mathbf{u})$  term explicitly, and the stiff but linear  $\mathbf{M}\mathbf{u}$  term implicitly. For a wide variety of problems, IMEX schemes

1. avoid the small time steps required by explicit methods to accommodate the stiff terms,

2. avoid solving the nonlinear systems that occur in implicit methods because the nonlinear terms are treated explicitly.

We consider the following IMEX linear multistep schemes from [ARW95] for use on Turing pattern formation problems in Section 3.6.4; however, IMEX Runge–Kutta schemes [ARS97] could also be used.

**IMEX Euler** This first-order method is given by

$$[1 - \Delta t M] \mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t \mathbf{f}(\mathbf{u}^n). \quad (1.14)$$

The IMEX Euler method applies backward Euler to the linear term and forward Euler to the nonlinear term.

**SBDF-2** The second-order semi-implicit backward difference formula [ARW95] is

$$\left[1 - \frac{2}{3}\Delta t M\right] \mathbf{u}^{n+1} = \frac{4}{3}\mathbf{u}^n - \frac{1}{3}\mathbf{u}^{n-1} + \frac{4\Delta t}{3}\mathbf{f}(\mathbf{u}^n) - \frac{2\Delta t}{3}\mathbf{f}(\mathbf{u}^{n-1}). \quad (1.15)$$

The IMEX Euler scheme can be used as a starting method.

### 1.3 Consistency, Stability and Convergence

Assume we have a discrete method (be it a finite difference method, time-stepping scheme or otherwise) which is applied to a continuous problem. The *local truncation error* measures the defect in applying this discrete method to the exact solution of the continuous problem. The discrete method is said to be *consistent* if the local truncation error goes to zero as  $\Delta x \rightarrow 0$  and/or  $\Delta t \rightarrow 0$ . That is, if the discrete problem approximates the correct continuous problem [Hea97].

Note that consistency does *not* say that the *solution* of the discrete problem *converges* to the solution of the continuous problem. For that we also need *stability*: that the discrete method is not overly sensitive to perturbations in the solution. For certain classes of methods and well-posed problems, the *Lax Equivalence Theorem* states that stability and consistency imply convergence [Tho95].

The rate at which the local truncation error, expressed as a function of  $\Delta x$  and/or  $\Delta t$ , approaches zero as  $\Delta x$  and  $\Delta t$  approach zero is the *order of accuracy* of the method. For

example, the centered finite difference approximation of  $u_{xx}$  for smooth  $u$  in Section 1.1.2 has

$$\frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} = u_{xx} + \mathcal{O}(\Delta x^2),$$

so it is consistent and second-order accurate. For time-stepping schemes, if the local truncation error is  $\mathcal{O}(\Delta t^{q+1})$  then when  $\frac{\mathcal{O}(1)}{\Delta t}$  steps are performed, the resulting global error is  $\mathcal{O}(\Delta t^q)$ .

## 1.4 Strong Stability Preserving Time Stepping

In addition to being stable (in the sense of solutions not blowing up), it is sometimes desirable for a time-stepping scheme to possess additional nonlinear (or *strong*) stability properties. For example, if the exact solution of the ODE satisfies a *monotonicity* property (i.e.,  $\|\mathbf{u}(t_2)\| \leq \|\mathbf{u}(t_1)\|$  for  $t_2 > t_1$ ) in some norm or semi-norm, then it may be desirable that the numerical solution  $\mathbf{u}^n$  also satisfy a discrete monotonicity property  $\|\mathbf{u}^{n+1}\| \leq \|\mathbf{u}^n\|$ . Alternatively, if the ODE system is dissipative and solutions satisfied a *contractive* property (see Section 4.2.1), then one might require that the numerical solution also be contractive. Monotonicity properties, for example where  $\|\cdot\|$  is the total variation semi-norm (see Section 4.1), are often useful in the context of nonlinear PDEs to minimize the appearance of spurious oscillations in the numerical solution.

The SSP(3,3) scheme of Section 1.2.1, under a suitable time-step restriction, will preserve monotonicity properties that happen to be satisfied by the forward Euler method. The scheme is thus said to be *strong stability preserving*. Strong stability preserving methods are discussed further in Chapter 4, where a promising class of methods are investigated.

## 1.5 Interpolation

Suppose we know the value of a function  $u$  at a discrete set of grid points  $\mathbf{x}_i \in \mathbb{R}^d$ . *Interpolation* is the process of approximating  $u$  at a point  $\mathbf{x}$  using the values of  $u$  at the surrounding grid points  $\mathbf{x}_i$  [Hea97]. Interpolation plays an important role in the Closest Point Method, and in this section some useful techniques are outlined.

Figure 1.1: Example of 1D interpolation. The values of a function  $u$  are given at the four grid points and  $u(x)$  is approximated using linear ( $p = 1$ ) and cubic ( $p = 3$ ) polynomial interpolation.

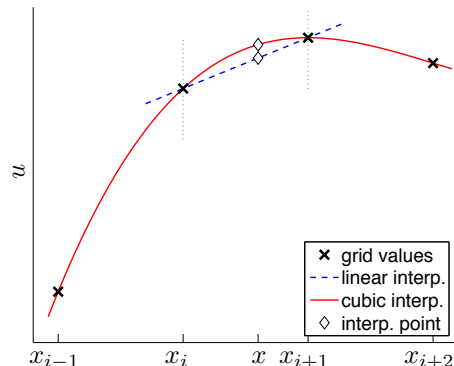
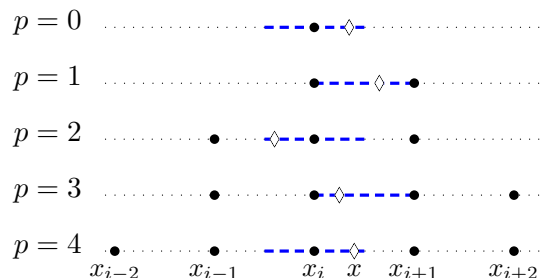


Figure 1.2: Choice of grid points for various degrees  $p$  of 1D interpolating polynomials. For each  $p = 0, \dots, 4$ , the selected grid points are indicated with  $\bullet$  and the point of interpolation  $\diamond$  lies anywhere on the dashed line. Note the difference between even and odd degree  $p$ .



### 1.5.1 1D polynomial interpolation

In one dimension, a common interpolation technique is to fit a degree  $p$  polynomial through the grid points surrounding  $x$  and evaluate at  $x$ . A linear ( $p = 1$ ) polynomial interpolant fits a straight line through neighbouring grid points  $x_i$  and  $x_{i+1}$  with  $x_i \leq x < x_{i+1}$ . Figure 1.1 shows an example of linear  $p = 1$  and cubic  $p = 3$  interpolation for some point data. Provided the underlying function is sufficiently smooth, degree  $p$  interpolation makes an error of  $\mathcal{O}(\Delta x^{p+1})$  where  $\Delta x$  is the spacing between grid points [BF01]. Degree  $p$  interpolation uses  $p + 1$  grid points around  $x$ , and these points should generally be chosen as close as possible to  $x$  to minimize the *Runge phenomenon* of oscillations at the far ends of the interpolant [Run01, BT04]. Figure 1.2 shows the choice of grid points used for various degree  $p$  interpolants. Various practical numerical techniques exist for performing polynomial interpolation: one particularly efficient method for the Closest Point Method is *barycentric Lagrange interpolation*, as discussed next.

### 1.5.2 Barycentric Lagrange interpolation

The *barycentric Lagrange formula* for the degree  $p$  interpolant through equispaced grid points  $x_0, \dots, x_p$  evaluated at interpolation point  $x$  is

$$u(x) \approx \frac{\sum_{j=0}^p \frac{w_j^{\text{bc}}}{x - x_j} u_j}{\sum_{j=0}^p \frac{w_j^{\text{bc}}}{x - x_j}}, \quad \text{where } w_j^{\text{bc}} = (-1)^j \binom{p}{j}, \quad (1.16)$$

where the  $w_j^{\text{bc}}$  are the barycentric weights (which are particularly simple here because of the equispaced grid). Despite the appearance of  $x - x_j$  in the denominators, the barycentric formula is stable for  $x$  close to  $x_j$ , although a straightforward special case must be used if  $x$  lies exactly at a grid point [BT04].

The weights in the barycentric formula are independent of the values  $u_i$  to be interpolated, depending only on the grid points  $x_i$  involved. This makes the form very efficient in the case where the interpolation scheme will be used repeatedly with different data values  $u_i$  but with the interpolation point  $x$  remaining fixed. Such is the case for the Closest Point Method, and in Chapter 3 we will pre-compute and store the weights  $w_j = \frac{w_j^{\text{bc}}}{x - x_j} / \sum_{i=0}^p \frac{w_i^{\text{bc}}}{x - x_i}$ . This allows the interpolation to be performed as an inner product of the weights  $w_j$  with the grid point data  $u_i$ .

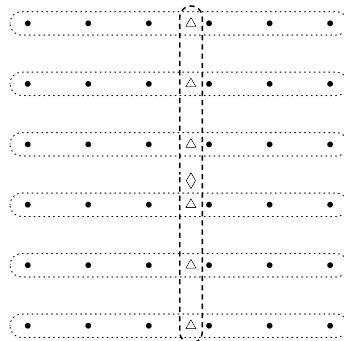
### 1.5.3 Higher dimensional interpolation

Higher dimensional interpolation is built in the standard fashion from one-dimensional interpolations. For example, two-dimensional degree  $p = 5$  interpolation is carried out by first interpolating six times in the  $x$ -direction to obtain six values which have  $x$ -coordinate values that agree with the interpolation point. One-dimensional interpolation is then carried out on these six points to get the desired interpolated value. See Figure 1.3 for an illustration. Three and higher dimensions are treated in a similar dimension-by-dimension manner.

### 1.5.4 Nonlinear interpolation

Barycentric Lagrange interpolation uses a linear combination of values at the grid points. This leads to efficient implementation, but in some cases it may be desirable to have the weights depend on the local behaviour of the grid data. For example, if the underlying

Figure 1.3: Using a 1D interpolation scheme to build a 2D interpolation routine. Suppose we want to approximate a value at the point  $\diamond$  by interpolating the values at the grid points  $\bullet$ . We begin by performing multiple 1D interpolations in the horizontal direction to obtain the values at the points  $\triangle$ . We then do a final 1D interpolation of the values at points  $\triangle$  to obtain a value at point  $\diamond$ .



function being interpolated is nonsmooth or discontinuous then we may wish to avoid using data from both sides of the discontinuity. In Section 2.2, this idea of dynamic stencils is used to derive a new weighted essentially non-oscillatory (WENO) interpolation scheme.

## 1.6 Surfaces

The importance of partial differential equations for modelling in the natural sciences and other areas cannot be overstated. Many of the problems of interest occur on surfaces rather than  $\mathbb{R}^2$  or  $\mathbb{R}^3$ . Examples include mammalian coat pattern formation, surfactant flow, fire and pest spread in forests on varying terrain, electrical signals on the cerebral cortex, wound healing and tissue regeneration, pyroclastic flows over topography, transport by surface currents in the oceans and large-scale atmospheric flows on the surface of the Earth. Surface problems also occur commonly in computer graphics, for example in texture mapping and synthesis, computer vision and image processing.

The Closest Point Method computes on surfaces such as a sphere, torus or general triangulated object. The method also works on curves in  $\mathbb{R}^2$  or higher dimensions and surfaces of arbitrary codimension such as the Klein bottle in  $\mathbb{R}^4$  (see Section 2.4.6). The term *surface* and symbol  $\mathcal{S}$  are used in this thesis to generically refer to all of these domains although the term *curve* is occasionally used when appropriate.

### 1.6.1 Partial differential equations on surfaces

Suppose we have a surface  $\mathcal{S}$  and a PDE defining a flow on the surface over time in terms of *intrinsic* in-surface differential operators [Küh05]. For example, the evolution of a function



$u : \mathcal{S} \rightarrow \mathbb{R}$  might be governed by

$$u_t = f(u, \nabla_{\mathcal{S}} u, \Delta_{\mathcal{S}} u),$$

where  $\nabla_{\mathcal{S}}$  is the intrinsic gradient, that is, the gradient that the “pure two-dimensional beings [who inhabit the surface] can recognize without any knowledge of the third dimension” [Küh05], and  $\Delta_{\mathcal{S}}$  is the *Laplace–Beltrami* operator, the intrinsic analogue of the Laplacian.

The PDE is time-dependent and we wish to propagate its solution on the surface over time. Any method to do this propagation will need a surface representation.

### 1.6.2 Representations of surfaces

One method of representing a surface is parameterization in terms of two local parameters (one in the case of a curve). For example a circle of radius  $R$  can be parameterized in terms of arc length  $s$  as  $\langle R \cos(\frac{s}{R}), R \sin(\frac{s}{R}) \rangle$ . For general surfaces, determining a parameterization is considerably more difficult.

It is sometimes straightforward to approximate the solution of a surface PDE using a parameterization. For example, consider the in-surface heat equation  $u_t = \Delta_{\mathcal{S}} u$  on a circle of radius  $R$ . If  $s_i$  are grid points of arc length, equispaced and separated by  $\Delta s$ , we can approximate the PDE as

$$\frac{\partial}{\partial t} u_i = \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta s^2},$$

with periodic boundary conditions, which is the same as the system (1.5). Again this process becomes more difficult on more general surfaces. In contrast, the Closest Point Method can be straightforwardly applied to general surfaces requiring only a closest point representation of  $\mathcal{S}$ , as discussed next.

### 1.6.3 Closest point representation of surfaces

An alternative approach to parameterizing the surface  $\mathcal{S}$  is to embed  $\mathcal{S}$  in a higher dimensional space  $\mathbb{R}^d$ . For example, a curve could be embedded in  $\mathbb{R}^2$  or  $\mathbb{R}^3$  and a surface could be embedded in  $\mathbb{R}^3$  or higher. *Embedding methods* are techniques for numerically solving PDEs or other processes on surfaces by computing on points in the embedding space  $\mathbb{R}^d$  instead of the surface itself. To do this, embedding methods must choose a representation of the surface in  $\mathbb{R}^d$ . The Closest Point Method for evolving PDEs on surfaces is an embedding method that uses a closest point representation of the surface.

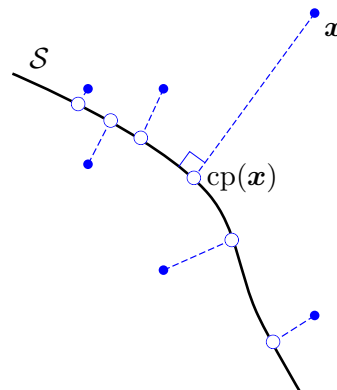


Figure 1.4: An example of the closest point function for a curve  $\mathcal{S}$  is shown for several example points in the embedding space  $\mathbb{R}^2$  indicated by  $\bullet$ . The corresponding closest points  $\text{cp}(\mathbf{x})$  on  $\mathcal{S}$  are indicated with  $\circ$ .

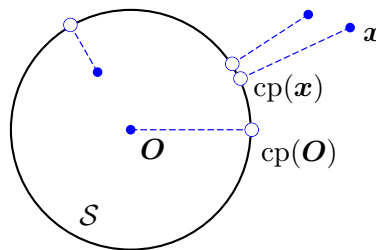


Figure 1.5: An example of the closest point function for a circle of radius  $R$  is shown for several points in  $\mathbb{R}^2$ . To ensure the closest point function is single-valued,  $\text{cp}(\mathbf{O})$  is set to an arbitrary point on the circle, in this case  $\langle R, 0 \rangle$ .

Let  $\mathcal{S}$  be a surface embedded in  $\mathbb{R}^d$  and define the *closest point function* as follows.

**Definition 1 (Closest point function)** For a given surface  $\mathcal{S}$ , the closest point function  $\text{cp} : \mathbb{R}^d \rightarrow \mathbb{R}^d$  takes point  $\mathbf{x} \in \mathbb{R}^d$  and returns a point  $\text{cp}(\mathbf{x}) \in \mathcal{S} \subset \mathbb{R}^d$  which is closest in Euclidean distance to  $\mathbf{x}$ . That is,  $\text{cp}(\mathbf{x}) = \min_{\mathbf{q} \in \mathcal{S}} \|\mathbf{x} - \mathbf{q}\|_2$ .

Figures 1.4 and 1.5 show examples of the closest point function. If  $\mathbf{x}$  is in a sufficiently small neighborhood of a smooth surface, then it will often have a unique closest point; however, if multiple points are closest to  $\mathbf{x}$ , then we define  $\text{cp}(\mathbf{x})$  to return an arbitrarily chosen closest point. For example, in Figure 1.5 where the surface is a circle of radius  $R$  centered at the origin, the closest point function is  $\text{cp}(\langle x, y \rangle) = \left\langle \frac{Rx}{x^2+y^2}, \frac{Ry}{x^2+y^2} \right\rangle$  provided  $\langle x, y \rangle \neq \langle 0, 0 \rangle$ . The origin is closest to any point on the circle and so we define  $\text{cp}(\langle 0, 0 \rangle)$  to be some arbitrary point on the circle.

A closest point representation of surface  $\mathcal{S}$  means we know the value of  $\text{cp}(\mathbf{x})$  for all  $\mathbf{x} \in \mathbb{R}^d$  (or at least on those points actually used in a computation). In this sense it is an *implicit representation* because  $\mathcal{S}$  is known only through the corresponding closest point function: we shall see that the Closest Point Method requires no knowledge of the surface other than a closest point representation. Some other embedding methods for surface PDEs (e.g. [CBMO02, Gre06]) make use of level set representations of the underlying surface,

which is also an implicit representation. Unlike level set representations, the closest point representation has the advantage of not requiring a notion of “inside/outside” allowing the straightforward representation of surfaces with boundaries or non-orientable surfaces (e.g., a Möbius strip). Surfaces of codimension-two or higher [Küh05] such as the Klein bottle in 4D (Section 2.4.6) or a filament in 3D (Section 3.6.2) can also be represented without additional complication. Thus, an important feature of the closest point representation is that it does not inherently impose any limitations on the geometry or dimension of surfaces that can be represented.

#### 1.6.4 Equivalence of gradients

Besides their flexibility for representing surfaces, closest point representations allow a natural extension of quantities  $u$  defined on the surface  $\mathcal{S}$  to points  $\mathbf{x}$  in the rest of the embedding space  $\mathbb{R}^d$  via  $u(\text{cp}(\mathbf{x}))$ . This process is known as a *closest point extension*.

**Definition 2 (closest point extension)** *Let  $\mathcal{S}$  be a surface embedded in  $\mathbb{R}^d$  and let  $\tilde{u} : \mathcal{S} \rightarrow \mathbb{R}$  be a scalar function defined over  $\mathcal{S}$ . Then the closest point extension of  $\tilde{u}$  is a function  $u : \mathbb{R}^d \rightarrow \mathbb{R}$  with  $u(\mathbf{x}) = \tilde{u}(\text{cp}(\mathbf{x}))$ .*

Closest point extensions result in functions which are constant in the direction normal to the surface, at least within a neighborhood of a smooth surface. This fact leads to simplified derivative calculations in the embedding space [RM08] which in turn can be easily approximated to accurately solve surface PDEs. To proceed, let  $\nabla$  denote the “standard” gradient in  $\mathbb{R}^d$  and let  $\nabla_{\mathcal{S}}$  denote the gradient intrinsic to the surface  $\mathcal{S}$ .

**Principle 1 (Gradients)** *For points  $\mathbf{x}$  on the surface,  $\nabla_{\mathcal{S}}u(\mathbf{x}) = \nabla u(\text{cp}(\mathbf{x}))$  because the function  $u(\text{cp}(\mathbf{x}))$  is constant in the normal direction and therefore only varies along the surface. In other words, at points  $\mathbf{x}$  on the surface, intrinsic surface gradients  $\nabla_{\mathcal{S}}u(\mathbf{x})$  are the same as gradients of  $u(\text{cp}(\mathbf{x}))$ .*

This principle will be all that we need to derive the embedding PDEs used in the Hamilton–Jacobi examples appearing in Chapter 2. For problems involving higher order derivatives such as those appearing in Chapter 3, a second principle also holds [RM08]:

**Principle 2 (Divergence)** *Let  $\nabla_{\mathcal{S}\cdot}$  denote the divergence operator intrinsic to the surface  $\mathcal{S}$  and let  $\mathbf{v}$  be any vector field on  $\mathbb{R}^d$  that is tangent at  $\mathcal{S}$  and also tangent at all surfaces*

displaced by a fixed distance from  $\mathcal{S}$  (i.e., all surfaces defined as level sets of the distance function to  $\mathcal{S}$ ). Then at points  $\mathbf{x}$  on the surface  $\nabla \cdot \mathbf{v}(\mathbf{x}) = \nabla_{\mathcal{S}} \cdot \mathbf{v}(\mathbf{x})$ .

Combinations of this and the gradient property may be made, to allow for very general laws for second-order differential operators, including the Laplace–Beltrami operator  $\Delta_{\mathcal{S}}$  and the level set equation for curvature motion and other nonlinear diffusion operators [RM08]. Indeed, even higher order and more general derivative replacements may be considered by carrying out multiple closest point extensions as first described in [RM08] and demonstrated in practice in Section 3.6.3 for the in-surface fourth-order biharmonic operator.

To help illustrate these ideas we provide two simple examples on a circle of radius  $R$ .

**Example 1** Consider the surface gradient, expressed in polar coordinates  $\nabla_{\mathcal{S}} u = 0\mathbf{e}_r + \frac{\partial u}{\partial s}\mathbf{e}_\theta$ , where  $s$  is the arc length. We have  $s = R\theta$ , therefore  $\frac{\partial u}{\partial s} = \frac{1}{R}u_\theta$  and

$$\nabla_{\mathcal{S}} u = \frac{1}{R}u_\theta\mathbf{e}_\theta.$$

Now applying the polar coordinate form of the standard gradient to  $u(\text{cp}(\mathbf{x}))$ , we have  $\nabla u(\text{cp}(\mathbf{x})) = u_r(\text{cp}(\mathbf{x}))\mathbf{e}_r + \frac{1}{r}u_\theta(\text{cp}(\mathbf{x}))\mathbf{e}_\theta$ . As described earlier,  $u(\text{cp}(\mathbf{x}))$  is constant in the direction normal to the surface which in this case is the radial direction (see also Figure 1.5), so  $u_r(\text{cp}(\mathbf{x})) = 0$ . Thus, for points  $\mathbf{x}$  on the surface (where  $\mathbf{x} = \text{cp}(\mathbf{x})$  and  $r = R$ ) we have

$$\nabla u(\text{cp}(\mathbf{x})) = \frac{1}{r}u_\theta(\text{cp}(\mathbf{x}))\mathbf{e}_\theta = \frac{1}{R}u_\theta(\mathbf{x})\mathbf{e}_\theta = \nabla_{\mathcal{S}} u(\mathbf{x}),$$

which is exactly Principle 1.

**Example 2** Consider, in polar coordinates, the surface Laplace–Beltrami operator  $\Delta_{\mathcal{S}} u = u_{ss} = \frac{1}{R^2}u_{\theta\theta}$  and the standard Laplacian operator  $\Delta u(\text{cp}(\mathbf{x})) = u_{rr}(\text{cp}(\mathbf{x})) + \frac{1}{r}u_r(\text{cp}(\mathbf{x})) + \frac{1}{r^2}u_{\theta\theta}(\text{cp}(\mathbf{x}))$ . Again,  $u(\text{cp}(\mathbf{x}))$  is constant in the radial direction so both radial derivatives are zero, and for  $\mathbf{x}$  on the surface

$$\Delta u(\text{cp}(\mathbf{x})) = \frac{1}{r^2}u_{\theta\theta}(\text{cp}(\mathbf{x})) = \frac{1}{R^2}u_{\theta\theta}(\mathbf{x}) = \Delta_{\mathcal{S}} u(\mathbf{x}).$$

And indeed, the equivalence of the Laplace–Beltrami operator  $\Delta_{\mathcal{S}} u$  and the Laplacian  $\Delta u(\text{cp}(\mathbf{x}))$  for points  $\mathbf{x}$  on the surface follows by combining Principles 1 and 2.

## 1.7 The Closest Point Method

The principles of *equivalence of gradients* and other differential operators yield a way of dealing with surface differential operators by evaluating the corresponding differential operator in the embedding space  $\mathbb{R}^d$ . Armed with these principles, we are now in a position to define the Closest Point Method. Central to this task is to determine a PDE, defined over the embedding space  $\mathbb{R}^d$ , to be used to generate a flow in  $\mathbb{R}^d$  corresponding to that of the surface PDE. Suppose that the surface PDE takes the form

$$u_t(t, \mathbf{x}) = f\left(t, \mathbf{x}, u(t, \mathbf{x}), \nabla_S u(t, \mathbf{x}), \nabla_S \cdot \left(\frac{\nabla_S u(t, \mathbf{x})}{|\nabla_S u(t, \mathbf{x})|}\right)\right), \quad (1.17a)$$

$$u(0, \mathbf{x}) = u_0(\mathbf{x}). \quad (1.17b)$$

for  $\mathbf{x} \in \mathcal{S}$  and  $t \geq 0$ . More general PDEs can be treated directly by the Closest Point Method, but this form includes many of the second-order flows that arise in geometric interface motion [Set99, OF03]. Based on the principles described in Section 1.6.4, and originally given in [RM08], we may replace the gradients and divergence operators by the standard Cartesian derivatives in the embedding space and accordingly

$$u_t(t, \mathbf{x}) = f\left(t, \text{cp}(\mathbf{x}), u(t, \text{cp}(\mathbf{x})), \nabla u(t, \text{cp}(\mathbf{x})), \nabla \cdot \left(\frac{\nabla u(t, \text{cp}(\mathbf{x}))}{|\nabla u(t, \text{cp}(\mathbf{x}))|}\right)\right), \quad (1.18a)$$

$$u(0, \mathbf{x}) = u_0(\text{cp}(\mathbf{x})), \quad (1.18b)$$

for  $\mathbf{x} \in \mathbb{R}^d$  and  $t \geq 0$ . The solutions of (1.17) and (1.18) will agree *at the surface* in the sense that if  $u^1(t, \mathbf{x})$  is a solution of (1.17) for  $\mathbf{x} \in \mathcal{S}$  and  $u^2(t, \mathbf{x})$  is a solution of (1.18) for  $\mathbf{x} \in \mathbb{R}^d$  then  $u^1(t, \mathbf{x}) = u^2(t, \mathbf{x})$  for  $t \geq 0$  and points on the surface  $\mathbf{x} \in \mathcal{S}$ .

At this point, we could try to discretize (1.18) directly: this idea is explored in Chapter 3 for implicit time stepping.

### 1.7.1 The explicit Closest Point Method

Notice that in (1.18) we begin with initial conditions  $u(0, \mathbf{x})$  which are a closest point extension of some surface data  $u_0$ , that is,  $u(0, \mathbf{x}) = u_0(\text{cp}(\mathbf{x}))$ . Because of this, the right-hand-side of (1.18) and that of the *embedding PDE*

$$u_t(t, \mathbf{x}) = f\left(t, \text{cp}(\mathbf{x}), u(t, \mathbf{x}), \nabla u(t, \mathbf{x}), \nabla \cdot \left(\frac{\nabla u(t, \mathbf{x})}{|\nabla u(t, \mathbf{x})|}\right)\right), \quad (1.19a)$$

$$u(0, \mathbf{x}) = u_0(\text{cp}(\mathbf{x})), \quad (1.19b)$$

agree *initially* at  $t = 0$  (although the solutions generally diverge at later times). That is, if we evaluate the right-hand-sides of (1.18a) and (1.19a) at  $t = 0$ , they will be equal for all  $\mathbf{x} \in \mathbb{R}^d$ . This is the key point of the explicit Closest Point Method because it suggests a way of explicitly treating (1.18) efficiently: starting from a closest point extension of the solution at time step  $t_n$ , take one forward Euler step (or *stage* of a higher order explicit Runge–Kutta scheme) of (1.19) to advance in time to  $\tilde{u}^{n+1}$ . After this evolution step,  $\tilde{u}^{n+1}$  will not be constant in a direction normal to the surface. To regain this property, we perform a closest point extension of  $\tilde{u}^{n+1}$  according to  $u^{n+1}(\mathbf{x}) = \tilde{u}^{n+1}(\text{cp}(\mathbf{x}))$ . This procedure gives an update which is constant in the direction normal to the surface, ensuring at  $t_{n+1}$  that (1.19a) will again agree with (1.18a) and hence with the original surface PDE (1.17). We can then repeat the process of alternating between time stepping (1.19) and performing closest point extensions to propagate the solution forward in time to any desired time.

The semi-discrete (discrete in time, continuous in space) explicit Closest Point Method with forward Euler time stepping would be as follows. First, using Principles 1 and 2, determine the embedding PDE (1.19) corresponding to the surface PDE (1.17): i.e., simply replace operators  $\nabla_S$  with  $\nabla$ ,  $\nabla_S \cdot$  with  $\nabla \cdot$ , etc. Second, perform a closest point extension of the initial conditions  $u^0(\mathbf{x}) = u^0(\text{cp}(\mathbf{x}))$  to determine the initial conditions for all points in the embedding space  $\mathbb{R}^d$ . Then use the following algorithm to advance from time  $t_n$  to time  $t_{n+1}$ .

**Algorithm 1: the explicit Closest Point Method**

1. Perform a forward Euler time step

$$\tilde{u}^{n+1} = u^n + \Delta t F \left( t_n, \text{cp}(\mathbf{x}), u^n, \nabla u^n, \nabla \cdot \left( \frac{\nabla u^n}{|\nabla u^n|} \right) \right).$$

2. Perform a closest point extension for each point in the embedding space  $\mathbf{x} \in \mathbb{R}^d$

$$u^{n+1}(\mathbf{x}) = \tilde{u}^{n+1}(\text{cp}(\mathbf{x})).$$

For higher order *explicit* Runge–Kutta methods, a closest point extension is employed following each *stage* of the Runge–Kutta scheme. That is, steps 1 and 2 would be repeated three times for the three-stage SSP(3,3) scheme.

Now, we consider spatial discretization and indeed, great flexibility is available in choosing a spatial discretization. Similar to [RM08] we will select finite difference methods on

banded, but regular, Cartesian grids. Indeed one of the benefits of the Closest Point Method is that the evolution step (step 1) is approximated using standard methods in the embedding space. Moreover the embedding PDE does not involve any projections or other surface-specific modifications; it is simply the corresponding PDE in the embedding space. This is very convenient since it implies that standard, well-understood algorithms in  $\mathbb{R}^3$  can be modified straightforwardly to accommodate surface flows. Indeed, this is exactly the approach taken in Section 2.4 where standard Hamilton–Jacobi algorithms are reused.

Note that the use and meaning of the embedding PDE is fundamentally different for the level set surface representation of PDEs on surfaces of [CBMO02] compared to the Closest Point Method. In a level set approach the embedding PDE gives the solution at the surface for all times. In the Closest Point Method the embedding PDE only gives a valid evolution initially and for one explicit time step (or stage in a Runge–Kutta method). Thus the extension step is necessary to ensure the consistency of the method with the exact solution on the surface.

### 1.7.2 Closest point extension and interpolation

The closest point extension is an interpolation step because, although  $\mathbf{x}$  is a grid point in a regular Cartesian grid,  $\text{cp}(\mathbf{x})$  likely will not be. Figure 1.6 illustrates an example where bilinear interpolation is used to estimate a value for  $u(\text{cp}(\mathbf{x}))$  from a stencil of the four neighboring grid points. One could use a larger interpolation stencil to increase the accuracy of the interpolation, for example using Lagrange interpolation as described in Section 1.5. Indeed, this is how interpolation was originally done in [RM08] and in this thesis in Chapter 3.3. Section 3.4 in particular discusses the degree of polynomial interpolation required for convergence.

In Chapter 2.4, a new high-order WENO interpolation scheme is constructed using dynamic stencils.

### 1.7.3 Banding and bandwidth

The Closest Point Method does not introduce any artificial boundaries at the edge of the computational band to carry out banded calculations near the surface. The method merely computes on a band of points around the surface which is wide enough so that all values in the interpolation stencil have been accurately evolved (which depends on the spatial

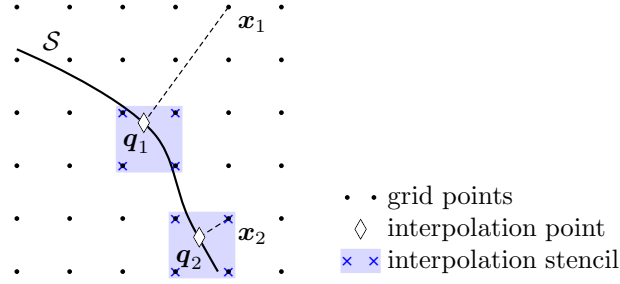


Figure 1.6: Closest point extensions for grid points  $\mathbf{x}_1$  and  $\mathbf{x}_2$  where  $\mathbf{q}_1 = \text{cp}(\mathbf{x}_1)$  and  $\mathbf{q}_2 = \text{cp}(\mathbf{x}_2)$ . In this case, low-order bilinear interpolation is used to estimate a value for  $u(\mathbf{q}_i) = u(\text{cp}(\mathbf{x}_i))$  from a stencil of the four neighboring grid points to  $\mathbf{q}_i$ .

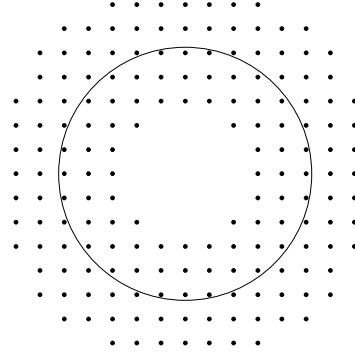


Figure 1.7: Example of the computational band used by the Closest Point Method where the curve  $\mathcal{S}$  is a circle. The grid points  $\mathbf{x}_i$  are indicated by  $\bullet$ .

discretization stencil and the degree of interpolation as discussed in Sections 2.3 and 3.5). As a general principle, artificial boundaries should be avoided as they can lead to a degradation of accuracy even in simple diffusive problems [Gre06].

An example of an appropriate computational band is shown in Figure 1.7 where the curve  $\mathcal{S}$  is a circle embedded in 2D.

## 1.8 Outline of Thesis Contributions

The remainder of this thesis presents new contributions to the Closest Point Method (Chapters 2 and 3) and an investigation and analysis of a class of time-stepping schemes (Chapter 4).

Chapter 2 covers the application of the explicit Closest Point Method to the problem of level set equations on surfaces. A new weighted essentially non-oscillatory (WENO)



interpolation scheme is derived, and many examples are presented. Numerical convergence studies demonstrate the first high-order results of the Closest Point Method.

Chapter 3 derives a new implicit Closest Point Method which uses implicit time stepping. A matrix operator interpretation of the evolution and extension steps of the Closest Point Method is presented. Using this notation, the implicit Closest Point Method is very easy to formulate, essentially by forming the product of two matrices. Numerical convergence studies on the in-surface heat equation and biharmonic problems confirm that the implicit Closest Point Method can also produce high-order results. Examples on a variety of surfaces of image blurring, heat-flow modelling and pattern formation using implicit-explicit (IMEX) time stepping are presented.

Chapter 4 investigates the class of unconditionally contractive diagonally split Runge–Kutta methods which profess to offer high-order accuracy combined with a desirable non-linear contractivity property. Through numerical tests and an analysis of stage order, it is demonstrated that these schemes suffer from order reduction which makes them inefficient in practice.

Finally Chapter 5 presents some conclusions of the work.

## Chapter 2

# Level Set Equations on Surfaces

Level set methods have been used extensively for interface problems in  $\mathbb{R}^2$  and  $\mathbb{R}^3$ , but there are also many applications which could benefit from level-set based techniques extended to surfaces. In this chapter we consider the treatment of level set equations on surfaces via the explicit Closest Point Method. The main modification is to introduce a Weighted Essentially Non-Oscillatory (WENO) interpolation step into the Closest Point Method. This interpolation, in combination with standard WENO for Hamilton–Jacobi equations, gives high-order results (up to fifth-order) on a variety of smooth test problems including passive transport, normal flow and redistancing. The algorithms proposed are straightforward modifications of standard codes, are carried out in the embedding space in a well-defined band around the surface and retain the robustness of the level set method with respect to the self-intersection of interfaces. Numerous examples are provided to illustrate the flexibility of the method with respect to geometry.

Most of the contents of this chapter appear in [MR08].

### 2.1 Level Set Methods

Application of the level set method [OS88] involves evolving a curve in  $\mathbb{R}^2$  or a surface in  $\mathbb{R}^3$  to solve a problem of interest. This curve or surface—the *interface*—is represented as the zero contour of a level set function  $\phi$  (see Figure 2.1(a)). A principal strength of the level set method comes from its ability to handle changes in topology of the evolving interface, i.e., interfaces break apart or merge naturally, and without the need for special code or instructions to detect or treat the shape changes as they occur. A second, and also key,

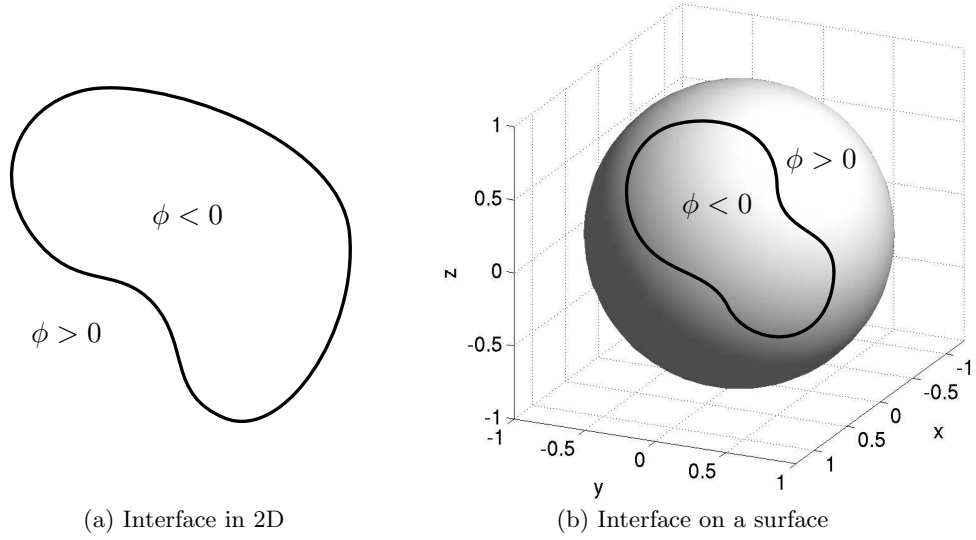


Figure 2.1: Level set representation of an interface in 2D (a) and on the surface of a sphere (b). Inside of the interface corresponds to  $\phi < 0$  and outside corresponds to  $\phi > 0$ . The interface itself corresponds to  $\phi = 0$ . In (b),  $\phi$  is evaluated on the surface of the sphere.

benefit is that the discretization of the underlying *level set equation* (of Hamilton–Jacobi type) can be carried out using well-known, accurate and reliable discretization techniques, such as the weighted essentially non-oscillatory (WENO) methods [LOC94, JS96, JP00] described in Section 1.1.4. Taken together, these benefits have contributed to a widespread adoption of level-set based techniques in different disciplines [OF03, Set99].

Level set methods have primarily been used to treat evolving interfaces in  $\mathbb{R}^2$  and  $\mathbb{R}^3$ . However, evolving level set equations on general domains would give a way of robustly capture the motion of interfaces on curved surfaces (see Figure 2.1(b)). Such an extension would be compelling because it could be exploited to generalize existing level set applications to curved surfaces. For example, suppose one wished to detect or “mark” objects appearing on a surface, a process known as *segmentation*. By extending level set methods to surfaces, we gain the possibility of solving this problem by simply transferring existing level set methods for segmentation to the case of surfaces. This approach becomes even more compelling if the algorithms for surface flows end up being based on existing codes for standard two- and three-dimensional flows. Indeed, we shall see that this is the case with the Closest Point Method.

### 2.1.1 Computing level sets on surfaces

The problem of level set methods on surfaces is not new, and one interesting method for evolving interfaces on surfaces was proposed by Cheng et al. [CBMO02]. In their approach, a level set representation of the underlying surface was taken, with the evolving interface being represented by the intersection of two level set functions. The level set evolution equation for  $\phi$  made use of standard gradients followed by projections onto the surface. Thus, the method evolved a level set PDE in  $\mathbb{R}^3$ , and, at any time, gave the position of the interface on the surface as the zero contour of  $\phi$  on the surface. See [CBMO02] for further details on the method as well as a selection of examples using the method.

Another way of developing a method to evolve interfaces on surfaces is to start from a level set equation defined on a surface, e.g., a Hamilton–Jacobi equation of the form

$$\phi_t + H(t, \mathbf{x}, \phi, \nabla_S \phi) = 0, \quad (2.1a)$$

$$\phi(0, \mathbf{x}) = \phi_0(\mathbf{x}), \quad (2.1b)$$

or some curvature-dependent generalization of this, and to solve it with some existing strategy for evolving PDEs on surfaces. For example, one might apply either the method of Bertalmío et al. [BCOS01] or that of Greer [Gre06] to treat the surface PDE. These methods use a level set representation of the surface and replace surface gradients by standard gradients and projection operators in  $\mathbb{R}^3$  to get an *embedding PDE* which is defined throughout time and space and agrees with the surface evolution of  $\phi$  on the surface. This leads to similar or the same PDEs as those appearing in [CBMO02], and will therefore be very similar in character to the methods described there.

In this chapter, level set equations of Hamilton–Jacobi type (2.1) are evolved according to the Closest Point Method. As discussed in Chapter 1, the Closest Point Method has a number of properties that make it quite attractive for solving level set equations on surfaces. First of all, it takes the underlying surface representation to be a closest point representation. This allows it to treat PDEs on surfaces that have boundaries, lack any clearly defined inside/outside or are of arbitrary codimension. Similar to level-set based methods, the method uses an embedding PDE defined in the embedding space (e.g.,  $\mathbb{R}^3$ ). However, the meaning and use of the embedding PDE is fundamentally different, because it is only valid initially, and requires an *extension step* (i.e., step 2 of the Closest Point Method) to ensure consistency. A desirable property of the Closest Point Method is that it works on sharply defined bands around the surface of interest without any loss of accuracy whatsoever.

Finally, we note that the method, in its explicit form, leads to embedding PDEs which are simply the PDEs of the corresponding flow in the embedding space. This last advantage means that with the insertion of a simple extension step we can reuse three-dimensional level set codes without any other modifications to obtain the motion of level sets on surfaces. Note that this chapter does not consider curvature-driven flows which involve higher order differential operators; such motions have been treated successfully using the Closest Point Method with a central difference spatial discretization in [RM08]. Additionally, the implicit Closest Point Method in Chapter 3 is also appropriate for curvature dependent flows as it allows implicit time stepping to avoid restrictive time steps.

A crucial step in applying the Closest Point Method to solve level set equations on surfaces is to design an appropriate extension step. This chapter considers a new extension based on a WENO interpolation which is sixth order in smooth regions of the solution and formally fourth order elsewhere. The approach has all the practical advantages of the Closest Point Method: flexibility when selecting a surface, efficiency with respect to banding and extreme simplicity of implementation. It is emphasized that while the new extension procedure will be used here for Hamilton–Jacobi equations, it also could be valuable for treating more general PDEs if high-order accuracy is desired but the PDE or the underlying surface is somewhere nonsmooth or marginally resolved.

Section 2.2 derives the new interpolation technique which is inspired by previous WENO methods. The section also includes details on situations where WENO interpolation will be preferred over standard fixed-stencil Lagrange interpolation.

## 2.2 WENO Interpolation

Weighted Essentially Non-Oscillatory (WENO) spatial discretizations [LOC94, JS96, JP00, Lan98] are well-studied and commonly used for the evolution of discontinuous or nonsmooth solutions to PDEs. Standard WENO discretizations (Section 1.1.4 and Appendix B) are used in Section 2.4 for the evolution step of the Closest Point Method (step 1 in Algorithm 1).

However, the extension step of the Closest Point Method (step 2 in Algorithm 1) requires an interpolation scheme to approximate  $\phi(\text{cp}(\mathbf{x}))$  from grid points near the surface. High-degree fixed-stencil Lagrange interpolation is derived for smooth data [BF01] and may

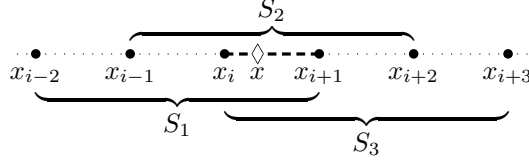


Figure 2.2: The one-dimensional WENO interpolation grid, for  $x \in [x_i, x_{i+1})$  and three candidate stencils  $S_1$ ,  $S_2$  and  $S_3$ .

encounter difficulties from either nonsmooth  $\phi$  or from nonsmooth or poorly resolved surfaces. As an alternative, in this section a WENO-based interpolation is derived in one dimension and in multiple dimensions. This derivation is followed by some numerical experiments which illustrate that WENO interpolation can give very good results even when fixed-stencil Lagrange interpolation fails or gives undesirable results.

WENO interpolation was considered in [SS03] to interpolate between subdomains for a multidomain WENO finite difference calculation for hyperbolic conservation laws. However, in [SS03] one of the candidate stencils corresponds to an extrapolation rather than an interpolation (as can be seen from Fig. 2 of [SS03]). In this chapter, WENO interpolation schemes in which all candidate polynomials are interpolants are derived and studied. The question of whether improved results can be obtained by allowing some extrapolation in the candidate polynomials will be addressed in future studies.

### 2.2.1 One-dimensional WENO interpolation

We consider two interpolation schemes. The first is formally sixth order in smooth regions.

#### Sixth-order WENO interpolation

Consider the 1D interpolation problem (Figure 2.2): given the six points  $x_{i-2}$ ,  $x_{i-1}$ ,  $x_i$ ,  $x_{i+1}$ ,  $x_{i+2}$ ,  $x_{i+3}$ , corresponding data  $f_{i-2}$ ,  $f_{i-1}$ ,  $f_i$ ,  $f_{i+1}$ ,  $f_{i+2}$ ,  $f_{i+3}$  and a value of  $x \in [x_i, x_{i+1})$ , we want to estimate  $f(x)$ .

We begin with three candidate interpolants  $p_1(x)$ ,  $p_2(x)$  and  $p_3(x)$  corresponding to cubic polynomial fits to the data given on each of the three candidate stencils (see Figure 2.2)  $S_1 = \{x_{i-2}, \dots, x_{i+1}\}$ ,  $S_2 = \{x_{i-1}, \dots, x_{i+2}\}$  and  $S_3 = \{x_i, \dots, x_{i+3}\}$ . We combine these to

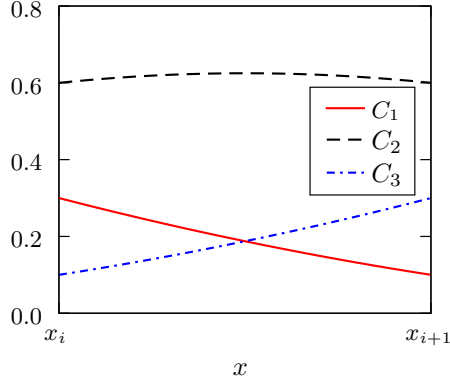


Figure 2.3: Values of the ideal WENO weights  $C_i(x)$  for  $x \in [x_i, x_{i+1})$ .

give the WENO interpolant

$$I_{\text{WENO6}}(x) = w_1(x)p_1(x) + w_2(x)p_2(x) + w_3(x)p_3(x),$$

where  $w_i(x), i = 1, 2, 3$  are the required weights (still to be determined). In a smooth problem, all the point data could be used to obtain an interpolation which is as high order as possible, i.e., that agrees with the degree five interpolating polynomial through all six points. These “ideal” weights  $C_i, i = 1, 2, 3$  are given by

$$\begin{aligned} C_1(x) &= \frac{(x_{i+2} - x)(x_{i+3} - x)}{20\Delta x^2}, \\ C_2(x) &= \frac{(x_{i+3} - x)(x - x_{i-2})}{10\Delta x^2}, \\ C_3(x) &= \frac{(x - x_{i-2})(x - x_{i-1})}{20\Delta x^2}, \end{aligned}$$

and their values on the interval  $x \in [x_i, x_{i+1})$  are shown in Figure 2.3. Note that unlike WENO for hyperbolic conservation laws [LOC94, JS96] and WENO for Hamilton–Jacobi problems [JP00], here the interpolation point  $x$  is not fixed and the values of the ideal weights depend on  $x$ . Still, these  $C_i(x)$  are completely analogous to the well-known “ $\frac{1}{10}, \frac{6}{10}, \frac{3}{10}$ ” weights in the latter work (see Appendix B).

In nonsmooth regions, at least one of the interpolants  $p_i(x), i = 1, 2, 3$  will be superior to an interpolation with the “ideal” weights because of the Runge phenomenon [Run01]—highly oscillatory results associated with high-degree polynomial interpolation. To decide which stencils to use, we compute a *smoothness indicator* for each interpolant. We take the smoothness indicator  $IS_i$  for interpolant  $p_i$  as “a sum of squares of scaled  $L^2$  norms of all the

derivatives of the [interpolant  $p_i$ ] over the interval [of interpolation]” [Shu97]. Specifically

$$IS_i = \sum_{j=1}^3 \int_{x_i}^{x_{i+1}} (\Delta x)^{2j-1} \left( \frac{d^j p_i(x)}{dx^j} \right)^2 dx, \quad i = 1, 2, 3. \quad (2.2)$$

If a particular interpolant exhibits rapid change on the interval  $(x_i, x_{i+1})$  compared to the other two interpolants, then its derivatives will be larger in magnitude on that interval, which in turn increases the corresponding smoothness indicator (2.2). Smooth interpolants—those that are desirable for use in the interpolation—will exhibit less drastic changes in their derivatives and thus minimize (2.2). If all three candidate interpolants are smooth, then all three smoothness indicators will have similar (small) values. The smoothness indicators can be worked out as

$$\begin{aligned} IS_1 &= (-3579f_{i+1}f_i + 2634f_{i+1}f_{i-1} - 683f_{i+1}f_{i-2} - 6927f_i f_{i-1} + 1854f_i f_{i-2} \\ &\quad - 1659f_{i-1}f_{i-2} + 814f_{i+1}^2 + 4326f_i^2 + 2976f_{i-1}^2 + 244f_{i-2}^2)/180, \\ IS_2 &= (-3777f_{i+1}f_i + 1074f_{i+1}f_{i-1} - 1269f_i f_{i-1} + 1986f_{i+1}^2 + 1986f_i^2 + 244f_{i-1}^2 \\ &\quad + 244f_{i+2}^2 - 1269f_{i+2}f_{i+1} + 1074f_{i+2}f_i - 293f_{i+2}f_{i-1})/180, \\ IS_3 &= (-3579f_{i+1}f_i + 4326f_{i+1}^2 + 814f_i^2 + 2976f_{i+2}^2 + 244f_{i+3}^2 - 683f_{i+3}f_i \\ &\quad - 6927f_{i+2}f_{i+1} + 2634f_{i+2}f_i - 1659f_{i+3}f_{i+2} + 1854f_{i+3}f_{i+1})/180. \end{aligned}$$

Note, as expected that the smoothness indicators do not depend on the particular point of interpolation  $x$  because they measure a property of the interpolant candidates themselves.

The computation of the weights is carried out using the smoothness indicators as in the standard WENO procedure by first calculating

$$\alpha_i(x) = \frac{C_i(x)}{(\varepsilon + IS_i)^2}, \quad i = 1, 2, 3,$$

where  $\varepsilon$  is a small parameter to prevent division-by-zero in the case when all  $IS_i \approx 0$ ; in this chapter,  $\varepsilon = 1 \times 10^{-6}$  is used in all calculations. Finally, the weights are

$$w_i(x) = \frac{\alpha_i(x)}{\alpha_1(x) + \alpha_2(x) + \alpha_3(x)}, \quad i = 1, 2, 3.$$

#### Fourth-order WENO interpolation

A fourth-order (in smooth regions) WENO interpolation scheme can also be constructed based on two quadratic interpolant candidates. In this case, we have the four points



$x_{i-1}, x_i, x_{i+1}, x_{i+2}$  and corresponding data  $f_{i-1}, f_i, f_{i+1}, f_{i+2}$  and again want to estimate  $f(x)$  for  $x \in [x_i, x_{i+1})$ . The two candidate interpolants are

$$\begin{aligned} p_1(x) &= f_i + \frac{f_{i+1}-f_{i-1}}{2\Delta x}(x-x_i) + \frac{f_{i+1}-2f_i+f_{i-1}}{2\Delta x^2}(x-x_i)^2, \\ p_2(x) &= f_i + \frac{-f_{i+2}+4f_{i+1}-3f_i}{2\Delta x}(x-x_i) + \frac{f_{i+2}-2f_{i+1}+f_i}{2\Delta x^2}(x-x_i)^2, \end{aligned}$$

with ideal weights  $C_1(x) = \frac{x_{i+2}-x}{3\Delta x}$  and  $C_2(x) = \frac{x-x_{i-1}}{3\Delta x}$ , and smoothness indicators

$$\begin{aligned} IS_1 &= (26f_{i+1}f_{i-1} - 52f_if_{i-1} - 76f_{i+1}f_i + 25f_{i+1}^2 + 64f_i^2 + 13f_{i-1}^2) / 12, \\ IS_2 &= (26f_{i+2}f_i - 52f_{i+2}f_{i+1} - 76f_{i+1}f_i + 25f_i^2 + 64f_{i+1}^2 + 13f_{i+2}^2) / 12. \end{aligned}$$

The fourth-order WENO interpolant is thus

$$I_{\text{WENO4}}(x) = w_1(x)p_1(x) + w_2(x)p_2(x),$$

where  $w_1(x)$  and  $w_2(x)$  are calculated from the smoothness indicators by  $\alpha_i(x) = \frac{C_i(x)}{(\varepsilon + IS_i)^2}$  and  $w_i(x) = \frac{\alpha_i(x)}{\alpha_1(x) + \alpha_2(x)}$ ,  $i = 1, 2$ .

### Advantages of WENO interpolation

Both the fourth-order and sixth-order WENO interpolation routines are constructed so that wherever the ideal weights are chosen (i.e.,  $w_i(x) = C_i(x)$ ), the results are identical to using fixed-stencil interpolating polynomials through all candidate points. Therefore, in this case, they also are identical to the fixed-stencil Lagrange interpolation procedure used in previous Closest Point Method works [RM08, MR07]. In nonsmooth problems, however, stencils corresponding to smooth regions are automatically selected. Figure 2.4 shows one such example, where a one-dimensional function is reconstructed using interpolation in a cell adjacent to a discontinuity. Lagrange interpolation based on the six data points gives a spurious oscillation in the cell of interest, since it interpolates across the discontinuity. On the other hand, with WENO interpolation the smoothness indicators  $IS_2$  and  $IS_3$  are very large, resulting in small weights  $w_2$  and  $w_3$ , and thus the data from the rightmost two data points give a negligible contribution to the interpolation. These weights lead to the desired non-oscillatory result.

Another type of problem for which WENO-based interpolation is expected to give superior results arises when the underlying surface is nonsmooth or is marginally resolved by the grid of the embedding space. The latter possibility is investigated in Section 2.4.4,



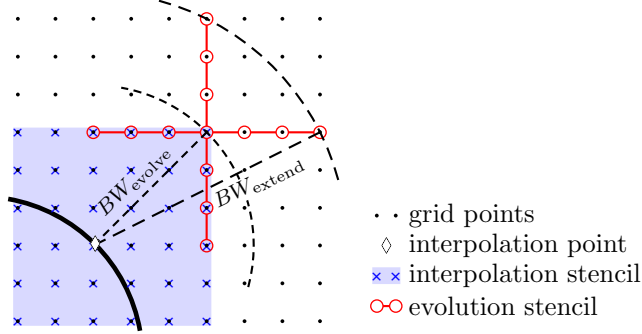


Figure 2.5: The minimum bandwidths in 2D involved in a Closest Point Method computation for a surface in the vicinity of point  $\diamond$ .

### 2.3.1 Bandwidth upper bounds

We begin by determining upper bounds on the bandwidth. Consider an  $\mathbb{R}^d$  embedding space and for simplicity assume that the grid spacing is  $\Delta x$  in all  $d$  dimensions. The WENO interpolation stencil is a  $d$ -dimensional hypercube (see Section 1.5.3) where each side has a width of  $5\Delta x$ . Considering Figure 2.5, the *evolution step* can therefore be carried out on a set of grid points which lie within a distance of

$$BW_{\text{evolve}} = \sqrt{3^2 + \dots + 3^2} \Delta x = 3\sqrt{d} \Delta x, \quad (2.4)$$

from the surface, i.e., the diagonal distance across a  $d$ -dimensional hypercube with side widths  $3\Delta x$ .

To generate accurate values on points inside the interpolation stencil at  $t_{n+1}$ , the evolution stencil needs accurate values inside its own finite difference stencil at  $t_n$ . Thus the interpolation step at  $t_n$  must update all grid points within the evolution stencil applied at every grid point in the interpolation stencil around every point on the surface. In our case, the fifth-order Hamilton–Jacobi WENO finite difference scheme has a stencil consisting of a “hypercross” (see Figure 2.5) where each arm has width  $6\Delta x$  (see Section 1.1.4 and Appendix B). From the furthest corner of the interpolation stencil hypercube, the evolution stencil extends in grid-aligned directions; the extension step should therefore be performed on a bandwidth of

$$BW_{\text{extend}} = \sqrt{3^2(d-1) + (3+3)^2} \Delta x. \quad (2.5)$$

Table 2.1: Sufficient bandwidths for the Closest Point Method evolution and interpolation steps in various embedding dimensions.

Dimension	$BW_{\text{evolve}}$	$BW_{\text{extend}}$
2D	$4.2426\Delta x$	$6.7082\Delta x$
3D	$5.1962\Delta x$	$7.3485\Delta x$
4D	$6\Delta x$	$7.9373\Delta x$

Values of the bandwidths (2.4) and (2.5) for two, three and four dimensions are tabulated in Table 2.1. Bandwidth calculations for standard Lagrange interpolation can be carried out using similar considerations; see [RM08] for further details.

When defining the computational domain based on these upper bounds, note there may be some points which are inside  $BW_{\text{evolve}}$  but are outside the union of the interpolation stencils around each closest point on the surface. These points therefore have, at most, a negligible effect<sup>1</sup> on the solution as the calculation proceeds. Evolving on these points thus introduces redundant computation. Similarly, there may be points inside  $BW_{\text{extend}}$  which are not needed for the evolution; they also introduce additional redundant calculations.

### 2.3.2 The stencil set approach

As noted above, using the bandwidth upper bounds (2.4) and (2.5) to define the computational bands may include unnecessary points thus increasing the computational expense without improving accuracy. An alternate approach is to explicitly construct the sets of points which define the evolution and extension bands. Let  $S_{\text{evolve}}$  be the set of nodes in the evolution band and  $S_{\text{extend}}$  be the set of nodes in the extension band. These sets are then determined by the following algorithm.

#### Algorithm 2: Identifying the stencil sets

- Initialize both sets to the empty set, i.e., set  $S_{\text{evolve}} = \emptyset$  and  $S_{\text{extend}} = \emptyset$ .
- Loop over all grid nodes  $\mathbf{x}$  in the embedding space that are within a distance  $BW_{\text{extend}}$  of the surface and for each:
  - Loop over all grid nodes  $\mathbf{y}$  in the interpolation stencil surrounding  $\text{cp}(\mathbf{x})$ :

---

<sup>1</sup>In principle, these points could influence the artificial dissipation parameters appearing in schemes such as the (global) Lax–Friedrichs (LF) scheme [CL84, OF03] or the local Lax–Friedrichs (LLF) scheme [SO88, OF03]. Although such effects were not observed in practice, the stencil set approach discussed next avoids this issue altogether.

Table 2.2: Numerical verification of the bandwidths for a 3D Closest Point Method calculation. The particular calculation or the values of the error are not important here<sup>†</sup>; we note only that the errors are identical (down to the bit-pattern of the double precision floating point numbers) provided that the bandwidths are taken larger than the minimums from Table 2.1.

Banding strategy		Error	Points in band	
$BW_{\text{evolve}}$	$BW_{\text{extend}}$		Evolution	Extension
No banding		$3.546949572342186954 \times 10^{-8}$	1030301	1030301
$10\Delta x$	$10\Delta x$	$3.546949572342186954 \times 10^{-8}$	165335	165335
$5.2\Delta x$	$7.35\Delta x$	$3.546949572342186954 \times 10^{-8}$	82830	119026
$5.15\Delta x$ <sup>‡</sup>	$7.35\Delta x$	$3.546949570771058627 \times 10^{-8}$	82326	119026
$5.2\Delta x$	$7.3\Delta x$ <sup>*</sup>	$3.546949572069116105 \times 10^{-8}$	82830	117970
Stencil set bands		$3.546949572342186954 \times 10^{-8}$	70296	111412

<sup>†</sup> In the interests of reproducibility, the computation is the same as Table 2.4 on a grid with  $\Delta x = 0.04$  with error measured as the maximum absolute value of  $\phi$  along the theoretical interface location with  $x \geq 0$  and  $z \geq 0$ .

<sup>‡</sup>  $5.15\Delta x$  is less than the minimum bandwidth  $BW_{\text{evolve}}$ .

<sup>\*</sup>  $7.3\Delta x$  is less than the minimum bandwidth  $BW_{\text{extend}}$ .

- Add node  $\mathbf{y}$  to the evolution band by setting  $S_{\text{evolve}} = \{\mathbf{y}\} \cup S_{\text{evolve}}$ .
- Let  $K$  be the set of grid nodes appearing in the evolution stencil for  $\mathbf{y}$ . Add this set to the extension band by setting  $S_{\text{extend}} = K \cup S_{\text{extend}}$ .

After this procedure,  $S_{\text{evolve}}$  and  $S_{\text{extend}}$  are the sets over which the evolution and extension steps should be carried out for the Closest Point Method.

Table 2.2 shows some results for a particular Closest Point Method calculation on bands which vary by width. We find that the banded procedures give results identical to computing over the entire embedding space. Table 2.2 also illustrates that using smaller bandwidths than those in Table 2.1 results in a different solution.

Table 2.2 confirms that the stencil set approach produces identical results to computing on the entire embedding domain. Finally, Table 2.2 reports the number of points in each band, and note that in three dimensions the extension band contains 94% of the points used in the bandwidth approach. Likewise, the evolution band contains 85% of the points used in the bandwidth approach. In the Klein bottle computation in 4D (see Section 2.4.6), these savings are more significant as only 72% and 54% of the points are required in the respective bands. The stencil set approach thus offers computational savings which, in combination with its simplicity, leads us to use this approach in the calculations appearing throughout this section.

### 2.3.3 Implementation

For the results in this chapter, the explicit Closest Point Method is implemented in C. For computations embedded in  $\mathbb{R}^3$ , large 3D arrays which enclose the surface are used. The various banding approaches work on only a small percentage of these arrays. This approach is easy to implement but is perhaps inefficient in terms of storage. In Chapter 3, a memory efficient approach to banding is implemented.

Visualizations of the results are done with MATLAB (the function `contourslice` is particularly useful) and Geomview.

## 2.4 Numerical Results

Numerical studies are provided next to illustrate the behaviour and convergence of the method for a variety of practical test cases: passive transport, geometric flow under constant normal flow and redistancing via the standard reinitialization PDE. The geometric flexibility of the method is also illustrated by treating normal flow on the triangulated surface of a human hand and flow on a Klein bottle, a codimensional-two object in four dimensions. In all of the examples, the new WENO-based interpolation procedure is used to carry out the extension step and standard Hamilton–Jacobi WENO-based techniques are used to treat the embedding PDE. For efficiency, calculations are performed in a narrow band around the surface as described in Section 2.3.

### 2.4.1 Passive transport: flow under a specified velocity field

An important example of interface motion on surfaces is *passive transport* or flow under a specified velocity field. In this case, an interface—represented on the surface by the zero-contour of the level set function  $\phi$ —is advected via a velocity field which is independent of the interface geometry. On the surface, such a motion corresponds to the equation

$$\phi_t + \mathbf{V} \cdot \nabla_S \phi = 0,$$

for some velocity field  $\mathbf{V}$  specified on the surface. To evolve this surface PDE using the Closest Point Method, we instead treat the embedding PDE

$$\phi_t + \mathbf{V}(\text{cp}(\mathbf{x})) \cdot \nabla \phi = 0, \tag{2.6}$$

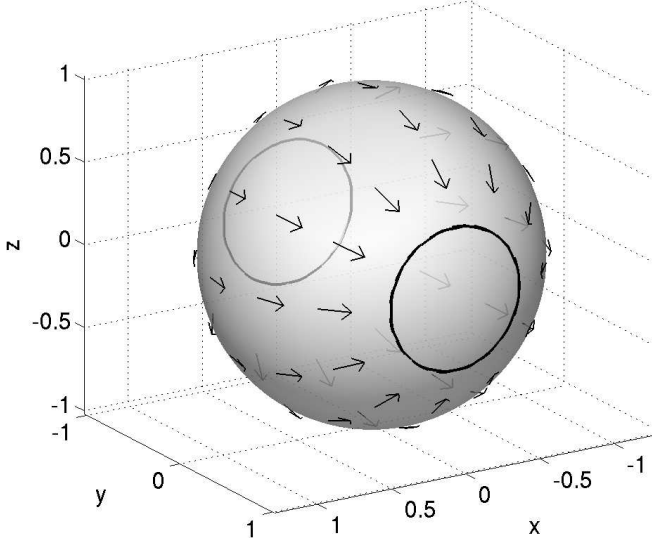


Figure 2.6: Passive transport of a circular interface on a sphere. The circle starts initially at  $y = -0.9$  on the far side of the sphere. It is advected over the surface of the sphere via the velocity field indicated with arrows to the final solution at  $t_f = 2.24$  shown on the front of the sphere. The exact solution—also a circle—is shown, but within the tolerances of the plot, it is almost indistinguishable from the numerical solution. The computation grid has  $\Delta x = 0.16$ .

$\Delta x$	error-in-position	order
0.16	$1.7263 \times 10^{-4}$	
0.08	$2.6721 \times 10^{-6}$	6.01
0.04	$7.3215 \times 10^{-8}$	5.19
0.02	$2.1474 \times 10^{-9}$	5.09
0.01	$6.3800 \times 10^{-11}$	5.07
0.005	$1.8378 \times 10^{-12}$	5.12

Table 2.3: Numerical convergence study for passive transport of a circular interface moving on a sphere. Error-in-position measures the error in the position of the interface along the surface where  $z = 0$ . Graphically, the situation is similar to Figure 2.6, but with the circle beginning initially at  $y = -0.25$  and running to  $t_f = 1$ .

on a uniform 3D grid. Equation (2.6) is simply the standard equation for passive transport in 3D since  $\mathbf{V}(\text{cp}(\mathbf{x}))$  is well-defined in  $\mathbb{R}^3$ . It is therefore natural to use standard methods [OF03] and approximate  $\nabla\phi$  using upwinding and WENO approximations in a dimension-by-dimension fashion as described in Sections 1.1.3 and 1.1.4. The result is then a method-of-lines procedure. Time stepping is done with the three-stage, third-order strong-stability-preserving (SSP) Runge–Kutta scheme [SO88] (SSP(3,3) from Section 1.2.1) with  $\Delta t = \frac{1}{2}\Delta x$  and with closest point extensions performed after each stage. It is emphasized that apart from the closest point extensions, this is simply a standard procedure used for evolving (2.6) in three dimensions.

To test the numerical convergence of the method, consider a circular interface on a unit sphere, evolving by passive transport. As shown in Figure 2.6, we take a velocity field that is of unit length and emanates from one pole to the other in a radially symmetric fashion

about the axis between the poles (like lines of longitude on a tilted globe). A comparison of the numerical result against the exact solution is provided in Table 2.3. These results show a clear fifth-order convergence.

For this smooth problem we see that the Closest Point Method recovers the expected fifth-order accuracy of the Hamilton–Jacobi WENO discretization of the embedding PDE [OF03]. This demonstrates that the closest point extension procedure based on WENO interpolation performs as anticipated and without degrading the fifth-order accurate treatment of the embedding PDE.

### 2.4.2 Normal flow

We next consider the case of *normal flow* where the motion of the interface is governed not by an external velocity field but by the shape of the interface itself. We begin with constant normal flow where the interface moves in the direction of its in-surface normal vector at a constant speed  $C$ , according to the surface PDE

$$\phi_t + C|\nabla_S \phi| = 0.$$

If  $C = 1$ , the problem is called *unit normal flow* and the underlying 3D embedding PDE is

$$\phi_t + |\nabla \phi| = 0, \tag{2.7}$$

which is a Hamilton–Jacobi equation with *Hamiltonian*  $H(\nabla \phi) = |\nabla \phi|$ .

The embedding PDE is discretized in space using Lax–Friedrichs for Hamilton–Jacobi equations [OF03, OS91]. Specifically, the *numerical Hamiltonian*

$$\begin{aligned} \hat{H} = & \left| \left\langle \frac{\phi_x^- + \phi_x^+}{2}, \frac{\phi_y^- + \phi_y^+}{2}, \frac{\phi_z^- + \phi_z^+}{2} \right\rangle \right| \\ & - \alpha^x \left( \frac{\phi_x^+ - \phi_x^-}{2} \right) - \alpha^y \left( \frac{\phi_y^+ - \phi_y^-}{2} \right) - \alpha^z \left( \frac{\phi_z^+ - \phi_z^-}{2} \right), \end{aligned} \tag{2.8}$$

is used where  $\phi_x^+$ ,  $\phi_y^-$ , etc. are calculated using Hamilton–Jacobi WENO and the latter three terms provide artificial dissipation. The dissipation coefficients  $\alpha^x$ ,  $\alpha^y$  and  $\alpha^z$  are calculated as the bounds for partial derivatives of the Hamiltonian  $H$  over some domain, the choice of which leads to variations of the Lax–Friedrichs scheme. The local Lax–Friedrichs (LLF) and stencil local Lax–Friedrichs (SLLF) variants [OF03] are implemented and used in this section. After computing the numerical Hamiltonian, we can proceed by the method of lines where time stepping is again done with the SSP(3,3) scheme with  $\Delta t = \frac{1}{2}\Delta x$  and closest point extensions after each stage.



$\Delta x$	error-in-position	order
0.16	$9.6093 \times 10^{-5}$	
0.08	$1.8654 \times 10^{-6}$	5.69
0.04	$3.4943 \times 10^{-8}$	5.74
0.02	$5.5902 \times 10^{-10}$	5.97
0.01	$1.0222 \times 10^{-11}$	5.77
0.05	$2.2932 \times 10^{-13}$	5.48

Table 2.4: Numerical convergence study for constant normal flow for a circle moving on a unit-radius sphere. Error-in-position measures the maximum error in the position of the zero-contour over the quadrant of the sphere where  $x \geq 0$  and  $z \geq 0$ . The circle begins at  $y = -0.25$  and the computation proceeds using LLF to  $t_f = 0.5$  with  $\Delta t = \frac{1}{2}\Delta x$ .

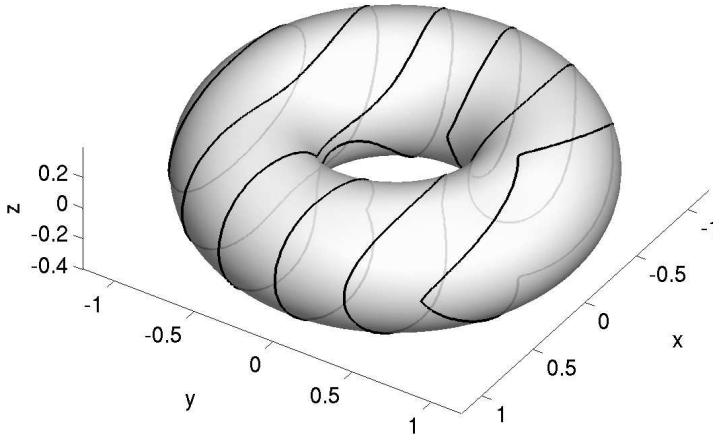


Figure 2.7: Unit normal flow on a torus with radii 0.8 and 0.4. The interface begins as an ellipse at  $y = -1$  and the computation proceeds using SLLF to  $t_f = 2$ . The interface is shown at every 0.4 units of time, travelling from left to right. The computational grid has  $\Delta x = 0.04$ .

To test the order of convergence of the method, we compute the motion of a circle on a sphere via unit normal flow. The exact solution is simply that the circle moved along the surface of the sphere, similar to the passive transport case in Figure 2.6. Table 2.4 shows that the Closest Point Method achieves at least fifth order on this problem, again validating the choice of WENO interpolation technique.

Of course, non-spherical surfaces may also be treated. Figure 2.7 shows the motion of an initial interface on a torus, as computed using the SLLF scheme for the embedding PDE. As anticipated, the interface moves from left to right parallel to the  $y$ -axis via unit normal flow, separating and re-combining as necessary.

### 2.4.3 Signed distance and reinitialization

In practical applications, level set functions may become either too steep or too flat during their evolution. *Reinitialization* is often used to take general level set functions closer to signed distance functions, or to generate accurate approximations of signed distance

functions. Given the widespread use of such techniques, it is of interest to see whether the corresponding surface reinitialization PDE

$$\phi_t + \text{sgn}(\phi_0) (|\nabla_S \phi| - 1) = 0, \quad (2.9)$$

[SSO94, RS00] can be accurately treated using the Closest Point Method. Here we assume that the initial interface is, as usual, specified as the zero-contour of an initial  $\phi_0$ . Starting from  $\phi_0$ , the surface level set equation (2.9) evolves so that in the steady state  $\phi(\mathbf{x})$  gives the signed distance (along the surface) from  $\mathbf{x}$  to the interface. In practice, this evolution may also move the zero-contour of  $\phi$ ; we want this motion to be small and to vanish as the discretization grid spacings  $\Delta x$  tend to zero.

Treating this problem using the Closest Point Method is straightforward; we discretize the corresponding three-dimensional redistancing embedding PDE

$$\phi_t + \text{sgn}(\phi_0) (|\nabla \phi| - 1) = 0, \quad (2.10a)$$

and as is standard practice [OF03, FAMO99, Mit04], we replace the signum function with a smoother version

$$\text{sgn}(\phi_0) \approx S(\phi_0) = \frac{\phi_0}{\sqrt{\phi_0^2 + \epsilon^2}}. \quad (2.10b)$$

Typically,  $\epsilon$  is set equal to  $\Delta x$  but in this work we use  $\epsilon = \sqrt{\Delta x}$ , as suggested in [CT08]. This latter choice of  $\epsilon$  gave considerably better convergence results than the former.

Following [FAMO99, Mit04], we implement a modified Godunov scheme for (2.10). Specifically, at each grid point  $\mathbf{x}_j$  we compute  $\phi_x^+$  and  $\phi_x^-$  using Hamilton–Jacobi WENO. We then select an approximation  $\Phi_x$  to  $\phi_x$  according to

$$\Phi_x = \begin{cases} \phi_x^- & \text{if } S(\phi_0)\phi_x^+ \geq 0 \text{ and } S(\phi_0)\phi_x^- \geq 0, \\ \phi_x^+ & \text{if } S(\phi_0)\phi_x^+ \leq 0 \text{ and } S(\phi_0)\phi_x^- \leq 0, \\ 0 & \text{if } S(\phi_0)\phi_x^+ > 0 \text{ and } S(\phi_0)\phi_x^- < 0, \\ \phi_x^- & \text{if } S(\phi_0)\phi_x^+ < 0 \text{ and } S(\phi_0)\phi_x^- > 0 \text{ and } s := S(\phi_0) \frac{|\phi_x^+| - |\phi_x^-|}{\phi_x^+ - \phi_x^-} \geq 0, \\ \phi_x^+ & \text{if } S(\phi_0)\phi_x^+ < 0 \text{ and } S(\phi_0)\phi_x^- > 0 \text{ and } s := S(\phi_0) \frac{|\phi_x^+| - |\phi_x^-|}{\phi_x^+ - \phi_x^-} < 0. \end{cases}$$

Having repeated this procedure in the  $y$  and  $z$  directions at  $\mathbf{x}_j$ , we can approximate

$$S(\phi_0) (|\nabla \phi| - 1) \approx S(\phi_0) \left( \sqrt{\Phi_x^2 + \Phi_y^2 + \Phi_z^2} - 1 \right).$$

Table 2.5: Numerical convergence study for signed distance / reinitialization at  $t = 5$  where  $\phi_0$  is a signed *half*-distance function (i.e.,  $\phi_0 = d/2$  where  $d$  is the signed distance function) to a circular interface at  $y = -0.25$  on the surface of a unit sphere. Error-in-position measures the maximum error in the position of the contour in the quadrant of the sphere where  $x \geq 0$  and  $z \geq 0$ .

$\Delta x$	zero-contour		0.15-contour	
	error-in-position	order	error-in-position	order
0.16	$4.42 \times 10^{-4}$		$6.06 \times 10^{-4}$	
0.08	$1.08 \times 10^{-5}$	5.36	$1.71 \times 10^{-5}$	5.15
0.04	$4.08 \times 10^{-7}$	4.72	$5.39 \times 10^{-7}$	4.99
0.02	$2.57 \times 10^{-8}$	3.99	$2.27 \times 10^{-8}$	4.57
0.01	$1.12 \times 10^{-9}$	4.52	$9.14 \times 10^{-10}$	4.64

The Closest Point Method then proceeds as a method-of-lines computation with closest point extensions following each stage of the SSP(3,3) scheme with  $\Delta t = \frac{1}{2}\Delta x$ .

Table 2.5 shows between fourth- and fifth-order convergence for the signed distance problem as measured by the error in the position of the 0 and 0.15 contours.

#### 2.4.4 WENO interpolation on marginally resolved surfaces

Even when the solution of the PDE is smooth, WENO interpolation can offer improved results over fixed-stencil Lagrange interpolation when the underlying surface is nonsmooth or marginally resolved by the spatial grid. The latter is demonstrated by considering the reinitialization equation with sinusoidal initial conditions on the lines  $x = a$  and  $x = b$ ,

$$\phi_0(a, y) = \sin(\pi y),$$

$$\phi_0(b, y) = \cos(\pi y).$$

Interpreting the two lines as representing different arcs of a curve embedded in  $\mathbb{R}^2$ , we may apply the Closest Point Method to compute approximations of the solution. The underlying PDE and initial conditions give a smooth flow. The problem becomes computationally interesting when the two lines are separated by only a few grid points, as the two lines are then marginally resolved by the grid.

Figure 2.8 shows the results when the lines are separated by only four grid nodes (that is,  $b - a = 4\Delta x$ ). Using Lagrange interpolation (based on degree five polynomials) for the

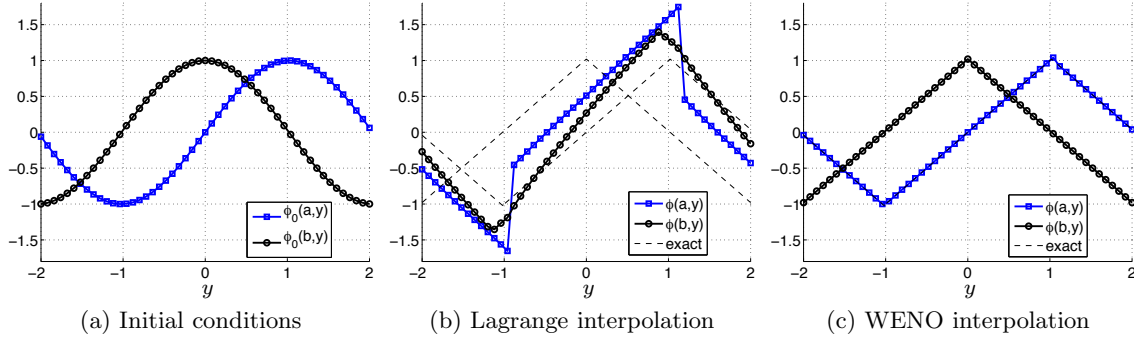


Figure 2.8: A comparison of Lagrange interpolation versus WENO interpolation on closely spaced surfaces with  $b - a = 4\Delta x$ . This is a signed distance computation with  $t_f = 4$  and periodic boundary conditions at  $y = -2$  and  $y = 2$ . The dots indicate the values of  $\phi$  on the grid nodes in the  $y$  direction.

closest point extension leads to an incorrect solution with discontinuities and substantial movement of the zero-contour on both lines. This error is due to the close proximity of the lines resulting in the stencil using data from both lines, an approach which is nonlocal and incorrect. WENO interpolation, however, chooses stencils based on smoothness and hence avoids using data from the more distant line. This approach leads to a non-oscillatory numerical approximation that is in excellent agreement (Figure 2.8(c)) with the exact result. It turns out via a straightforward examination of the two interpolation stencils and the evolution stencil that to avoid nonlocal interactions, there must be eight grid points between the lines in the Lagrange interpolation case but only four grid points in the WENO interpolation case. Finally, note in Figure 2.9 that WENO interpolation fails much more gracefully when the lines are moved even closer together, unlike Lagrange interpolation which results in wildly incorrect results with large discontinuities.

#### 2.4.5 Triangulated surfaces

The examples thus far have involved fairly simple surfaces. Complex surfaces may also be treated by the Closest Point Method so long as a closest point representation of the underlying surface is available or can be computed.

Extensive and freely available collections of complex shapes exist, and many of the available surfaces are in a triangulated form. Naturally, we might wish to be able to compute flows on such surfaces, a task that, for the Closest Point Method, requires the construction

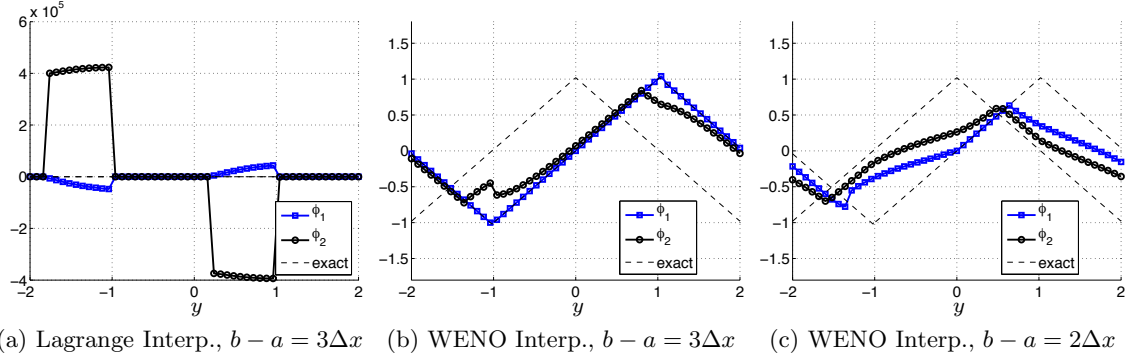


Figure 2.9: A comparison of Lagrange interpolation versus WENO interpolation on increasingly closely spaced surfaces. All three solutions are incorrect but (b) and (c) using WENO interpolation fail much more gracefully (note vertical scale in (a)).

of a closest point representation from a surface triangulation.

A straightforward method to convert triangulated surfaces into closest point representations is to loop over the list of triangles to directly determine the triangle closest to each grid node in the embedding space. Then the closest point on the surface is given by the closest point on the corresponding closest triangle. Naïvely implemented, this approach is computationally expensive (and inefficient) since each triangle must be examined for each node.

A much faster construction of the closest point function is obtained by taking into account that the method works on a narrow computational band [MR]. First, for each triangle, we determine all nodes that are within the bandwidth,  $BW_{\text{extend}}$ , of the triangle (nodes that are further away cannot be influenced by that part of the surface). For each node, this approach gives a list of triangles. This list is sufficiently small that it is normally quite practical to directly examine each member to determine the closest triangle (and hence the closest point on the surface). See [MR] for full details on this initialization procedure.

Notice that the triangulation is used only in the initial computation of the closest point representation and for plotting purposes; otherwise the Closest Point Method calculation proceeds exactly as described in Chapter 1 and in the previous examples. For example, Figure 2.10 gives a computation for unit normal flow on the surface of “Laurent’s Hand” [SAA07] (the hand consists of 351,048 vertices comprising 701,543 triangles). The flow itself was carried out using the same code as was used in Section 2.4.2; as anticipated, the only modifications appear in the initial computation of the closest point representation and in

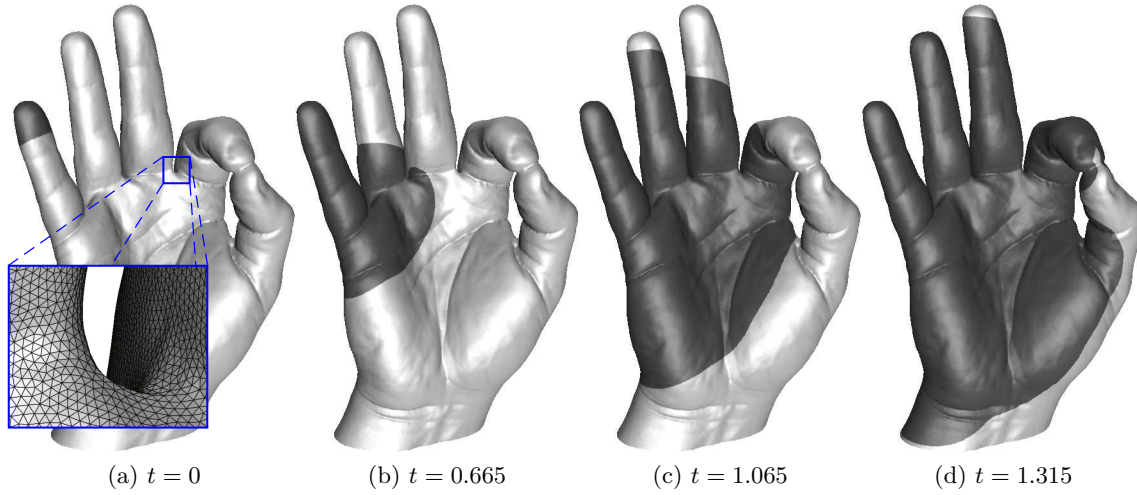


Figure 2.10: Unit normal flow on “Laurent’s Hand” shown at various times. The interface—visualized here as a transition from black to white—begins at the tip of the little finger. The computation uses LLF on the computational domain  $[-1, 1]^3$  with a  $401 \times 401 \times 401$  grid.

the visualization procedure.

### 2.4.6 Klein bottle

The Klein bottle is a famous surface embedded in 4D (thus a surface of codimension 2) which is closed but has no inside and outside. Although the Klein bottle appears self-intersecting when drawn projected into 3D, the complete surface in 4D is not. We consider a parameterization [Wik07] in terms of  $(u, v) \in [0, 2\pi)^2$

$$x = \begin{cases} \frac{3}{7} \cos u (1 + \sin u) + \frac{2}{7} r \left(1 - \frac{\cos u}{2}\right) \cos u \cos v, & \text{if } u \leq \pi, \\ \frac{3}{7} \cos u (1 + \sin u) - \frac{2}{7} r \left(1 - \frac{\cos u}{2}\right) \cos v, & \text{otherwise,} \end{cases} \quad (2.11a)$$

$$y = \begin{cases} \frac{8}{7} \sin u + \frac{2}{7} r \left(1 - \frac{\cos u}{2}\right) \sin u \cos v - \frac{1}{7}, & \text{if } u \leq \pi, \\ \frac{8}{7} \sin u - \frac{1}{7}, & \text{otherwise,} \end{cases} \quad (2.11b)$$

$$z = \frac{2}{7} r \left(1 - \frac{\cos u}{2}\right) \sin v, \quad (2.11c)$$

$$w = -\frac{8}{7} \cos u, \quad (2.11d)$$

where  $r$  controls the radius of the bottle (we use  $r = 1$ ). We define a function  $\mathbf{f}_{\text{Klein}} : \mathbb{R}^2 \rightarrow \mathbb{R}^4$  such that  $\mathbf{x} = \langle x, y, z, w \rangle = \mathbf{f}_{\text{Klein}}(u, v)$  as in (2.11).

This surface is unlikely to have a simple closest point function. However we do have the parameterization (2.11), and thus for a given grid point  $\mathbf{x}_0 = \langle x_0, y_0, z_0, w_0 \rangle$ , we can compute the closest point by minimizing

$$d^2(u, v) = \|\langle x_0, y_0, z_0, w_0 \rangle - \mathbf{f}_{\text{Klein}}(u, v)\|_2^2,$$

over  $(u, v)$  (using, for example, MATLAB's `fminsearch`) to find  $(u_{\min}, v_{\min})$ . The closest point to  $\mathbf{x}_0$  is then

$$\text{cp}(\mathbf{x}_0) = \mathbf{f}_{\text{Klein}}(u_{\min}, v_{\min}).$$

Once this straightforward—albeit time consuming—series of optimizations have been performed, the results are stored and can then be reused for any further Closest Point Method calculations on the same grid.

Figure 2.11 shows the results of the reinitialization equation calculation (Section 2.4.3) on the surface of the Klein bottle. This example illustrates that the Closest Point Method can treat both non-orientable surfaces and surfaces of codimension-two. Note that this computation requires no special changes to the Closest Point Method, illustrating that the method can handle very general surfaces without any particular modifications. We also note that although the computational grid was  $51 \times 51 \times 51 \times 51$ , only 340,057 points or about 5% of that grid is contained in the band used for computation. It is anticipated that problems of high codimension would benefit in terms of memory requirements from a more flexible storage scheme than the simple 4D array used here. An appropriate scheme is considered in Chapter 3.

### 2.4.7 Forest fires

This example is a departure from the previous high-order examples using WENO interpolation. Instead it demonstrates another feature of the Closest Point Method: namely, the ease at which existing 3D codes can be straightforwardly modified to compute the corresponding flows on surfaces. The Toolbox of Level Set Methods [Mit04] is a add-on for MATLAB which implements many level set techniques appearing in [OF03].

Using the toolbox to implement level set methods on surfaces via the Closest Point Method is straightforward. Aside from acquiring the closest point values and visualization code, a single line of matlab code is all that is required:

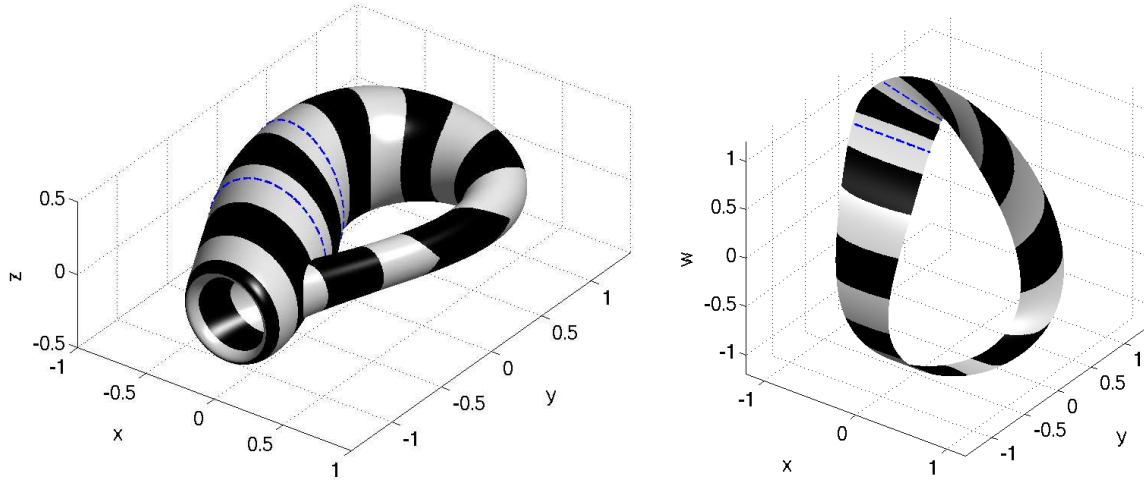


Figure 2.11: Reinitialization / Signed Distance on a Klein bottle from initial conditions  $\phi_0 = 1.1 - w$ . Left:  $(x, y, z)$ -projection, right:  $(x, y, w)$ -projection. Each transition from dark-to-light or light-to-dark represents contours of equal distance from the closer of the two initial interfaces indicated with dashed lines. Note that some shaded bands appear narrower than others in the 3D projections of the Klein bottle. The grid is  $51 \times 51 \times 51 \times 51$  on the domain  $[-2, 2]^4$ .

```
phi = interpn(gridx, gridy, gridz, phi, cpx, cpy, cpz);
```

This line calls the matlab function `interp()` to perform a closest point extension of `phi`. The variable `phi` is an array of values of the level set function  $\phi$  at a set of grid points  $(x, y, z) = (\text{gridx}, \text{gridy}, \text{gridz})$  and the arrays `cpx`, `cpy`, `cpz` store the  $x$ ,  $y$  and  $z$  coordinates of the associated closest points.

A simple level set model for a burning flame front was used based on [ABP<sup>+</sup>07]. The flame front propagates outwards in the (in-surface) normal direction and also burns faster in the up-slope direction.

$$\phi_t = -C(\mathbf{x})|\nabla_S \phi| + \nabla \Psi \cdot \nabla_S \phi, \quad (2.12)$$

where  $\nabla \Psi$  is the (non-intrinsic) gradient of the surface and  $C(\mathbf{x})$  is the normal rate of spread possibly depending on local properties such as the fuel type [ABP<sup>+</sup>07]. Figure 2.12 shows an example of this model computing the evolution of a forest fire on the surface of a mountain.



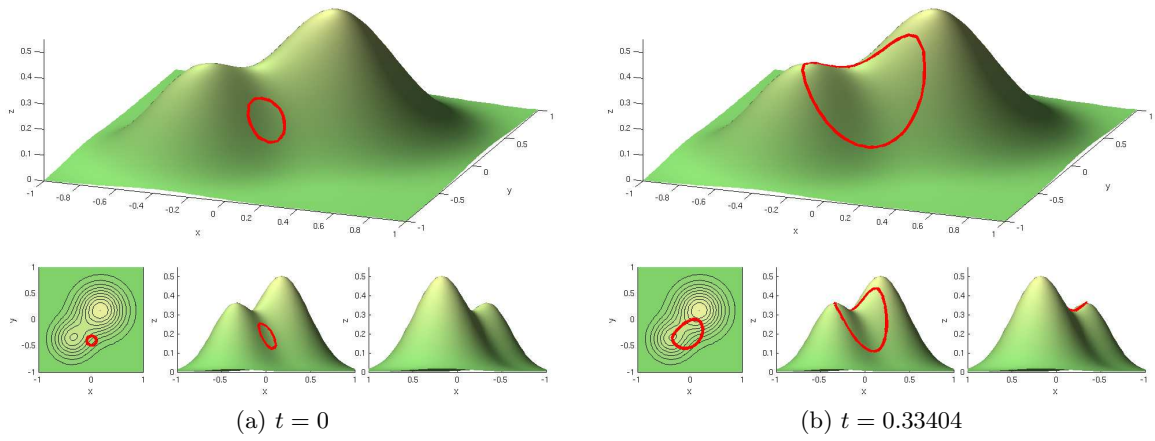


Figure 2.12: Forest fire computation on the surface of a mountain.

## Chapter 3

# The Implicit Closest Point Method

The explicit Closest Point Method in Chapter 1 works very well for Hamilton–Jacobi problems such as the level set equation in Chapter 2. The method efficiently achieves high-order accuracy using standard spatial discretizations and explicit Runge–Kutta time-stepping methods with time-step sizes  $\Delta t = \mathcal{O}(\Delta x)$ , i.e., on the order of the spatial grid.

For problems with second- and higher-order derivatives such as the heat equation, explicit time stepping may be inefficient because the ODE system resulting from the spatial discretization is often stiff: small time steps are required to maintain linear stability even when larger time steps would be adequate from the point of view of consistency. Nonetheless, the explicit Closest Point Method has been used successfully in [RM08] to compute the solution of second-order problems on surfaces (specifically, in-surface curvature motion and diffusion). Developing stable methods becomes even more critical when higher-order spatial derivatives arise, such as biharmonic terms, where the resulting stiffness makes explicit time stepping prohibitively expensive.

In this chapter, a new implicit Closest Point Method is presented. It uses implicit linear multistep or Runge–Kutta time-stepping schemes and allows large time steps. Numerical convergence studies demonstrate the high-order properties of this method on both the heat equation and biharmonic problems. Several applications are presented, including image processing (blurring an image on a triangulated surface) and modelling heat flow on a complicated surface with components of various codimension. Finally, in-surface pattern formation results are presented using implicit-explicit (IMEX) time-stepping schemes.

### 3.1 Introduction

Let  $\mathcal{S}$  be a surface embedded in  $\mathbb{R}^d$  and consider, as an example, the in-surface heat equation

$$u_t = \Delta_{\mathcal{S}} u, \quad (3.1a)$$

$$u(0, \mathbf{x}) = u_0(\mathbf{x}). \quad (3.1b)$$

Based on the principles in Chapter 1 and [RM08], we may replace the Laplace–Beltrami operator  $\Delta_{\mathcal{S}}$  by the standard Laplacian  $\Delta$  in the embedding space  $\mathbb{R}^d$ , provided we evaluate at the closest point  $\text{cp}(\mathbf{x})$ , thus

$$u_t(t, \mathbf{x}) = \Delta u(\text{cp}(\mathbf{x})), \quad (3.2a)$$

$$u(0, \mathbf{x}) = u_0(\text{cp}(\mathbf{x})), \quad (3.2b)$$

and the solutions of (3.1) and (3.2) will agree on the surface.

At this point in the explicit Closest Point Method, we would drop the closest point operator  $\text{cp}$  in (3.2a) and note that the resulting embedding PDE agrees *initially* with (3.2); an appropriate spatial discretization combined with explicit time stepping could then be used, followed by closest point extensions to regain agreement with (3.2) at the start of the next step (or stage) as described in Chapter 1.

In contrast, the implicit Closest Point Method is based on discretizing (3.2) directly. A brief outline of this idea is presented first in 2D before deriving the more general form in some detail in Section 3.2. Assume for the moment that the surface  $\mathcal{S}$  is a curve embedded in 2D and consider a second-order centered finite difference scheme applied to the Laplacian  $\Delta$  in (3.2), resulting in

$$\begin{aligned} \frac{\partial}{\partial t} u(x, y) = \frac{1}{\Delta x^2} \Big( & -4u(\text{cp}(x, y)) + u(\text{cp}(x + \Delta x, y)) + u(\text{cp}(x - \Delta x, y)) \\ & + u(\text{cp}(x, y + \Delta x)) + u(\text{cp}(x, y - \Delta x)) \Big), \end{aligned} \quad (3.3a)$$

$$u(0, x, y) = u_0(\text{cp}(x, y)), \quad (3.3b)$$

where  $x$  and  $y$  are still continuous variables (so this is not yet a spatial semi-discretization). The spatial semi-discretization can be completed by choosing a grid  $x_i$  and  $y_j$  and by using interpolation of the values at grid nodes surrounding each  $\text{cp}(x_i, y_j)$  to approximate the value of  $u(\text{cp}(x_i, y_j))$ . As the value of  $\text{cp}(x_i, y_j)$  is known beforehand, the interpolation

weights can be precomputed with barycentric Lagrange interpolation (Section 1.5.2, [BT04]). These interpolation weights and the evolution weights  $\{\frac{-4}{\Delta x^2}, \frac{1}{\Delta x^2}, \frac{1}{\Delta x^2}, \frac{1}{\Delta x^2}, \frac{1}{\Delta x^2}\}$  can then be combined into a matrix (as explained later in detail in Section 3.2) which approximates  $\Delta u(\text{cp}(x, y))$ . Thus, this method-of-lines approach yields a linear system of coupled ODEs in time for  $u_{ij}(t) \approx u(t, x_i, y_j)$ . The ODE system can then be discretized in time with standard implicit Runge–Kutta or linear multistep methods.

In practice, the system resulting from (3.3) exhibits linear instabilities and instead the following stabilized form is used:

$$\frac{\partial}{\partial t} u(x, y) = \frac{1}{\Delta x^2} \left( -4u(x, y) + u(\text{cp}(x + \Delta x, y)) + u(\text{cp}(x - \Delta x, y)) \right. \\ \left. + u(\text{cp}(x, y + \Delta x)) + u(\text{cp}(x, y - \Delta x)) \right), \quad (3.4a)$$

$$u(0, x, y) = u_0(\text{cp}(x, y)), \quad (3.4b)$$

where the only change is that the diagonal entries no longer involve the closest point operator  $\text{cp}$ . Note that this equation (3.4) and the previous (3.3) will agree at the surface because for points  $(x, y)$  on the surface we have  $\text{cp}(x, y) = (x, y)$ . This ensures that the solution of (3.4) is consistent with (3.1). This stabilizing procedure is discussed further in Section 3.2.3.

## 3.2 Matrix Formulation of the Closest Point Method

A matrix formulation of the Closest Point Method is presented. While the above introduction was specific to curves embedded in  $\mathbb{R}^2$ , the following derivation is presented in a more abstract fashion. Suppose the surface  $\mathcal{S}$  is embedded in an embedding space  $\mathbb{R}^d$ . Assume we have an interpolation scheme where the interpolated value is a linear combination of the values at neighbouring grid points in a hypercube in  $\mathbb{R}^d$ ; this hypercube of grid points forms the *interpolation stencil* of the scheme. We will use barycentric Lagrange interpolation (Section 1.5.2, [BT04]) of degree  $p$  so the hypercube has  $p + 1$  points on each side.

Now consider two discrete sets of points in the embedding space. The first is  $L_{\text{evolve}} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$  and it contains every grid point which occurs in the interpolation stencil for some point on the surface  $\mathcal{S}$ .<sup>1</sup> These grid points form the computational band on which the solution is propagated. We approximate the solution  $u$  at all points in  $L_{\text{evolve}}$

---

<sup>1</sup>While this condition is sufficient, a weaker condition can be used in practice. Namely that  $L_{\text{evolve}}$  must contain the interpolation stencil for each  $\text{cp}(\mathbf{x}_i)$  for  $\mathbf{x}_i \in L_{\text{evolve}} \cup L_{\text{ghost}}$ .

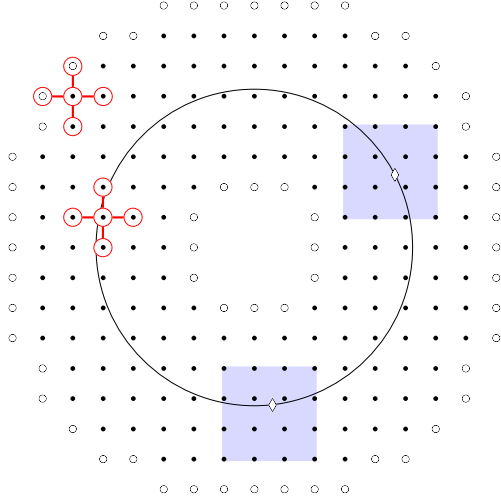


Figure 3.1: Example of the sets of grid points  $L_{\text{evolve}}$  (indicated by  $\bullet$ ) and  $L_{\text{ghost}}$  (indicated by  $\circ$ ) where the curve  $\mathcal{S}$  is a circle. The interpolation stencil is a  $4 \times 4$  grid (for example using degree  $p = 3$  barycentric Lagrange interpolation) and the shaded regions illustrate the use of this stencil at the two points on the circle indicated by  $\diamond$ . Five-point evolution stencils are shown for two example points in  $L_{\text{evolve}}$ , in one case illustrating the use of ghost points.

with the vector  $\mathbf{u} \in \mathbb{R}^m$  where  $u_i \approx u(\mathbf{x}_i)$  for each grid point  $\mathbf{x}_i \in L_{\text{evolve}}$ . The second set of points  $L_{\text{ghost}} = \{\mathbf{x}_{m+1}, \mathbf{x}_{m+2}, \dots, \mathbf{x}_{m+m_g}\}$  is disjoint from  $L_{\text{evolve}}$  and contains  $m_g$  additional “ghost points” along the edges of the computational band. These points will be used in the derivation below but their values are not propagated in time. Figure 3.1 shows an example of the two sets  $L_{\text{evolve}}$  and  $L_{\text{ghost}}$  where the curve  $\mathcal{S}$  consists of a circle embedded in  $\mathbb{R}^2$ . The construction of these sets is discussed further in Section 3.5.

Finally, define two other vectors over the sets  $L_{\text{evolve}}$  and  $L_{\text{ghost}}$  for use in the derivation. Let  $\mathbf{U} \in \mathbb{R}^m$  denote the vector with components  $U_i = u(\text{cp}(\mathbf{x}_i))$  for  $\mathbf{x}_i \in L_{\text{evolve}}$  and let  $\mathbf{U}_g \in \mathbb{R}^{m_g}$  be the vector with  $U_{g_i} = u(\text{cp}(\mathbf{x}_{m+i}))$  for  $\mathbf{x}_{m+i} \in L_{\text{ghost}}$ .

### 3.2.1 The discrete extension operator

The closest point extension is an operator which assigns a value of  $u(\text{cp}(\mathbf{x}))$  to  $u(\mathbf{x})$ . However, in a discrete setting,  $\text{cp}(\mathbf{x})$  is generally not a grid point. Hence instead we use a discrete closest point extension which interpolates the values of the solution  $u$  at the grid points (i.e., the vector  $\mathbf{u}$ ) in the interpolation stencil surrounding  $\text{cp}(\mathbf{x})$ . Because each interpolation is a linear combination of values in  $\mathbf{u}$ , this operation can be expressed as a matrix multiplication using the *discrete extension operator*  $\mathbf{E}$  defined next.

**Definition 3 [discrete extension operator  $\mathbf{E}$ ]** *Given the vectors  $\mathbf{u}$ ,  $\mathbf{U}$ ,  $\mathbf{U}_g$  and an interpolation scheme as described above, the discrete extension operator is a  $(m + m_g) \times m$*

matrix  $\mathbf{E}$  such that

$$\begin{pmatrix} \mathbf{U} \\ \mathbf{U}_g \end{pmatrix} \approx \mathbf{E} \mathbf{u}. \quad (3.5)$$

The nonzero entries in the  $i^{\text{th}}$  row of  $\mathbf{E}$  consist of the weights in the interpolation scheme for  $u(\text{cp}(\mathbf{x}_i))$ . That is, the components of  $\mathbf{E} = [\gamma_{ij}]$  are

$$\gamma_{ij} = \begin{cases} w_j & \text{if } \mathbf{x}_j \text{ is in the interpolation stencil for } \text{cp}(\mathbf{x}_i), \\ 0 & \text{otherwise,} \end{cases}$$

with  $w_j$  denoting the weight associated with the grid point  $\mathbf{x}_j$  in the interpolation scheme for point  $\text{cp}(\mathbf{x}_i)$ .

It is worth reiterating that we have assumed that all points appearing in an interpolation stencil will also be in  $L_{\text{evolve}}$ : one way this can be guaranteed is to construct  $L_{\text{evolve}}$  to contain exactly the points appearing in any interpolation stencil as will be done in Section 3.5.

### 3.2.2 The discrete differential operator

Suppose we are interested in a spatial discretization of the Laplace–Beltrami operator  $\Delta_{\mathcal{S}} u(\mathbf{x})$ . The procedure works for more general operators, but it is instructive to use a particular example. Recall that for points  $\mathbf{x}$  on the surface,  $\Delta_{\mathcal{S}} u(\mathbf{x})$  is consistent with  $\Delta u(\text{cp}(\mathbf{x}))$ . We thus consider approximating  $\Delta u(\text{cp}(\mathbf{x}))$  for all points of  $L_{\text{evolve}}$  as

$$\Delta u(\text{cp}(L_{\text{evolve}})) \approx \mathbf{\Delta}_h \begin{pmatrix} \mathbf{U} \\ \mathbf{U}_g \end{pmatrix} \quad (3.6)$$

where  $\mathbf{\Delta}_h$  is an  $m \times (m + m_g)$  matrix which approximates the Laplacian operator in  $\mathbb{R}^d$  using a linear finite difference scheme. The finite difference scheme is applied at each point in  $L_{\text{evolve}}$  by taking a combination of neighbouring points appearing in an *evolution stencil*. Some of these neighbouring grid points will be in  $L_{\text{evolve}}$  with corresponding values of  $u(\text{cp}(\mathbf{x}))$  in the vector  $\mathbf{U}$ ; the remaining grid points will be ghost points in  $L_{\text{ghost}}$ , and hence  $\mathbf{U}_g$  also appears in (3.6).

For example, in 2D, the classic second-order centered difference approximation to the Laplacian  $\Delta$  would see the nonzero entries of the  $i^{\text{th}}$  row of  $\mathbf{\Delta}_h$  consisting of the values

$\left\{\frac{-4}{\Delta x^2}, \frac{1}{\Delta x^2}, \frac{1}{\Delta x^2}, \frac{1}{\Delta x^2}, \frac{1}{\Delta x^2}\right\}$  (see Figure 3.2(a)). That is, the Laplacian of  $u$  at the point  $\text{cp}(\mathbf{x}_i)$  is approximated by

$$\Delta u(\text{cp}(\mathbf{x}_i)) \approx \frac{1}{\Delta x^2} \left( -4u(\text{cp}(\mathbf{x}_i)) + u(\text{cp}(\mathbf{x}_{i_N})) + u(\text{cp}(\mathbf{x}_{i_S})) + u(\text{cp}(\mathbf{x}_{i_E})) + u(\text{cp}(\mathbf{x}_{i_W})) \right),$$

(cf. (3.3)) where  $i_N, i_S, i_E$  and  $i_W$  are the indices of the four neighbouring points (some of which may occur in  $L_{\text{ghost}}$  as illustrated in Figure 3.1).

Both  $\mathbf{U}$  and  $\mathbf{U}_g$  can be approximated from the vector  $\mathbf{u}$  by multiplying by the  $(m + m_g) \times m$  discrete extension operator  $\mathbf{E}$  as

$$\begin{pmatrix} \mathbf{U} \\ \mathbf{U}_g \end{pmatrix} \approx \mathbf{E} \mathbf{u} = \begin{array}{c} \boxed{\phantom{\mathbf{U}}} \boxed{\phantom{\mathbf{U}_g}} \\ \mathbf{E} \end{array} \mathbf{u}. \quad (3.7)$$

Combining with (3.6) we have

$$\begin{aligned} \Delta u(\text{cp}(\mathbf{x})) &\approx \Delta_h (\mathbf{E} \mathbf{u}) = \Delta_h \mathbf{E} \mathbf{u} = \begin{array}{c} \boxed{\phantom{\Delta_h}} \boxed{\phantom{\mathbf{E}}} \boxed{\phantom{\mathbf{u}}} \\ \Delta_h \quad \mathbf{E} \end{array} \mathbf{u} \\ &= \begin{array}{c} \boxed{\phantom{\Delta_h}} \boxed{\phantom{\mathbf{E}}} \\ \tilde{\mathbf{M}} \end{array} \mathbf{u}, \end{aligned} \quad (3.8)$$

where  $\tilde{\mathbf{M}} = \Delta_h \mathbf{E}$  is an  $m \times m$  matrix. Thus a spatial discretization of the in-surface heat equation  $u_t = \Delta_S u$  is

$$\frac{\partial}{\partial t} \mathbf{u} = \tilde{\mathbf{M}} \mathbf{u}. \quad (3.9)$$

Note that  $\mathbf{U}_g$  and the ghost points were used only in the derivation; the system (3.9) is defined on the set of grid points  $L_{\text{evolve}}$ .

This procedure, as described above, used the 1D second-order finite difference approximation

$$u_{xx} = \frac{1}{\Delta x^2} (u_{j-1} - 2u_j + u_{j+1}) + \mathcal{O}(\Delta x^2),$$

in a dimension-by-dimension fashion to approximate the Laplacian in 2D and 3D as shown in Figure 3.2. However the procedure is not specific to second-order centered finite differences

Figure 3.2: Second-order stencils for the Laplacian in various dimensions.

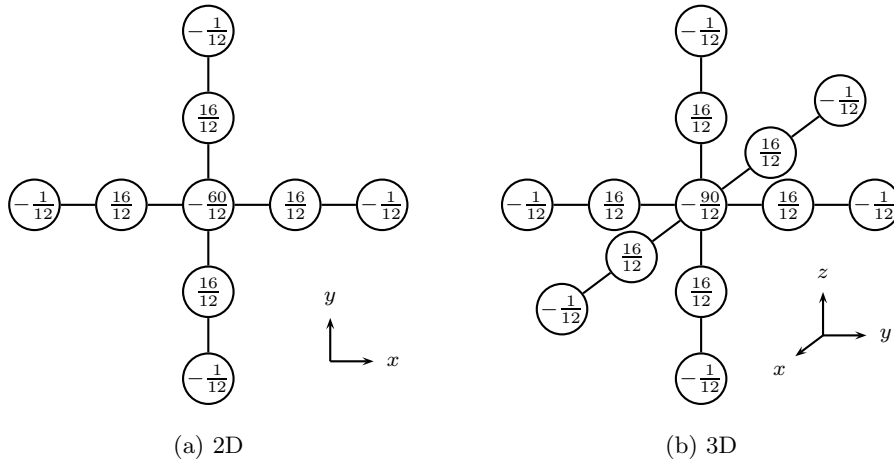
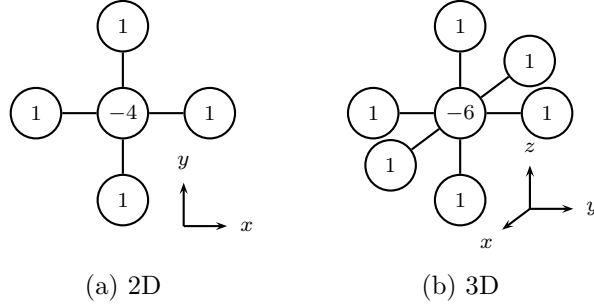


Figure 3.3: Fourth-order stencils for the Laplacian in various dimensions.

or to the Laplace–Beltrami operator. For example, we can also use the 1D fourth-order finite difference approximation

$$u_{xx} = \frac{1}{12\Delta x^2} (-u_{j-2} + 16u_{j-1} - 30u_j + 16u_{j+1} - u_{j+2}) + \mathcal{O}(\Delta x^4),$$

in a dimension-by-dimension fashion to approximate the Laplacian in 2D and 3D as shown in Figure 3.3. Other operators such as the intrinsic gradient considered in Chapter 2 or the intrinsic biharmonic operator in Section 3.6.3 can also be considered.

Given the semidiscrete problem (3.9), we next consider the temporal discretization using the implicit linear multistep methods of Section 1.2.2. Implicit Runge–Kutta methods would also be straightforward, particularly diagonally implicit Runge–Kutta schemes [KMG08, HW96]. For each time step, the multistep schemes perform a single linear system solve of the form  $\mathbf{A}\mathbf{u} = \mathbf{b}$  with  $\mathbf{A} = \mathbf{I} - \gamma\Delta t\tilde{\mathbf{M}}$  for some value of  $\gamma$  that depends on the particular method.



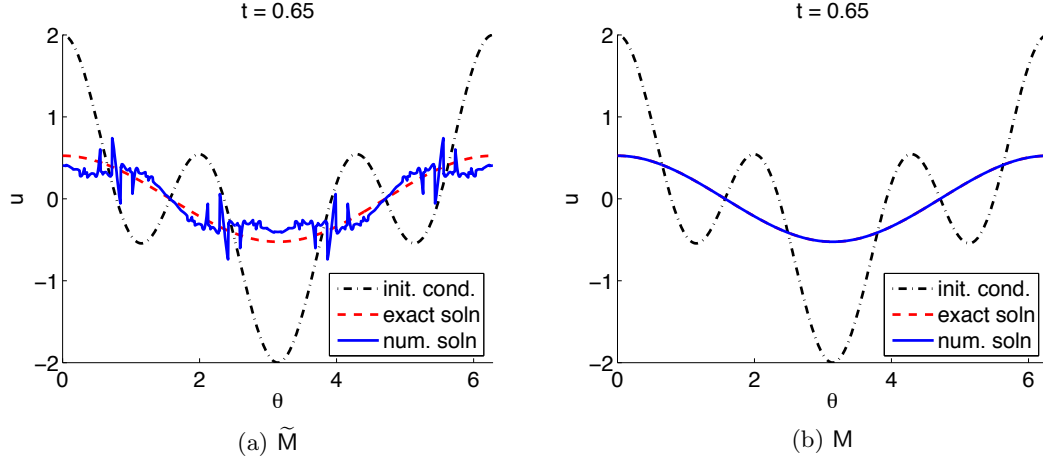


Figure 3.4: Stable and unstable solutions of the in-surface heat equation on a unit circle embedded in 2D, with  $\Delta x = 0.1$ , degree  $p = 4$  interpolation, and Backward Euler time stepping. Note oscillations indicating instability in (a) whereas the numerical solution essentially overlaps the exact solution in (b).

We denote the time-stepping matrices corresponding to the four linear multistep methods in Section 1.2.2 as  $A_{BE}$ ,  $A_{CN}$ ,  $A_{BDF2}$  and  $A_{BDF3}$ . The structure and properties of these matrices are important for understanding the behavior of direct and iterative methods for solving linear systems. Various properties and solution techniques are discussed in Section 3.3.

### 3.2.3 Stabilizing the Closest Point Method matrix

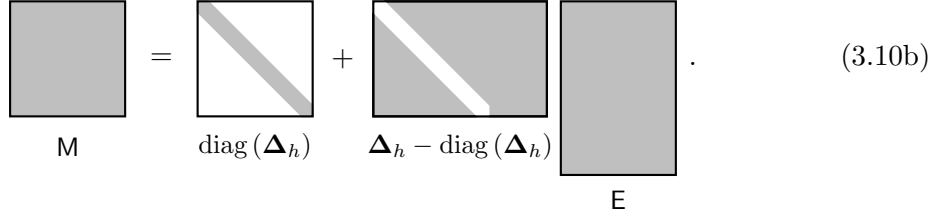
Recall that if the matrix in a linear system of ODEs such as the semidiscrete problem (3.9) has eigenvalues with positive real components, then the corresponding eigenmodes grow exponentially in time, causing a growing component in the solution. If the original PDE is diffusive and does not have growth in the exact solution, then the exponentially growing components in the solution of the semidiscrete problem correspond to instability. Indeed, Figure 3.4(a) shows that using the implicit Closest Point Method with  $\tilde{M}$  (as derived in (3.9)) to solve the in-surface heat equation (3.1) on a unit circle exhibits instabilities. By examining the spectra of  $\tilde{M}$  in Figure 3.5(a), we note the matrix has some eigenvalues with positive real components corresponding to growth in some eigenmodes.

We consider a modification of the implicit Closest Point Method procedure which has stable results. When approximating  $\Delta_S$  at the point  $\mathbf{x}_i$  with a finite difference stencil,

we map only the neighbouring points of the stencil  $\mathbf{x}_j$  back to their closest points  $\text{cp}(\mathbf{x}_j)$  and use  $u_i$  itself instead of  $u(\text{cp}(\mathbf{x}_i))$  (as in (3.4)). This special treatment of the diagonal elements yields a new  $m \times m$  matrix

$$\mathbf{M} = \text{stab}(\Delta_h, \mathbf{E}) := \text{diag}(\Delta_h) + (\Delta_h - \text{diag}(\Delta_h))\mathbf{E}, \quad (3.10a)$$

or diagrammatically



$$\mathbf{M} = \text{diag}(\Delta_h) + (\Delta_h - \text{diag}(\Delta_h))\mathbf{E}. \quad (3.10b)$$

Note that as explained in Section 3.1, this splitting does not impact the consistency of  $u$  at the surface. Additionally, the magnitude of the diagonal of the operator with respect to the off-diagonal elements has increased, thus increasing its stability (see also Section 3.3.3 which investigates the diagonal dominance of  $\tilde{\mathbf{M}}$  and  $\mathbf{M}$ ). Figure 3.4(b) shows a stable solution to the in-surface heat equation using  $\mathbf{M}$  and Figure 3.5(b) shows that all eigenvalues of  $\mathbf{M}$  have negative real parts.

### 3.2.4 Re-examining the explicit Closest Point Method

Finally for this section, note that the explicit Closest Point Method can be formulated using the matrices  $\Delta_h$  and  $\mathbf{E}$ . Recall from Chapter 1 that the explicit Closest Point Method for solving  $u_t = \Delta_S u$  consists of alternating between two steps:

1. Perform a forward Euler time step to advance from  $t_n$  to  $t_{n+1}$ :

$$\tilde{\mathbf{u}}^{n+1} = \mathbf{u}^n + \Delta t \Delta_h \mathbf{u}^n.$$

for each grid point in  $L_{\text{evolve}}$ .

2. Perform a closest point extension for each grid point  $\mathbf{x} \in \{L_{\text{evolve}} \cup L_{\text{ghost}}\}$ :

$$\mathbf{u}^{n+1} = \mathbf{E} \tilde{\mathbf{u}}^{n+1}.$$

The matrices  $\Delta_h$  and  $\mathbf{E}$  can be similarly used to formulate the explicit Closest Point Method for nonlinear problems. The the stabilization procedure of Section 3.2.3 could also be applied to the explicit Closest Point Method. Thus the matrix formulation is general enough to describe both the implicit and explicit Closest Point Methods.

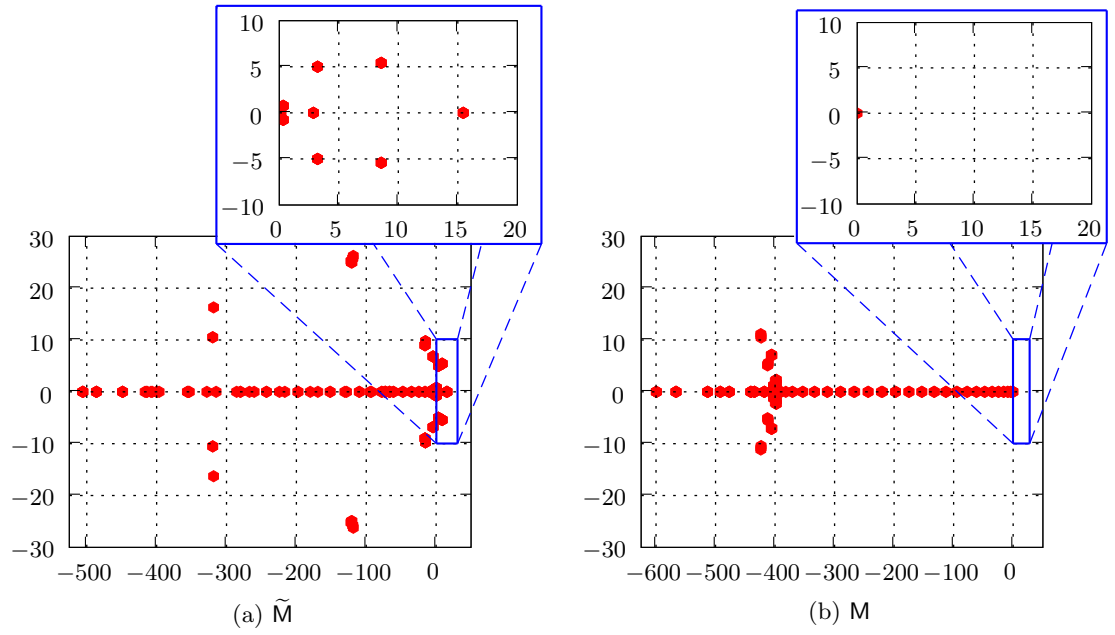


Figure 3.5: Spectra of the  $\tilde{M}$  (left) and  $M$  (right) matrices. Geometry: unit circle in 2D,  $E$  with biquartic interpolation ( $p = 4$ ), spatial grid has  $\Delta x = 0.1$ . Observe that  $\tilde{M}$  has eigenvalues in the right half plane.

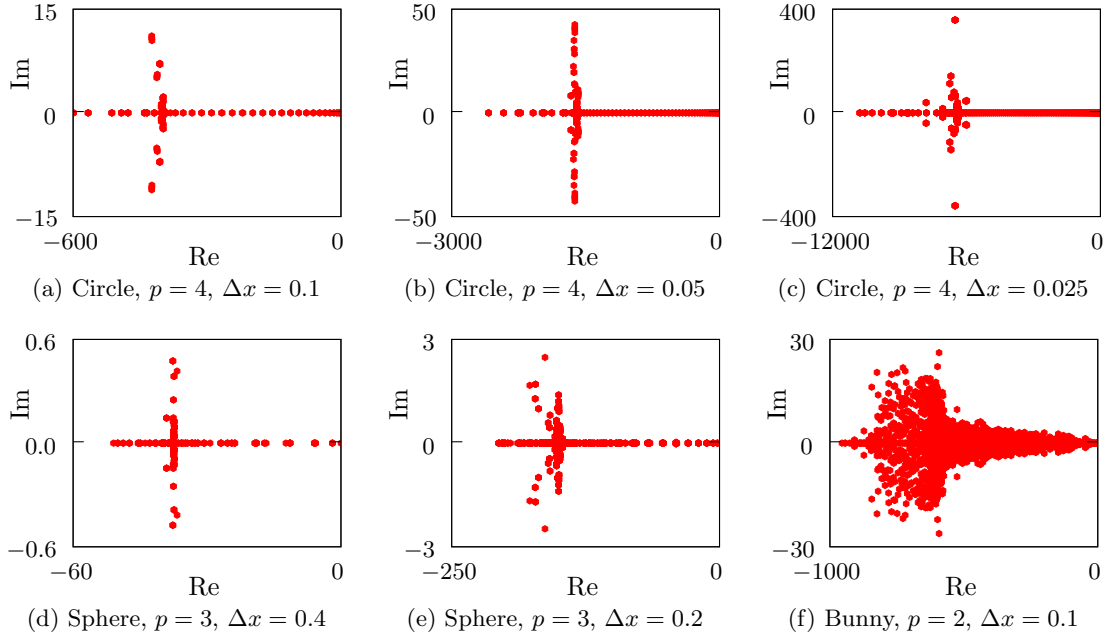


Figure 3.6: Examples of the spectrum of  $\mathbf{M}$  for various surfaces including a unit circle in 2D, a unit sphere in 3D and the Stanford Bunny (see Section 3.6.4) in 3D. Axes scales are linear.

### 3.3 Matrix Properties

In this section, some of the properties of the matrix  $\mathbf{M}$  are investigated, particularly those which relate to direct or iterative solution techniques for the system  $[\mathbf{I} - \Delta t \mathbf{M}] \mathbf{u} = \mathbf{b}$  which is used in the time-stepping schemes.

#### 3.3.1 Spectra, spectral radius and condition number

Figure 3.6 shows the spectrum (the distribution of the eigenvalues in the complex plane) of the matrix  $\mathbf{M}$  for several surfaces and various values of the grid spacing  $\Delta x$  and the degree of interpolation  $p$ . As previously noted in Section 3.2.3, the eigenvalues of  $\mathbf{M}$  have negative real parts, leading to stability. Note that the majority of eigenvalues appear to be real or almost real with relatively small imaginary parts. The largest imaginary components of the spectra are smaller than the smallest (most negative) real components by at least an order of magnitude. This decay of eigenmodes is consistent with the diffusive behavior we expect from a heat equation.

$\Delta x$	$\rho(\mathbf{M})$	$\rho(\mathbf{A}_{\text{BE}})$	$\text{cond}(\mathbf{A}_{\text{BE}})$
0.4	35.5	4.5	94.1
0.2	144	8.2	281
0.1	600	16.0	664
0.05	2587	33.3	1520
0.025	10777	68.4	3114
0.0125	41242	130	6289
0.00625	169999	267	12703
0.003125	694760	544	25984

Table 3.1: Condition number and spectral radius for time stepping matrix  $\mathbf{A}_{\text{BE}} = \mathbf{I} - \Delta t \mathbf{M}$  with  $\Delta t = \frac{1}{4}\Delta x$  for various  $\Delta x$ . The curve  $\mathcal{S}$  is a unit circle in 2D and the interpolation is degree  $p = 4$ .

Table 3.1 lists the condition number and spectral radius  $\rho$  for the time-stepping matrix  $\mathbf{A}_{\text{BE}} = \mathbf{I} - \Delta t \mathbf{M}$  for various  $\Delta x$ . Note in particular, that with  $\Delta t = \mathcal{O}(\Delta x)$ , the condition number of the time-stepping matrix  $\mathbf{A}_{\text{BE}}$  is relatively small and only doubles when  $\Delta x$  is halved. Similar results are obtained for the time-stepping matrices  $\mathbf{A}_{\text{CN}}$ ,  $\mathbf{A}_{\text{BDF2}}$  and  $\mathbf{A}_{\text{BDF3}}$ . Thus the time stepping matrices are well-conditioned.

Let  $\mathbf{A}_{\text{BE}}$  be factored into diagonal, lower and upper components as  $\mathbf{A}_{\text{BE}} = \mathbf{D} - \mathbf{L} - \mathbf{U}$ ; then  $\mathbf{T}_{\text{J}} = \mathbf{D}^{-1}(\mathbf{U} + \mathbf{L})$  and  $\mathbf{T}_{\text{GS}} = (\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}$  are the iteration matrices associated with the Jacobi and Gauss–Seidel iterative schemes [BF01]. Recall that the spectral radius of these iteration matrices measures (at least asymptotically) how fast the corresponding method converges [BF01]. Figure 3.7 shows the effect of  $p$  and  $\Delta x$  on the spectral radius of the Jacobi and Gauss–Seidel iteration matrices  $\mathbf{T}_{\text{J}}$  and  $\mathbf{T}_{\text{GS}}$ . These results are important for iterative methods, because the spectral radii are strictly less than one, and thus these iterative methods can be expected to converge.

### 3.3.2 Sparsity

Figure 3.8 shows the sparsity structure of the  $\Delta_h$ ,  $\mathbf{E}$  and  $\mathbf{M}$  matrices in 2D and 3D for several values of the grid spacing  $\Delta x$  and degree of interpolation  $p$ . The effects of the ghost points  $L_{\text{ghost}}$  are noticeable on the right of  $\Delta_h$  and the bottom of  $\mathbf{E}$ . The final  $\mathbf{M}$  has limited bandwidth which is related to the ordering of the  $L_{\text{evolve}}$  and  $L_{\text{ghost}}$  chosen by the banding algorithm in Section 3.5. Table 3.2 shows some sparsity properties of  $\mathbf{M}$  including the number of nonzero elements and the bandwidth.

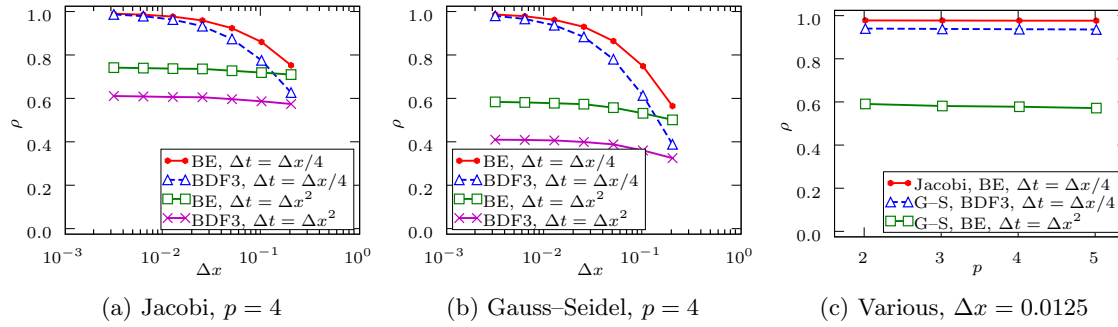


Figure 3.7: Spectral radii  $\rho$  for Jacobi and Gauss-Seidel iteration matrices against grid spacing  $\Delta x$  and interpolation order  $p$  for backward Euler and BDF3 time-stepping systems. Note that the higher-order BDF-3 scheme generally has smaller  $\rho$  than the backward Euler scheme. Also note in (c) that  $p$  has a negligible effect on  $\rho$ .

Table 3.2: Sparsity properties of the matrix  $M$  for a unit circle in 2D. As  $\Delta x$  increases and degree of interpolation  $p$  increases, the percentage of nonzero entries increases. Roughly the same relation is reflected in the bandwidth as a percentage of total width.

$\Delta x$	nonzero entries (percentage)				bandwidth (percentage)			
	$p = 2$	$p = 3$	$p = 4$	$p = 5$	$p = 2$	$p = 3$	$p = 4$	$p = 5$
0.2	12.8	16.3	18.2	21.3	31.7	55.4	51.5	65.6
0.1	6.5	8.0	9.0	10.5	15.8	21.4	27.2	34.1
0.05	3.3	3.9	4.5	5.2	8.8	11.0	18.2	16.3
0.025	1.6	1.9	2.3	2.6	5.6	6.4	8.6	8.1
0.0125	0.8	1.0	1.1	1.3	2.9	2.8	4.5	3.9

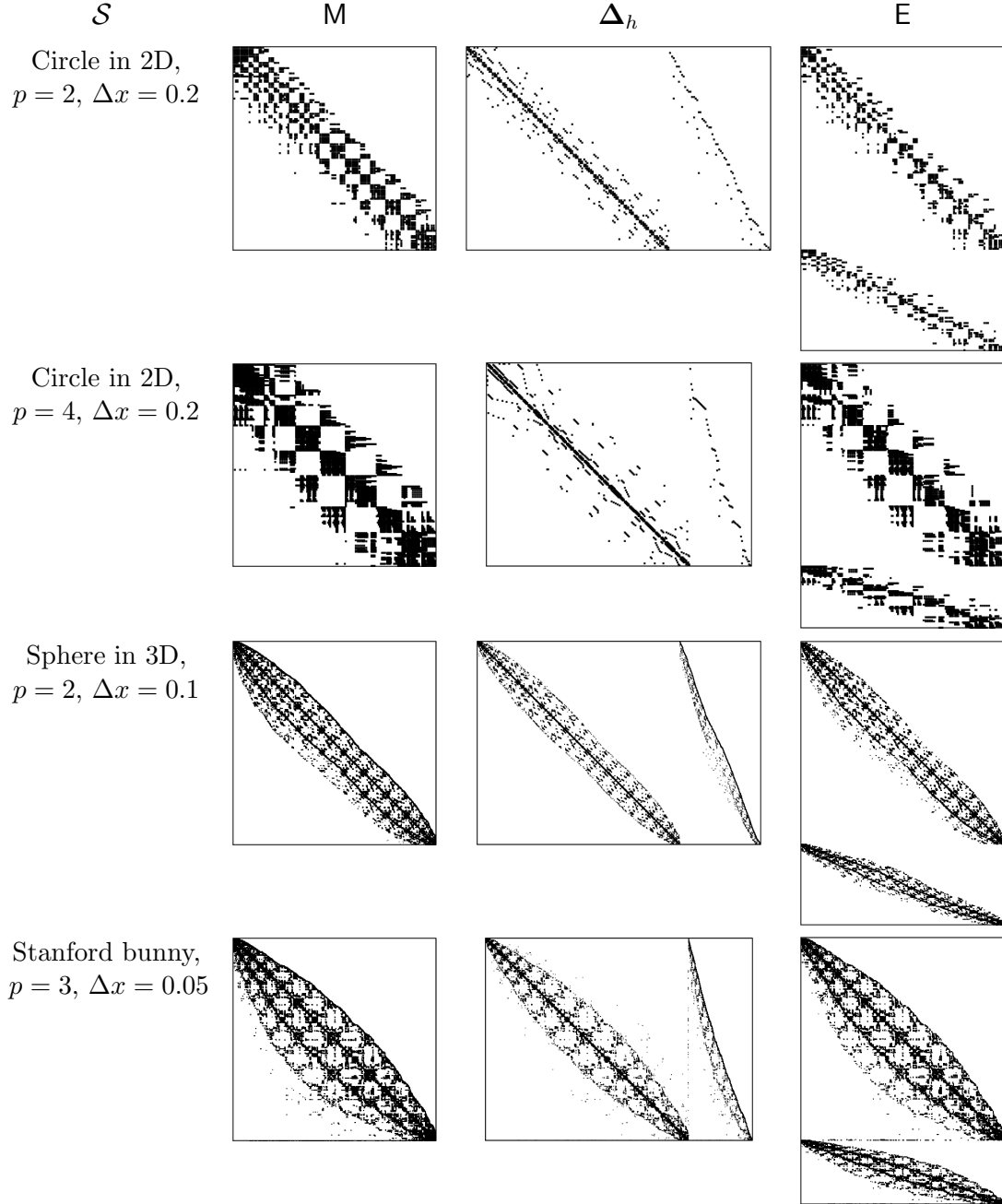


Figure 3.8: Sparsity structure of the  $M$ ,  $\Delta_h$  and  $E$  matrices (from left to right) for the in-surface heat equation on various surfaces using degree  $p$  interpolation.

Table 3.3: Experimentally determined values of  $\alpha$  for which  $\mathbf{A}_{\text{BE}} = \mathbf{I} - \Delta t \mathbf{M}$  is diagonally dominant with  $\Delta t \leq \alpha \Delta x^2$ .

$\Delta x$	$\alpha$ (circle in 2D)				$\alpha$ (sphere in 3D)			
	$p = 2$	$p = 3$	$p = 4$	$p = 5$	$p = 2$	$p = 3$	$p = 4$	$p = 5$
0.2	0.665	0.618	0.403	0.396	0.306	0.316	0.166	0.140
0.1	0.505	0.599	0.342	0.373	0.273	0.333	0.175	0.198
0.05	0.519	0.623	0.343	0.383	0.286	0.320	0.169	0.185
0.025	0.534	0.625	0.355	0.391	0.273	0.328	0.174	0.192
0.0125	0.507	0.595	0.324	0.371	0.285	0.324	0.171	0.179

### 3.3.3 Diagonal dominance

The standard 2D 5-point discrete Laplacian is weakly diagonally dominant. This is not the case with the implicit Closest Point Method matrices  $\mathbf{M}$  or  $\tilde{\mathbf{M}}$ ; neither is diagonally dominant, although  $\mathbf{M}$  comes closer than  $\tilde{\mathbf{M}}$ .

The standard 2D 5-point discrete Laplacian results in a strictly diagonally dominant backward Euler matrix  $\mathbf{A}_{\text{BE}} = \mathbf{I} - \Delta t \mathbf{\Delta}_h$  for all  $\Delta t$ . However, this is again not the case when either  $\mathbf{M}$  or  $\tilde{\mathbf{M}}$  replaces  $\mathbf{\Delta}_h$ . Table 3.3 demonstrates experimentally that  $\mathbf{I} - \Delta t \mathbf{M}$  is diagonally dominant only for  $\Delta t \leq \alpha \Delta x^2$  where  $\alpha$  depends on the degree of interpolation  $p$  and the dimension  $d$  but not significantly on  $\Delta x$ . Values of  $\alpha$  vary from 0.15 to 0.6. Further calculations, not included, show that for  $\tilde{\mathbf{M}}$ , the values of  $\alpha$  are even smaller. For practical calculations, we wish to use larger time steps  $\Delta t = \mathcal{O}(\Delta x)$ , and for these step sizes, the backward Euler matrix  $\mathbf{I} - \Delta t \mathbf{M}$  is not diagonally dominant.

### 3.3.4 Positive definiteness

The Closest Point Method matrices  $\mathbf{M}$  and  $\tilde{\mathbf{M}}$  are generally not symmetric (as noted above their spectra are not purely real). Thus they cannot be symmetric positive definite. Additionally, they are indefinite in the sense of Golub and van Loan because the symmetric part  $A_S = \frac{1}{2}(A + A^T)$  has both positive and negative eigenvalues [GVL96].

It follows that the time-stepping matrices of the form  $\mathbf{I} - \gamma \Delta t \mathbf{M}$  are also not symmetric. However, Table 3.4 shows that  $\mathbf{A}_{\text{BE}}$  is unsymmetric positive definite for restricted time steps  $\Delta t \leq \alpha \Delta x^2$  for experimentally determined values of  $\alpha$  between 0.15 and 0.5. In fact the values of  $\alpha$  agree roughly with those for diagonal dominance in Table 3.3. Again, these time steps are too restrictive for practical usage, and  $\mathbf{A}_{\text{BE}}$  is certainly indefinite for  $\Delta t = \mathcal{O}(\Delta x)$ .



Table 3.4: Experimentally determined values of  $\alpha$  for which  $\mathbf{A}_{\text{BE}} = \mathbf{I} - \Delta t \mathbf{M}$  is unsymmetric positive definite with  $\Delta t \leq \alpha \Delta x^2$ .

$\Delta x$	$\alpha$ (circle in 2D)				$\alpha$ (sphere in 3D)			
	$p = 2$	$p = 3$	$p = 4$	$p = 5$	$p = 2$	$p = 3$	$p = 4$	$p = 5$
0.2	0.538	0.450	0.329	0.298	0.282	0.236	0.162	0.143
0.1	0.517	0.427	0.300	0.280	0.286	0.231	0.167	0.149
0.05	0.503	0.417	0.298	0.273	0.275	0.223	0.164	0.146
0.025	0.496	0.416	0.296	0.271	0.269	0.216	0.159	0.143

### 3.3.5 Summary of matrix properties

Performing time stepping for the implicit Closest Point Method results in linear systems of the form  $\mathbf{A}_{\text{BE}} \mathbf{u} = \mathbf{b}$  with  $\mathbf{A}_{\text{BE}} = [\mathbf{I} - \Delta t \mathbf{M}]$  and similarly for  $\mathbf{A}_{\text{CN}}$ ,  $\mathbf{A}_{\text{BDF2}}$  and  $\mathbf{A}_{\text{BDF3}}$ . As noted above, the time-stepping matrices are well-conditioned, and thus direct solves based on variations of LU factorization work well, at least for small systems such as 2D with large grid spacing  $\Delta x$ . The MATLAB backslash operator and `scipy.linalg.solve` in SciPy [JOP<sup>+</sup>01] are used in many of the calculations in Section 3.6.

The time-stepping matrices become increasingly sparse as  $\Delta x$  decreases, and thus the direct methods tend to use more memory due to “fill-in” from the LU factorization. For larger systems, particularly in 3D, iterative schemes become more efficient. For time steps of  $\Delta t = \mathcal{O}(\Delta x)$ , the time-stepping matrix  $\mathbf{A}_{\text{BE}}$  is not positive definite nor diagonally dominant (which would guarantee convergence of many iterative schemes). However, the spectral radii of the Jacobi and Gauss–Seidel iteration matrices are strictly less than one, and thus we can expect convergence for those methods. In Section 3.6, the GMRES algorithm [TB97] is used, which does not require symmetry of  $\mathbf{A}_{\text{BE}}$  and seems to converge well in practice. The algorithm is used in Python as `scipy.linalg.gmres` and MATLAB with the function `gmres`.

Finally, note that it is not necessary to explicitly construct  $\mathbf{M}$  and  $\mathbf{A}_{\text{BE}}$  in order to use these iterative schemes. For example, GMRES requires only a subroutine that computes the product  $\mathbf{A}_{\text{BE}} \mathbf{u}$ . However, in the numerical results that follow, the matrices  $\mathbf{M}$ ,  $\mathbf{A}_{\text{BE}}$  and  $\mathbf{A}_{\text{BDF3}}$  have been used.

## 3.4 Error Analysis

This section presents an analysis of the discretization error in the Closest Point Method approximation of the Laplace–Beltrami operator. This error has two components, arising

from the discrete extension operator  $\mathbf{E}$  and from the discretization of the Laplacian operator  $\Delta_h$ .

Assume that  $u(x)$  is a smooth  $C^{p+1}(\mathbb{R}^d)$  scalar function where  $p$  is the degree of interpolation to be used. Let  $\mathbf{u} \in \mathbb{R}^m$  be the vector of values at points in  $L_{\text{evolve}}$  and consider a point  $\mathbf{x} \in \mathbb{R}^d$  on the surface  $\mathcal{S}$ . Then the degree- $p$  interpolation of the values in  $\mathbf{u}$  to approximate  $u(\text{cp}(\mathbf{x}))$  results in

$$u(\text{cp}(\mathbf{x})) = \mathbf{E}\mathbf{u} + \mathcal{O}((\mathbf{x} - \mathbf{x}_0)^{p+1}),$$

where  $\mathbf{x}_0$  is a nearby grid point to  $\text{cp}(\mathbf{x})$ . Formally, two derivatives of the  $\mathcal{O}((\mathbf{x} - \mathbf{x}_0)^{p+1})$  term results in an  $\mathcal{O}((\mathbf{x} - \mathbf{x}_0)^{p-1})$  term. Thus consider the Laplacian

$$\Delta u(\text{cp}(\mathbf{x})) = \Delta(\mathbf{E}\mathbf{u}) + \mathcal{O}((\mathbf{x} - \mathbf{x}_0)^{p-1}).$$

Then by approximating the Laplacian  $\Delta$  with  $\Delta_h$  using second-order centered finite differences we obtain

$$\Delta u(\text{cp}(\mathbf{x})) = \Delta_h \mathbf{E}\mathbf{u} + \underbrace{\mathcal{O}(\Delta x^2)}_{\Delta \text{ disc. error}} + \underbrace{\mathcal{O}(\Delta x^{p-1})}_{\text{interp. error}}. \quad (3.11)$$

This analysis indicates that the interpolation scheme should be chosen to have  $p \geq 3$  to obtain a second-order semi-discretization of  $\Delta u(\text{cp}(\mathbf{x}))$ , and hence with suitable time stepping, an overall second-order solution to the in-surface heat equation (3.1). Section 3.6.1 compares (3.11) to numerical convergence studies.

### 3.5 Constructing the Computational Band

This section presents an algorithm to construct the band of grid points  $L_{\text{evolve}}$  enveloping the surface  $\mathcal{S}$  and the band of ghost points  $L_{\text{ghost}}$ . The algorithm is related to the “stencil set approach” in Section 2.3 and differs mainly in that for the implicit Closest Point Method we need to keep track of the indices and weights associated with evolution and extension so that we can build the matrix  $\mathbf{M}$ ; in Chapter 2 we merely need to explicitly execute these operations, and thus solutions and other data were organized primarily by their indices in a grid of the  $d$ -dimensional embedding space. Here we build data structures based on the position of grid points in the two lists of points  $L_{\text{evolve}}$  and  $L_{\text{ghost}}$ . These lists are related to the  $S_{\text{evolve}}$  and  $S_{\text{extend}}$  sets of Section 2.3 in the sense that  $L_{\text{evolve}} = S_{\text{evolve}}$  and  $L_{\text{evolve}} \cup L_{\text{ghost}} = S_{\text{extend}}$ .

The algorithm proceeds in two passes. In the first pass, we build the list  $L_{\text{evolve}}$  of all points needed for the closest point extension; these are exactly the points on which the approximate solution will be propagated. This pass also identifies special *base points*  $\mathbf{b} \in L_{\text{evolve}}$  which are grid points that appear in the lower-left corner of an interpolation stencil (for example, the grid points in the lower-left corner of each of the shaded regions in Figure 3.1). Each of these base points stores the corresponding interpolation stencil as a list  $S_{\text{interp}}^{\mathbf{b}}$  of indices pointing into the list  $L_{\text{evolve}}$ .

The second pass adds the list  $L_{\text{ghost}}$  of ghost points; these grid points appear in an evolution stencil but not in any of the interpolation stencils. Each point  $\mathbf{x}$  in  $L_{\text{evolve}} \cup L_{\text{ghost}}$  stores  $\text{cp}(\mathbf{x})$ , its closest point on  $\mathcal{S}$ , and the index (pointing into  $L_{\text{evolve}}$ ) of the base point  $\mathbf{b}$  of its interpolation stencil. Points  $\mathbf{x}$  in  $L_{\text{evolve}}$  also store their evolution stencil as a list  $S_{\text{evolve}}^{\mathbf{x}}$  of indices pointing into  $L_{\text{evolve}}$  and  $L_{\text{ghost}}$  (recall that some of those evolution points will be in  $L_{\text{evolve}}$  and some of those points may be in  $L_{\text{ghost}}$ , see Figure 3.1).

More detailed algorithms for the two passes are included in Appendix A.1.

Table 3.5 shows the resulting band sizes and computation times required to construct  $L_{\text{evolve}}$  and  $L_{\text{ghost}}$  for various  $\Delta x$ . Note that for a curve embedded in 2D, the number of points in the band scales like  $\mathcal{O}(\frac{1}{\Delta x})$ : i.e., not like a full discretization of 2D which would scale like  $\mathcal{O}(\frac{1}{\Delta x^2})$ . In this sense, the Closest Point Method is (asymptotically) optimal in the number of computation points required because it has the same scaling as a parameterized discretization of the curve.

After building and storing the computational band, it is straightforward to compute the  $\Delta_h$  and E and M matrices.

### 3.5.1 Forming the $\Delta_h$ matrix

From the  $L_{\text{evolve}}$  and  $L_{\text{ghost}}$  lists and associated data structures, the matrix  $\Delta_h$  is constructed as follows.

- Loop over each index  $i$  where  $\mathbf{x}_i \in L_{\text{evolve}}$ .
- Loop over each index  $j \in S_{\text{evolve}}^{\mathbf{x}_i}$ , the evolution stencil for  $\mathbf{x}_i$ .
  - Set  $[\Delta_h]_{i,j}$  to the appropriate weight for the  $j^{\text{th}}$  component of the evolution stencil.

### 3.5.2 Forming the $\mathbf{E}$ matrix

The discrete extension matrix  $\mathbf{E}$  is constructed as follows.

- Loop over each index  $i$  where  $\mathbf{x}_i \in L_{\text{evolve}} \cup L_{\text{ghost}}$ :
  - Retrieve the base point  $\mathbf{b}$  for  $\mathbf{x}_i$ .
  - Compute the weights  $S_{\text{weights}}^{\mathbf{b}}$  with barycentric Lagrange interpolation. This depends on the  $S_{\text{interp}}^{\mathbf{b}}$  and  $\text{cp}(\mathbf{x}_i)$
  - For each index  $j$  in the interpolation  $S_{\text{interp}}^{\mathbf{b}}$ :
    - Set  $[\mathbf{E}]_{i,j} = w_j$ , the appropriate weight for the  $j^{\text{th}}$  component of the interpolation stencil.

It is emphasized that most of the effort with respect to programming the computer implementation of the implicit Closest Point Method is in constructing the computational band: explicitly constructing  $\Delta_h$ ,  $\mathbf{E}$  and their stabilized product  $\mathbf{M}$  is straightforward as the above algorithms suggest. The algorithm for building the computational band  $L_{\text{evolve}}$  and  $L_{\text{ghost}}$  spends most of its time searching the current candidate lists  $L_{\text{evolve}}$  and  $L_{\text{ghost}}$  for particular points. The long running times demonstrated in Table 3.5 would benefit from improvements to this searching routine (which currently simply searches each list from the beginning for every search). However, in practice this does not impact the running time to compute a numerical solution to a surface PDE problem because finding the computational band for a particular surface  $\mathcal{S}$ , grid spacing  $\Delta x$ , evolution stencil and interpolation stencil is effectively a preprocessing step which is done once and the resulting  $L_{\text{evolve}}$  and  $L_{\text{ghost}}$  stored for later use.

## 3.6 Numerical Results

### 3.6.1 Numerical convergence studies

The convergence of the implicit Closest Point Method is tested on two test problems.

**2D test problem** The problem consists of the in-surface heat equation (3.1) on a unit circle with initial conditions  $u(0, \theta) = \cos 2\theta$ . It is solved discretely in time until  $t = T_f = \frac{1}{2}$  using the BDF-3 scheme with  $\Delta t = \frac{1}{4}\Delta x$ . The exact solution is  $u(t, \theta) = e^{-4t} \cos 2\theta$ . The

Table 3.5: Properties of the computation band for a unit circle in 2D and a unit sphere in 3D for a second-order centered finite difference approximate to Laplacian with degree  $p = 4$  interpolation. The “band time” column shows how long it took to generate the bands on a particular machine, “matrix time” shows how long it took to form the matrices from the band structures.

$N$ $= \frac{1}{\Delta x}$	2D			3D				
	band size		band time	band size		band time	matrix time	
	$L_{\text{evolve}}$	$L_{\text{ghost}}$		$L_{\text{evolve}}$	$L_{\text{ghost}}$		$\Delta_h$	E
2.5	96	36	0.2s	896	480	3.75s	0.1s	1s
5	186	56	0.3s	2425	863	13.7s	0.5s	3s
10	386	116	0.8s	9433	2480	108s	1.3s	9s
20	786	228	1.7s	37657	8688	1420s	5.7s	35s
40	1586	452	3.6s	150697	33752	20699s	21s	142s
80	3186	908	8.6s					
160	6386	1812	24s					
320	12786	3620	79s					
640	25586	7244	278s					
1280	51186	14484	1103s					
2560	102386	28964	4426s					

problem is embedded in 2D. Linear systems are solved directly with MATLAB's `backslash` operator.

**3D test problem** The problem consists of the in-surface heat equation on a unit sphere with initial conditions  $u(\theta, \phi) = \cos(\theta)$ . As this is a spherical harmonic function of degree 1, the solution is  $u(t, \theta, \phi) = e^{-2t} \cos \theta$  [Pow99]. The problem is again discretized in time until  $t = T_f = \frac{1}{2}$  using the BDF-3 scheme with  $\Delta t = \frac{1}{4} \Delta x$ . The embedding space is  $\mathbb{R}^3$ . Linear system solves are performed in MATLAB with GMRES using a tolerance of  $1 \times 10^{-10}$ .

### Heat equation, second-order differences in space

Figure 3.9 shows the results of a numerical convergence study on these two problems using second-order finite-differences (Figure 3.2) and various degrees  $p$  of interpolating polynomials. We note clear second-order convergence using  $p \geq 3$ . The results are consistent with the analysis in Section 3.4. The computational results appear identical for  $p \geq 4$  because the error is dominated by the  $\Delta$  discretization error term in (3.11) and the interpolation error term is insignificant. Degree  $p = 3$  still exhibits second-order convergence but with a larger error constant which is likely due to the interpolation error term dominating in (3.11). Note that  $p = 2$  behaves surprisingly well (often better than the first-order predicted by (3.11)) and that  $p = 2$  is the minimum degree of interpolation as  $p = 0$  and  $p = 1$  appear inconsistent. This latter result is in good correspondence with the error analysis as (3.11) suggests that for  $p \leq 1$  the error will not decay as  $\Delta x$  decreases.

Larger values for the interpolation degree  $p$  result in larger interpolation stencils which translates into denser matrices (see Table 3.2). Degree  $p = 3$  or  $p = 4$  are thus good choices as both exhibit clear second-order convergence; although note the error for  $p = 3$  is roughly five times that of  $p = 4$ . Degree  $p \geq 5$  would be overkill as no benefit in accuracy results from the extra work required. Degree  $p = 2$  may be appealing for applications where calculation speed is more important than accuracy.

### Heat Equation, fourth-order differences in space

The convergence of the implicit Closest Point Method is next tested using the fourth-order centered finite difference stencil in Figure 3.2 for  $\Delta_h$ . Time stepping is still performed with the third-order BDF-3 scheme, so overall third-order results are expected. Figure 3.10 shows the results of a numerical convergence study on the two test problems above. We

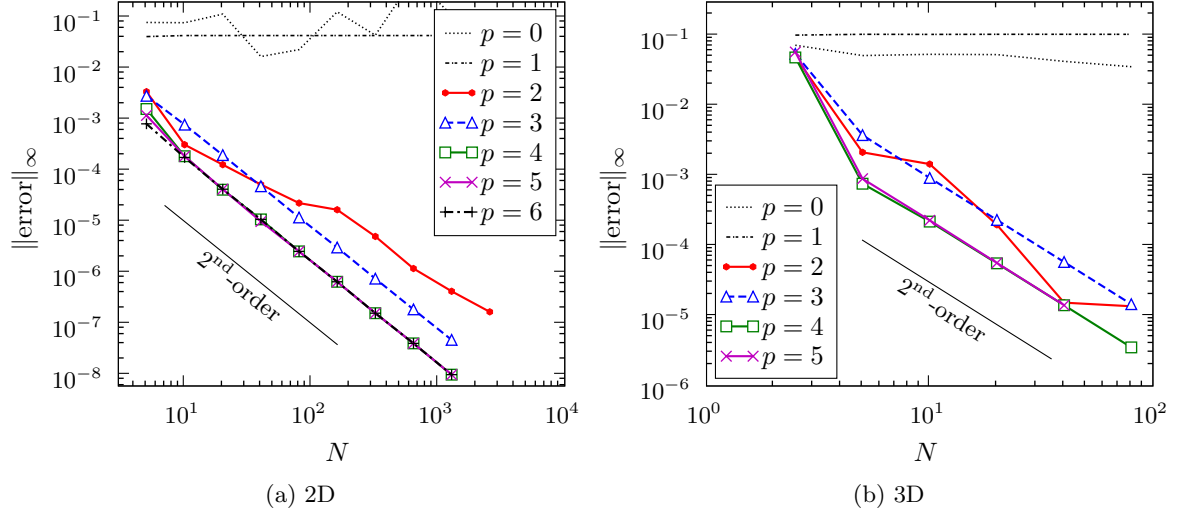


Figure 3.9: Numerical convergence study results for the in-surface heat equation in 2D and 3D using second-order finite differences. Here  $N = \frac{1}{\Delta x}$ .

note again that  $p = 0$  and  $p = 1$  degree interpolation appear inconsistent. Degree  $p = 2$  or  $p = 3$  produce second-order results. Degree  $p \geq 4$  produces at least the third-order convergence anticipated. In Figure 3.10(a), the error stagnates around  $10^{-11}$ : this is likely due to roundoff error and is discussed further in Section 3.6.3 where the effect is more pronounced.

### 3.6.2 Heat equation on surfaces

The computations in this section were performed in Python [vR<sup>+</sup>91] using SciPy [JOP<sup>+</sup>01] and NumPy [Oli07] for numerical calculations. Visualizations use VTK [SML98].

#### Heat equation on Laurent's Hand

Figure 3.11 shows the results of blurring an image on the surface of Laurent's Hand [SAA07] by solving several steps of the in-surface heat equation. Note the phone number written on the hand is completely indistinguishable after only a few steps. The Closest Point Method could also be used to apply an in-surface unsharp mask or other image processing technique to attempt to recover the text from the blurred image: these applications to image processing on surfaces are proposed for future work.

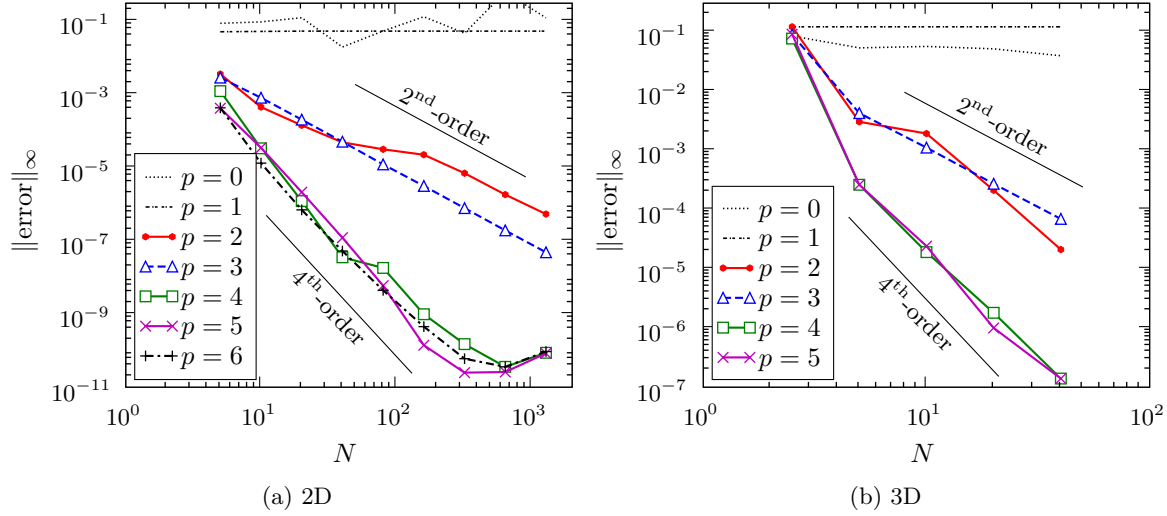


Figure 3.10: Numerical convergence study results for the in-surface heat equation in 2D and 3D using fourth-order finite differences. Here  $N = \frac{1}{\Delta x}$ .

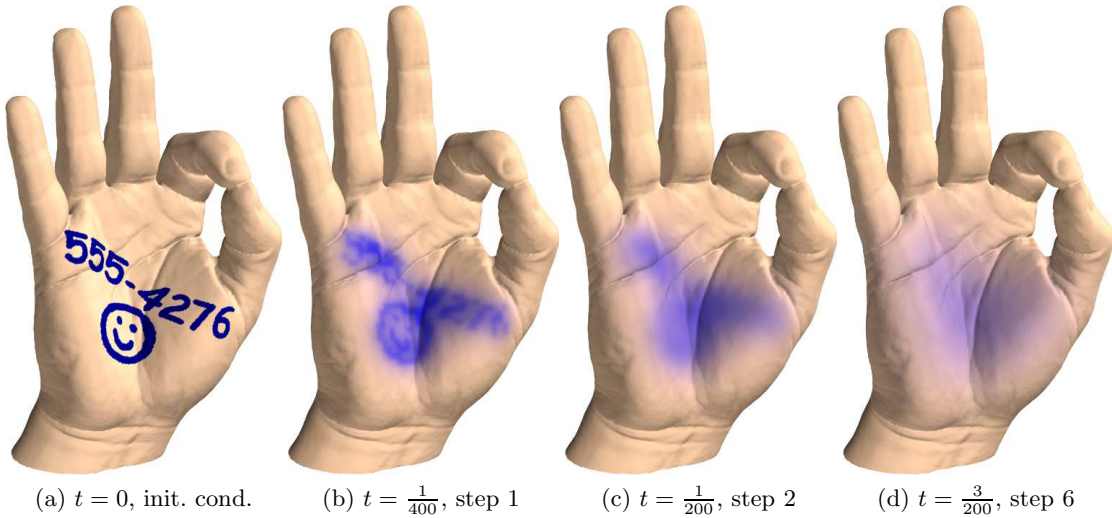


Figure 3.11: Blurring on the surface of Laurent's Hand [SAA07] by solving the in-surface heat equation. Here  $\Delta x = \frac{1}{40}$  and the length of the hand is about  $80\Delta x$ . Time stepping is performed with backward Euler and  $\Delta t = \frac{\Delta x}{10}$ . Closest point extensions use  $p = 2$ .



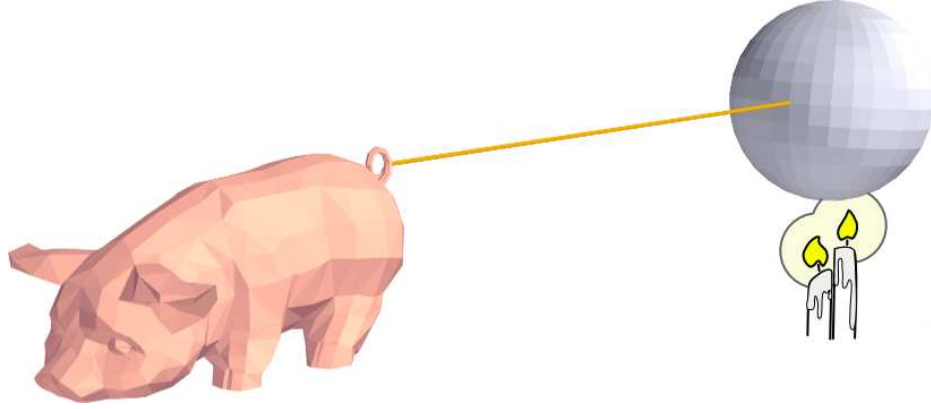


Figure 3.12: A composite domain consisting of Annie Hui's pig [Hui08] connected by the tail to a filament which is connected to a sphere. A heat source is applied under the sphere (modelled as a local forcing consisting of Newton's law of heating and cooling). (The candle image is modified from a public domain image by Pascal Terjan.)

### Heat equation on a complicated domain

Consider a composite domain consisting of Annie Hui's pig [Hui08] connected by the tail to a filament (infinitesimally thin wire) which is also connected to a sphere (see Figure 3.12). This surface is comprised of components of various codimension: the pig and sphere are codimension 1 whereas the filament is codimension 2. Nonetheless we will see that this poses no particular difficulty for the implicit Closest Point Method. A heat source is applied under the sphere and the temperature  $u$  over the surface of the domain is modelled according to

$$u_t = \kappa(\mathbf{x}) \Delta_{\mathcal{S}} u + k(T - u(\mathbf{x})) L(\mathbf{x}), \quad (3.12a)$$

where the heat coefficient  $\kappa(\mathbf{x})$  is chosen as

$$\kappa(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in \text{Pig}, \\ 2 & \mathbf{x} \in \text{Sphere}, \\ 10 & \mathbf{x} \in \text{Filament}, \end{cases} \quad (3.12b)$$

and the forcing term consists of Newton's law of heating and cooling with constant  $k = 30$  and maximum temperature  $T = 10$ . The function  $L(\mathbf{x})$  is a Gaussian which localizes the heat source directly under the sphere. Initially the temperature is set to zero everywhere. The problem is solved until  $t = 25$  using  $\Delta x = 0.05$  and degree  $p = 3$  interpolation. Time stepping is performed by the BDF2 scheme with  $\Delta t = 2\Delta x$ . Thus, only 250 steps are used

in contrast with the 10,000 steps required for explicit time stepping for this problem. The closest point representation is straightforward to obtain for this type of composite domain: simply compute the closest point in each of the three components and then return the closest of the three. This closest point calculation can be further accelerated by using bounding boxes around the individual components (particularly the pig which has a nontrivial cp function as described in Section 2.4.5)

Figure 3.13 shows the initial conditions and heat distribution at  $t = 25$ . Note that the surface of the pig has not significantly increased in temperature. This example demonstrates the great flexibility of the method with respect to geometry: heat flow over this complicated geometry of variable codimension is no more complicated than it would be over a sphere.

### 3.6.3 Biharmonic problems

Consider the fourth-order problem

$$u_t = -\nabla_S^4 u, \quad (3.13)$$

where  $\nabla_S^4$  is the biharmonic operator intrinsic to the surface.

In  $\mathbb{R}^2$ , the biharmonic operator is

$$-\nabla^4 = -\frac{\partial^4}{\partial x^4} - \frac{\partial^4}{\partial y^4} - 2\frac{\partial^4}{\partial x^2 \partial y^2}, \quad (3.14)$$

and thus one approach to the Closest Point Method solution of (3.13) is to discretize (3.14) and use the resulting matrix (say  $D_{bi}$  using the stencil shown in Figure 3.14) combined with the discrete extension operator  $E$  in the construction of

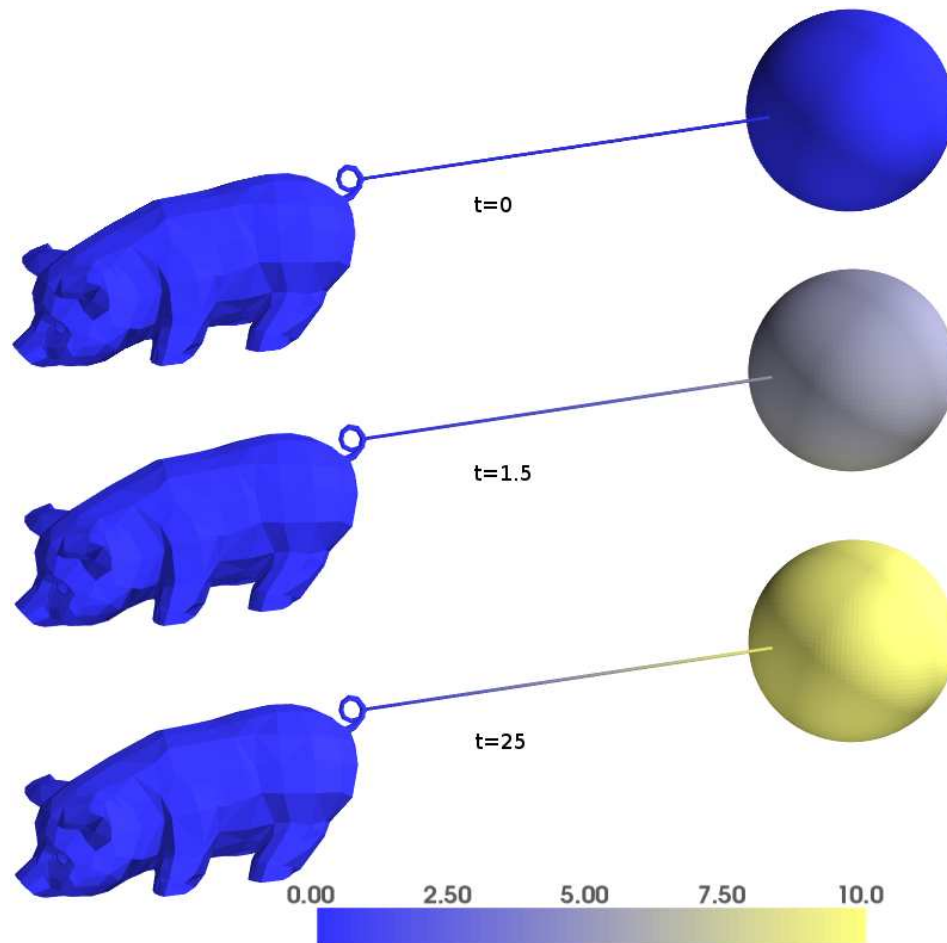
$$\widehat{M}_{bi} = \text{stab}(D_{bi}, E) = \text{diag}(D_{bi}) + (D_{bi} - \text{diag}(D_{bi}))E,$$

and then solve the semi-discrete system

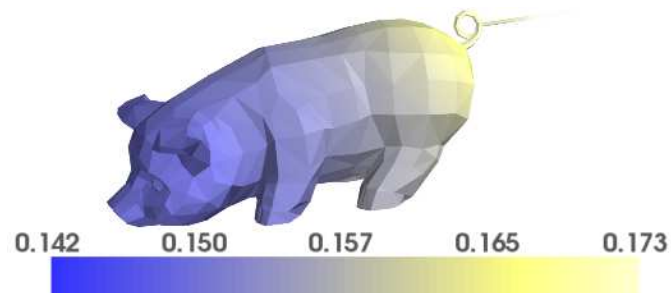
$$\frac{\partial}{\partial t} \mathbf{u} = \widehat{M}_{bi} \mathbf{u}.$$

However, as the principles in Section 1.6.4 do not explicitly hold for the biharmonic operator, we have little reason to expect the Closest Point Method based on  $\widehat{M}_{bi}$  to be effective. In fact Figure 3.15 demonstrates that although the method appears stable, it is not consistent.

In Section 1.6.4 (and originally in [RM08]), it was suggested that operators such as the biharmonic operator which are composed of multiple surface gradient  $\nabla_S$  or Laplace–Beltrami  $\Delta_S$  operators would be implemented using multiple extension steps. This idea



(a) Initial conditions (top), solution at  $t = 1.5$  (middle), and at  $t = 25$  (bottom)



(b) Rescaled to show heat distribution on pig surface at  $t = 25$

Figure 3.13: Numerical solution of the in-surface heat equation on the composite pig-wire-sphere domain. At  $t = 25$ , the temperature is roughly constant at  $u = 10$  in the sphere. The pig has warmed up only slightly and there is a gradient in temperature in the wire.

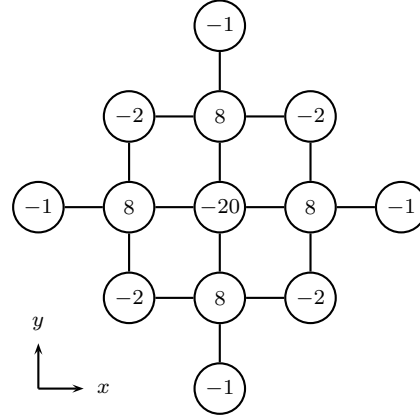


Figure 3.14: A possible second-order stencil for the biharmonic operator in 2D.

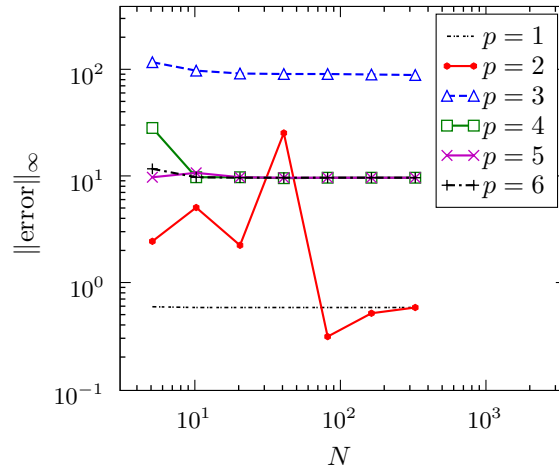


Figure 3.15: Biharmonic numerical convergence study results in 2D using the operator  $\widehat{M}_{bi}$  which applied second-order finite differences directly to the biharmonic operator. Results appear inconsistent. Note  $N = \frac{1}{\Delta x}$ .

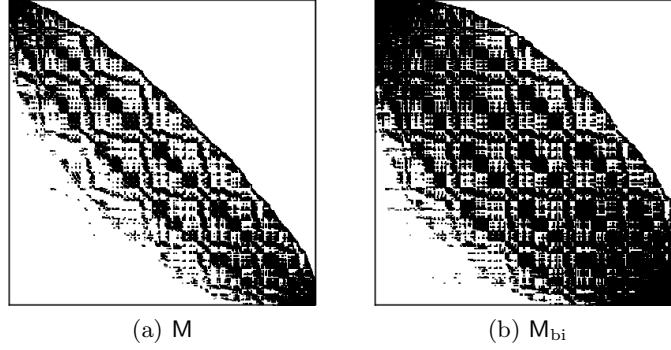


Figure 3.16: Comparison of the sparsity structure of  $M$  and  $M_{bi}$ . Note fill-in apparent in  $M_{bi}$ . Geometry: 2D embedding of a unit circle using degree  $p = 4$  interpolation and  $\Delta x = 0.1$ .

can be implemented straightforwardly in the implicit Closest Point Method. Recall that  $-\nabla_{\mathcal{S}}^4 u = -\Delta_{\mathcal{S}}(\Delta_{\mathcal{S}} u)$  and thus, by the principles in Section 1.6.4,

$$-\nabla_{\mathcal{S}}^4 u(\mathbf{x}) = -\Delta(\text{cp}(\Delta(\text{cp}(\mathbf{x})))),$$

for points  $\mathbf{x}$  on the surface. With  $\Delta_h$  as the discrete diffusion operator from Section 3.2.2, we approximate this expression as

$$\widetilde{M}_{bi} = -\Delta_h E \Delta_h E. \quad (3.15)$$

This matrix  $\widetilde{M}_{bi}$  has eigenvalues with positive real components, and thus for stability, we use the stabilizing procedure from Section 3.2.3 to form the matrix

$$M_{bi} = -\text{stab}(\Delta_h, E) \text{stab}(\Delta_h, E) = -MM, \quad (3.16)$$

and use this for computation instead. Note that this is a “one-line” change in computer code for the surface heat equation—requiring only the squaring of the  $M$  matrix. However this operation results in a fair amount of “fill-in” in as can be seen by a comparison of the structure of  $M_{bi}$  and  $M$  in Figure 3.16. This fill-in typically results in higher cost in terms of processor time and memory for solving biharmonic problems: however, the fill-in can likely be avoided altogether with an iterative method.

The convergence of the method is tested on two problems. The first consists of (3.13) on a unit circle with initial conditions  $u(0, \theta) = \cos \theta + 6 \cos 2\theta$  solved discretely in time until  $t = T_f = \frac{1}{2}$  using the BDF-3 scheme with  $\Delta t = \frac{\Delta x}{4}$ . The exact solution is  $u(t, \theta) = e^{-t} \cos \theta + 6e^{-16t} \cos 2\theta$ . Linear systems are solved directly with MATLAB’s backslash operator.

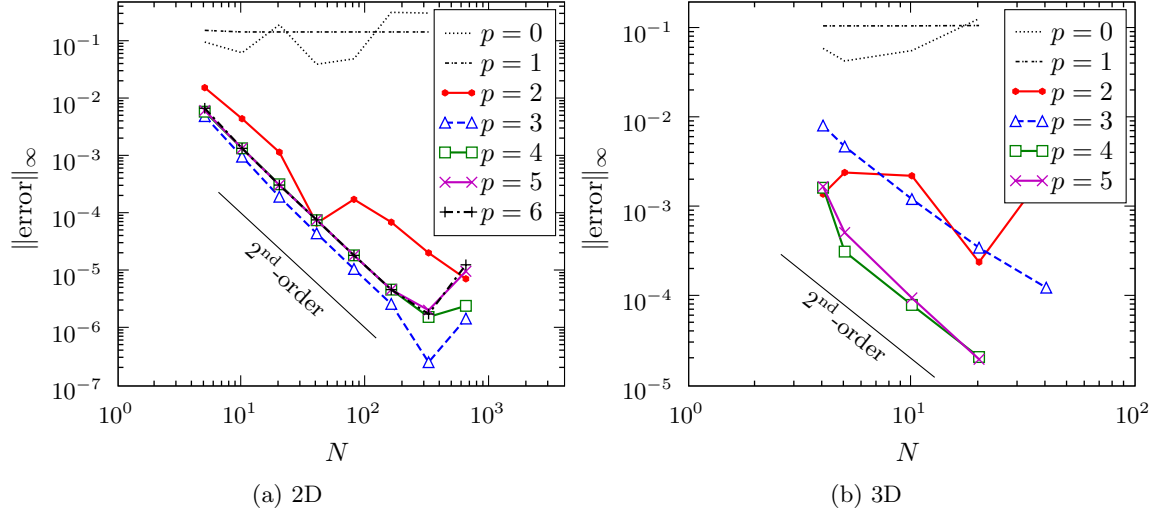


Figure 3.17: Biharmonic numerical convergence studies in 2D and 3D using the operator  $M_{bi}$  which uses a composition of the implicit Closest Point Method discrete Laplace–Beltrami operator  $M$ . Note  $N = \frac{1}{\Delta x}$ .

The second test problem consists of (3.13) on a unit sphere with initial conditions  $u(\theta, \phi) = \cos(\theta)$ . As this function is a spherical harmonic of degree 1, the solution is  $u(t, \theta, \phi) = e^{-4t} \cos \theta$ . The problem is discretized in time until  $t = T_f = \frac{1}{2}$  using the BDF-3 scheme with  $\Delta t = \frac{1}{4}\Delta x$ . Linear system solves are performed in MATLAB with GMRES using a tolerance of  $1 \times 10^{-10}$ .

Figure 3.17 demonstrates that the Closest Point Method achieves the expected second-order accuracy on both test problems although the error appears to stagnate, in this case around  $10^{-6}$ . Note that the entries in the  $M_{bi}$  matrix are each multiplied by  $\frac{1}{\Delta x^4}$ . Thus, rounding errors could be magnified by roughly  $10^8$  because  $\Delta x = \mathcal{O}(10^{-2})$  and  $\frac{1}{\Delta x^4} = \mathcal{O}(10^8)$ . Overall, the calculations demonstrate that the implicit Closest Point Method is viable for biharmonic problems, although for highly accurate results, some attention should be paid to the control of roundoff error.

### 3.6.4 Pattern formation

The solutions of reaction-diffusion equations can exhibit rich pattern forming behaviour from random initial conditions. It is thought that this may be the mechanism for coat patterns on animals such as zebras or cheetahs [Mur03]. The application of reaction-diffusion equations

on surfaces has been investigated by previous studies, such as [Tur91] (by computing directly on a surface mesh) and [BCOS01] (using a level set representation of the surface). In this section, the implicit Closest Point Method is applied to pattern formation problems.

A possible set of reaction-diffusion equations for two chemicals  $u$  and  $v$  is

$$u_t = f(u, v) + \nu_u \Delta_S u, \quad (3.17a)$$

$$v_t = g(u, v) + \nu_v \Delta_S v, \quad (3.17b)$$

$$u(0, \mathbf{x}) = u_0(\mathbf{x}), \quad (3.17c)$$

$$v(0, \mathbf{x}) = v_0(\mathbf{x}), \quad (3.17d)$$

where  $f$  and  $g$  are nonlinear reaction terms. The chemicals locally react with one another while at the same time diffusing spatially at different rates. One particular form of (3.17) is known as the Brusselator [PL68, YZE04] and has

$$f(u, v) = a - (b - 1)u + u^2v, \quad g(u, v) = bu - u^2v.$$

The implicit Closest Point Method can be applied to (3.17) using the implicit-explicit (IMEX) schemes of Section 1.2.3. Recall that using these schemes, the nonlinear terms are treated explicitly and the diffusion terms are treated linearly. Additionally, the implicit system solves for  $u$  and  $v$  are decoupled from each other so the time stepping proceeds essentially the same as the previous heat equation examples, solving two systems, each of the form  $[I - \gamma \Delta t M] \mathbf{u} = \mathbf{b}$ , to advance to the next time step. The SBDF-2 scheme (Section 1.2.3) is known to work well for pattern formation problems [Ruu95], and it is used here with one step of IMEX Euler as a starting method. For initial conditions, small random perturbations around the zero-diffusion ( $\nu_u = \nu_v = 0$ ) steady state of  $u = a$  and  $v = \frac{b}{a}$  are used.

Depending on the values of the coefficients  $a$ ,  $b$ ,  $\nu_u$  and  $\nu_v$ , the Brusselator exhibits various types of pattern forming behaviour [YZE04]. Figure 3.18 shows the results of three simulations of the Brusselator on the surface of the Stanford Bunny [TL94] with  $a = 3$ ,  $b = 10.2$ ,  $\nu_v = 10/900$  and various values of  $\nu_u$ . This choice of parameters closely matches those used for planar calculations in [YZE04, Figure 2a)]<sup>2</sup>, and indeed we notice the same transition from “honeycomb” (Figure 3.18(a)) to stripes (Figure 3.18(b)) to roughly hexagonal patterns of spots (Figure 3.18(c)).

---

<sup>2</sup>We rescale the equations to the size of the bunny which accounts for the additional factors of  $\frac{1}{900}$  which do not appear in [YZE04].

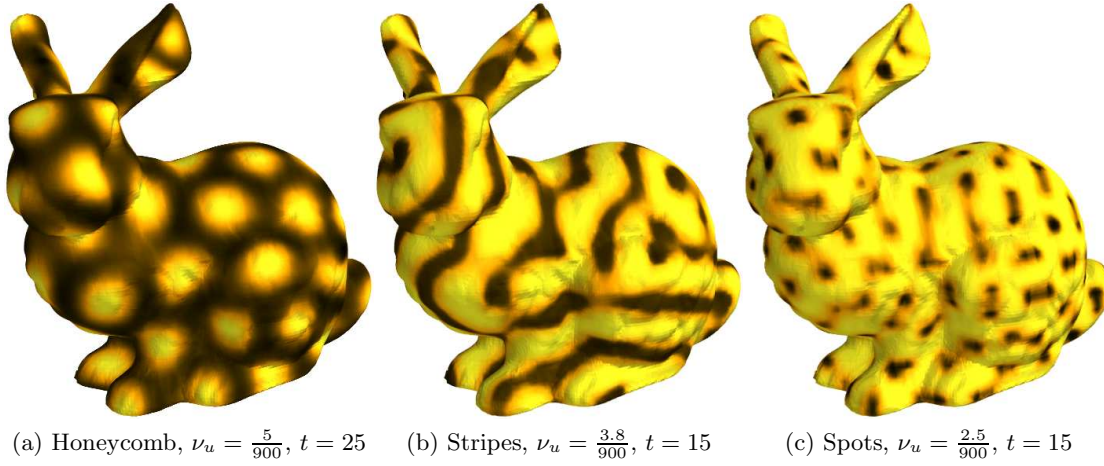


Figure 3.18: Patterns formed by the Brusselator on the surface of the Stanford Bunny for different values of  $\nu_u$  with  $a = 3$ ,  $b = 10.2$  and  $\nu_v = 10/900$ . Darker color indicates high concentrations of  $u$ . The computation uses  $\Delta x = 0.05$  with  $p = 2$  degree interpolation and the bunny is approximately 2 units long.

The results indicate the effectiveness of the Closest Point Method for reaction-diffusion systems on surfaces. An interesting application would be to study mammalian coat-pattern formation [Mur03] directly on the embryonic surface.

### 3.7 Nonlinear Problems

For most of this chapter, the implicit Closest Point Method has been applied to linear problems and nonlinear problems where the high-order terms appear linearly. The procedure can be generalized to fully nonlinear problems. Suppose, for example, that the surface PDE on  $\mathcal{S}$  takes the form

$$\begin{aligned} u_t &= f(u, \nabla_{\mathcal{S}} u, \Delta_{\mathcal{S}} u), \\ u(0, \mathbf{x}) &= u_0(\mathbf{x}). \end{aligned}$$

Consider the embedding PDE

$$\begin{aligned} u_t &= f(u(\text{cp}(\mathbf{x})), \nabla_{\mathcal{S}} u(\text{cp}(\mathbf{x})), \Delta_{\mathcal{S}} u(\text{cp}(\mathbf{x}))), \\ u(0, \mathbf{x}) &= u_0(\text{cp}(\mathbf{x})), \end{aligned}$$



and consider the semi-discretization

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{f}(\mathbf{E}\mathbf{u}), \quad (3.19a)$$

$$\mathbf{u}(0) = \mathbf{u}^0, \quad (3.19b)$$

where  $\mathbf{f}$  is a nonlinear function resulting from the spatial discretization of  $f(u, \nabla u, \Delta_S u)$ . The nonlinear system of ODEs could also arise from a nonlinear spatial discretization (e.g., WENO) of a linear PDE.

The system (3.19) could be discretized implicitly in time, and the nonlinear system solves performed with Newton's method or similar techniques [BF01]. If  $\mathbf{f}(\mathbf{u}) = \Delta_h \mathbf{u}$ , then (3.19) is equivalent to the unstable equation (3.9) and we may want to consider the diagonal stabilization as in Section 3.2.3. This is done by defining the *diagonal splitting function* of  $\mathbf{f}$  as

$$\mathfrak{F}_j(\mathbf{u}, \mathbf{z}) = \mathbf{f}(z_1, z_2, \dots, z_{j-1}, u_j, z_{j+1}, \dots, z_m), \quad j = 1, \dots, m,$$

that is, the  $j^{\text{th}}$  component of  $\mathfrak{F}(\mathbf{u}, \mathbf{z})$  is computed using the  $j^{\text{th}}$  component of  $\mathbf{u}$  for the  $j^{\text{th}}$  input of  $\mathbf{f}$  and components of  $\mathbf{z}$  for the other inputs of  $\mathbf{f}$ .

Using this splitting function, the stabilized semi-discrete system becomes

$$\begin{aligned} \frac{\partial}{\partial t} \mathbf{u} &= \mathfrak{F}(\mathbf{u}, \mathbf{E}\mathbf{u}), \\ \mathbf{u}(0) &= \mathbf{u}^0, \end{aligned}$$

where one can verify that in the case when  $\mathbf{f}$  is linear this splitting function recovers the stabilized form demonstrated in Section 3.2.3. For example, if  $\mathbf{f}(\mathbf{u}) = \Delta_h \mathbf{u}$ , then  $\mathfrak{F}(\mathbf{u}, \mathbf{E}\mathbf{u}) = \text{stab}(\Delta_h, \mathbf{E})\mathbf{u} = [\text{diag}(\Delta_h) + (\Delta_h - \text{diag}(\Delta_h))\mathbf{E}]\mathbf{u} = \mathbf{M}\mathbf{u}$ .

While the fully nonlinear procedure has not yet been tried in practice, there are certainly parabolic or mixed type nonlinear problems that could be posed on surfaces (for instance, the examples of Chapter 2 with curvature dependent terms). For some of these problems (for example applying implicit WENO [GMR06] on surfaces) it may be desirable for the implicit time-stepping schemes to have additional nonlinear stability properties. One such class of methods, the diagonally split Runge–Kutta methods, is investigated next in Chapter 4. Interestingly—and coincidentally—we will revisit the diagonally splitting function in another context in Chapter 4.

## Chapter 4

# Diagonally Split Runge–Kutta Methods

Diagonally split Runge–Kutta (DSRK) time discretization methods are a class of implicit time-stepping schemes which offer both high-order accuracy and a form of nonlinear stability known as unconditional contractivity. This combination is not possible within the classes of Runge–Kutta or linear multistep methods and therefore appears promising for the strong stability preserving (SSP) time-stepping community, which is generally concerned with computing oscillation-free numerical solutions of PDEs. Using a variety of numerical test problems, it will be shown that although second- and third-order unconditionally contractive DSRK methods do preserve the strong stability property for all time-step sizes, they suffer from order reduction at large time-step sizes. Indeed, for time steps larger than those typically chosen for explicit methods, these DSRK methods behave like first-order implicit methods. This is unfortunate, because it is precisely to allow a large time step that we choose to use implicit methods. These results suggest that unconditionally contractive DSRK methods are limited in usefulness as they are unable to compete with either the first-order backward Euler method for large time-step sizes or with Crank–Nicolson or high-order explicit SSP Runge–Kutta methods for smaller time-step sizes.

New stage order conditions for DSRK methods are derived, and it is shown that the observed order reduction is associated with the necessarily low stage order of the unconditionally contractive DSRK methods.

Most of the contents of this chapter appear in [MGR08].

## 4.1 Introduction

### 4.1.1 Strong stability preserving time stepping

Strong stability preserving (SSP) high-order time discretizations [Shu88, SO88, GST01] were developed for the solution of semi-discrete method-of-lines approximations of hyperbolic partial differential equations with discontinuous solutions. In such cases, carefully constructed spatial discretization methods guarantee a desired nonlinear or *strong* stability property (for example, that the solution be free of oscillations) when coupled with first-order forward Euler time stepping.

First, consider the system of  $m$  ordinary differential equations (ODEs)

$$\frac{\partial}{\partial t} \mathbf{u}(t) = \mathbf{f}(\mathbf{u}(t)), \quad (4.1a)$$

$$\mathbf{u}(0) = \mathbf{u}^0, \quad (4.1b)$$

typically arising from the spatial discretization of a partial differential equation (PDE). We assume the spatial discretization  $\mathbf{f}$  has been chosen such that the forward Euler method  $\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t \mathbf{f}(\mathbf{u}^n)$  is strong stability preserving (SSP)

$$\|\mathbf{u}^{n+1}\| \leq \|\mathbf{u}^n\|, \quad (4.2)$$

in some norm, semi-norm or convex functional  $\|\cdot\|$ , under the restricted time step

$$\Delta t \leq \Delta t_{\text{FE}}.$$

One particularly important choice for  $\|\cdot\|$  is the *total variation semi-norm*

$$\|\mathbf{u}\|_{\text{TV}} = \sum_j |u_{j+1} - u_j|,$$

and a spatial discretization  $\mathbf{f}$  is said to be *total variation diminishing* (TVD) if forward Euler applied to (4.1) is SSP in this semi-norm.

Recall from Section 1.2.1 that a general explicit  $s$ -stage Runge–Kutta method for (4.1) can be written in Shu–Osher form [SO88]

$$\begin{aligned} \bar{\mathbf{u}}^{(0)} &= \mathbf{u}^n, \\ \bar{\mathbf{u}}^{(i)} &= \sum_{k=0}^{i-1} \left( \alpha_{ik} \bar{\mathbf{u}}^{(k)} + \Delta t \beta_{ik} \mathbf{f}(\bar{\mathbf{u}}^{(k)}) \right), \quad i = 1, \dots, s, \\ \mathbf{u}^{n+1} &= \bar{\mathbf{u}}^{(s)}. \end{aligned} \quad (4.3)$$

Consistency requires that  $\sum_{k=0}^{i-1} \alpha_{ik} = 1$ , and if  $\alpha_{ik} \geq 0$  and  $\beta_{ik} \geq 0$ , all the intermediate stages  $\bar{\mathbf{u}}^{(i)}$  in (4.3) are simply convex combinations of forward Euler operators, with  $\Delta t$  replaced by  $\frac{\beta_{ik}}{\alpha_{ik}} \Delta t$ . Therefore—as originally shown in [SO88]—any norm, semi-norm or convex function property (4.2) satisfied by the forward Euler method will be preserved by the Runge–Kutta method, under the time-step restriction

$$\Delta t \leq \min_{i < k} \frac{\alpha_{ik}}{\beta_{ik}} \Delta t_{\text{FE}}, \quad (4.4)$$

where we assume  $\frac{\alpha_{ik}}{\beta_{ik}} = \infty$  if  $\beta_{ik} = 0$ .

Much of the research in the field of SSP methods centers around the search for high-order SSP methods where the allowable time step is as large as possible. If a method has a SSP time-step restriction  $\Delta t \leq \mathcal{C} \Delta t_{\text{FE}}$ , then  $\mathcal{C}$ , the *SSP coefficient*, is often used to measure the allowable time step of a method in multiples of that of forward Euler. When the time step is limited by stability (rather than accuracy), then the *effective SSP coefficient*  $\mathcal{C}_{\text{eff}} = \frac{\mathcal{C}}{s}$  (that is, the SSP coefficient scaled by the number of stages), may be used to enable the fair comparison of explicit SSP schemes. Many optimal methods with the largest possible SSP coefficients for a given order and number of stages are listed in [RS02, SR03, Got05]. Three popular methods, SSP(2,2), SSP(3,3) and SSP(5,4), are given in Appendix C.

### 4.1.2 Implicit strong stability preserving time stepping

Historically, TVD spatial discretizations were constructed in conjunction with the forward Euler method for which the step size is restricted by  $\Delta t_{\text{FE}}$ . In contrast, the implicit backward Euler method will then preserve this TVD property for all time-step sizes [HRS03, Hig04]. However, a higher-order time discretization, such as the second-order Crank–Nicolson method, may only preserve the TVD property for a limited range of step sizes. For example, consider the case of the linear wave equation

$$u_t + au_x = 0,$$

with  $a = -2\pi$ , a step-function initial condition

$$u(x, 0) = \begin{cases} 1 & \text{if } \frac{\pi}{2} \leq x \leq \frac{3\pi}{2}, \\ 0 & \text{otherwise,} \end{cases}$$

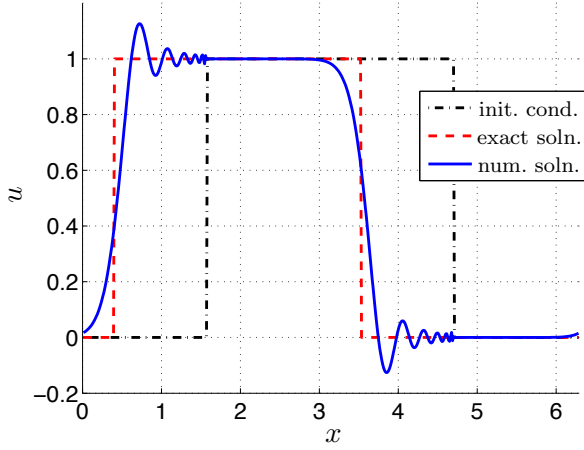


Figure 4.1: Oscillations from Crank–Nicolson time stepping in the advection of a square wave with  $\Delta t = 8\Delta t_{\text{FE}} = 8\Delta x$  and  $\Delta x = \frac{2\pi}{512}$ .

and periodic boundary conditions on the domain  $(0, 2\pi]$ . The solution is a step function convected around the domain. For a simple first-order forward-difference TVD spatial discretization  $\mathbf{f}(\mathbf{u})$  of  $-au_x$ , the result will be TVD for all sizes of  $\Delta t$  when using the implicit backward Euler method. Using forward Euler time stepping, the result is TVD for  $\Delta t \leq \Delta t_{\text{FE}} = \frac{\Delta x}{|a|}$ . On the other hand, consider the Crank–Nicolson method

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \frac{1}{2}\Delta t \mathbf{f}(\mathbf{u}^n) + \frac{1}{2}\Delta t \mathbf{f}(\mathbf{u}^{n+1}). \quad (4.5)$$

Using the Shu–Osher theory, Crank–Nicolson can be shown in general to be SSP only for values  $\Delta t \leq 2\Delta t_{\text{FE}}$  [Got05]. This restriction is illustrated for this problem in Figure 4.1 where an excessively large  $\Delta t$  leads to oscillations and a clear violation of the TVD property.

Crank–Nicolson requires extra computational cost due to the solution of an implicit system, but with respect to strong stability only allows a doubling of the step size compared to forward Euler or the second-order SSP(2,2). This means that, again with respect to strong stability, it will not be efficient to use this method.

The Shu–Osher form (1.9) has been generalized for implicit Runge–Kutta methods [Got05, FS05, Hig05], and the search for implicit methods which are SSP without a time-step restriction has generated much interest. The first-order backward Euler method is one such method. Unfortunately, there are no Runge–Kutta or linear multistep methods of order greater than one which will satisfy this property [Spi83, GST01, HR06]. The search for higher-order implicit SSP Runge–Kutta methods with finite but optimal SSP coefficients has been documented in [FS08, KMG08]. As discussed further in Section 4.2.1, strong stability and contractivity are closely related for the class of implicit Runge–Kutta methods.

Table 4.1: The 14 order conditions for fourth-order DSRK schemes written in matrix form where  $\mathbf{C} = \text{diag}(\mathbf{c})$ . See [BT97] for an explanation of the bi-coloured trees.

order 1	$\bullet \mathbf{b}^\top \mathbf{e} = 1$
order 2	$\bullet \mathbf{b}^\top \mathbf{C} \mathbf{e} = \frac{1}{2}$
order 3	$\mathbf{V} \mathbf{b}^\top \mathbf{C}^2 \mathbf{e} = \frac{1}{3}$ $\mathbf{b}^\top \mathbf{W} \mathbf{C} \mathbf{e} = \frac{1}{6}$ $\mathbf{b}^\top \mathbf{A} \mathbf{C} \mathbf{e} = \frac{1}{6}$
order 4	$\mathbf{V} \mathbf{b}^\top \mathbf{C}^3 \mathbf{e} = \frac{1}{4}$ $\mathbf{b}^\top \mathbf{C} \mathbf{W} \mathbf{C} \mathbf{e} = \frac{1}{8}$ $\mathbf{b}^\top \mathbf{C} \mathbf{A} \mathbf{C} \mathbf{e} = \frac{1}{8}$ $\mathbf{b}^\top \mathbf{W} \mathbf{C}^2 \mathbf{e} = \frac{1}{12}$ $\mathbf{b}^\top \mathbf{A} \mathbf{C}^2 \mathbf{e} = \frac{1}{12}$ $\mathbf{b}^\top \mathbf{W}^2 \mathbf{C} \mathbf{e} = \frac{1}{24}$ $\mathbf{b}^\top \mathbf{A} \mathbf{W} \mathbf{C} \mathbf{e} = \frac{1}{24}$ $\mathbf{b}^\top \mathbf{W} \mathbf{A} \mathbf{C} \mathbf{e} = \frac{1}{24}$ $\mathbf{b}^\top \mathbf{A}^2 \mathbf{C} \mathbf{e} = \frac{1}{24}$

This motivates us to search outside the class of Runge–Kutta methods for methods which are unconditionally contractive and high-order in the hope that they have good SSP properties as well. One class of high-order unconditionally contractive methods is the family of diagonally split Runge–Kutta (DSRK) methods.

## 4.2 Diagonally Split Runge–Kutta Methods

Diagonally split Runge–Kutta (DSRK) methods [BJZ94, BT97, itH96, Hor98] are one-step methods which are based on a Runge–Kutta formulation, but where the ODE operator  $\mathbf{f}$  in (4.1) has different inputs used for the diagonal and off-diagonal components. We define the *diagonal splitting function* of  $\mathbf{f}$  as

$$\mathfrak{F}_j(\mathbf{u}, \mathbf{z}) = \mathbf{f}(z_1, z_2, \dots, z_{j-1}, u_j, z_{j+1}, \dots, z_m), \quad j = 1, \dots, m, \quad (4.6)$$

that is, the  $j^{\text{th}}$  component of  $\mathfrak{F}(\mathbf{u}, \mathbf{z})$  is computed using the  $j^{\text{th}}$  component of  $\mathbf{u}$  for the  $j^{\text{th}}$  input of  $\mathbf{f}$  and components of  $\mathbf{z}$  for the other inputs of  $\mathbf{f}$ .

The general  $s$ -stage DSRK method is

$$\mathbf{U}^i = \mathbf{u}^n + \Delta t \sum_{j=1}^s a_{ij} \mathfrak{F}(\mathbf{U}^j, \mathbf{Z}^j), \quad i = 1, \dots, s \quad (4.7a)$$

$$\mathbf{Z}^i = \mathbf{u}^n + \Delta t \sum_{j=1}^s w_{ij} \mathfrak{F}(\mathbf{U}^j, \mathbf{Z}^j), \quad i = 1, \dots, s \quad (4.7b)$$

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t \sum_{j=1}^s b_j \mathfrak{F}(\mathbf{U}^j, \mathbf{Z}^j). \quad (4.7c)$$

The schemes are consistent [BJZ94] and the coefficients  $(\mathbf{A}, \mathbf{b}^\top, \mathbf{c}, \mathbf{W})$  must satisfy the order conditions [BT97] in Table 4.1. Note that these include the order conditions of the so-called *underlying Runge–Kutta method* (i.e., conditions only on  $\mathbf{A} = (a_{ij})$ ,  $\mathbf{b}$ , and  $\mathbf{c}$ ) and are augmented by additional order conditions on the coefficients  $\mathbf{W} = (w_{ij})$ . Finally, note that if  $\mathbf{W} = \mathbf{A}$ , then  $\mathbf{U}^i = \mathbf{Z}^i$  and the DSRK scheme reduces to the underlying Runge–Kutta method.

#### 4.2.1 Dissipative systems and contractivity

Bellen et al. [BJZ94] introduced the class of DSRK methods for *dissipative systems*  $\mathbf{u}_t = \mathbf{f}(t, \mathbf{u})$ . In the special case of the maximum norm  $\|\cdot\|_\infty$ , a dissipative system is characterized (see, e.g., [BT97]) by the condition

$$\sum_{j=1, j \neq i}^m \left| \frac{\partial \mathbf{f}_i(t, \mathbf{u})}{\partial u_j} \right| \leq -\frac{\partial \mathbf{f}_i(t, \mathbf{u})}{\partial u_i}, \quad i = 1, \dots, m,$$

for all  $t \leq t_0$  and  $\mathbf{u} \in \mathbb{R}^m$ . Note in particular that the ODEs resulting from the spatial discretizations of the linear PDE test problems in Sections 4.3.1, 4.3.2 and 4.3.3 satisfy this condition. The ODE system resulting from the nonlinear problem in Section 4.3.4 can be shown to be dissipative in  $\|\cdot\|_1$ .

If the ODE system is dissipative, then solutions satisfy a *contractivity* property [Spi83, Kra91, Zen93]. Specifically, if  $\mathbf{u}(t)$  and  $\mathbf{v}(t)$  are two solutions corresponding to initial conditions  $\mathbf{u}(t_0)$  and  $\mathbf{v}(t_0)$  then

$$\|\mathbf{u}(t) - \mathbf{v}(t)\| \leq \|\mathbf{u}(t_0) - \mathbf{v}(t_0)\|,$$

in some norm of interest. Naturally, if solutions to the ODE system obey a contractivity property then it is desirable that a numerical method for solving the problem be contractive as well, i.e., that given numerical solutions  $\mathbf{u}_n$  and  $\tilde{\mathbf{u}}_n$ ,  $\|\tilde{\mathbf{u}}_{n+1} - \mathbf{u}_{n+1}\| \leq \|\tilde{\mathbf{u}}_n - \mathbf{u}_n\|$  (possibly subject to a time-step restriction).

In [itH96], in 't Hout showed that if a DSRK method is unconditionally contractive in the maximum norm, the underlying Runge–Kutta method is of classical order  $p \leq 4$ , and has stage order  $\tilde{p} \leq 1$ . In [Hor98], Horváth studied the positivity of Runge–Kutta and DSRK methods, and showed that DSRK schemes can be unconditionally positive, that is preserve positivity properties for any step size.

The results on DSRK methods in terms of positivity and contractivity appear promising when searching for implicit SSP schemes, because positivity, contractivity, and the SSP condition are all very closely related for Runge–Kutta and multistep methods [Hig04, Hig05, FS04, FS05, Kra91]. For example, a loss of positivity implies the loss of the max-norm SSP property. For Runge–Kutta methods a link has also been established between time-step restrictions under the SSP condition and contractivity, namely that the time-step restrictions under either property agree [FS04], thereby enabling the possibility of transferring results established for the contractive case to the SSP case [Hig04], and vice versa. For multistep methods, the time-step restrictions coming from either an SSP or contractivity analysis are the same, as can be seen by examining the proofs appearing in [Len91, Len89, Shu88]. If we include the starting procedure into the analysis, or if we consider boundedness (a related nonlinear stability property) rather than the SSP property, significantly milder time-step restrictions may arise [HRS03]. However, even with this less restrictive boundedness property, we find that unconditional strong stability is impossible for multistep schemes that are more than first order [HR06]. The promise of DSRK methods is that there exist higher order implicit unconditionally contractive methods, and therefore possibly DSRK methods which are unconditionally SSP, in this class.

### 4.2.2 DSRK schemes

It is illustrative to examine (4.7) when the ODE operator is linear, that is  $\mathbf{f}(\mathbf{u}) = \mathbf{L}\mathbf{u}$ . In this case, with matrix  $\mathbf{L}$  decomposed into  $\mathbf{L} = \mathbf{L}_D + \mathbf{L}_N$  where  $\mathbf{L}_D = \text{diag}(\mathbf{L})$ , we have  $\mathfrak{F}(\mathbf{u}, \mathbf{z}) = \mathbf{L}_D\mathbf{u} + \mathbf{L}_N\mathbf{z}$  and (4.7) becomes

$$\mathbf{U}^i = \mathbf{u}^n + \Delta t \sum_{j=1}^s a_{ij} (\mathbf{L}_D \mathbf{U}^j + \mathbf{L}_N \mathbf{Z}^j), \quad i = 1, \dots, s \quad (4.8a)$$

$$\mathbf{Z}^i = \mathbf{u}^n + \Delta t \sum_{j=1}^s w_{ij} (\mathbf{L}_D \mathbf{U}^j + \mathbf{L}_N \mathbf{Z}^j), \quad i = 1, \dots, s \quad (4.8b)$$

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t \sum_{j=1}^s b_j (\mathbf{L}_D \mathbf{U}^j + \mathbf{L}_N \mathbf{Z}^j), \quad (4.8c)$$

and thus we see that for a linear ODE system, DSRK methods decompose the system into diagonal and off-diagonal components, and treat each differently.

The DSRK schemes listed next are used in Section 4.3 for the numerical tests.



**Second-order DSRK (“DSRK2”)** This second-order DSRK from [BJZ94] is based on the underlying two-stage, second-order implicit Runge–Kutta method specified by the Butcher tableau

$$\begin{array}{c|c} c & A \\ \hline & b^\top \end{array} = \begin{array}{c|cc} 0 & \frac{1}{2} & -\frac{1}{2} \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}, \quad \text{combined with } W = \begin{bmatrix} 0 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}. \quad (4.9a)$$

Thus the DSRK2 scheme is

$$U^1 = u^n + \frac{1}{2}\Delta t \mathfrak{F}(U^1, u^n) - \frac{1}{2}\Delta t f(u^{n+1}), \quad (4.9b)$$

$$u^{n+1} = u^n + \frac{1}{2}\Delta t \mathfrak{F}(U^1, u^n) + \frac{1}{2}\Delta t f(u^{n+1}). \quad (4.9c)$$

Note that the  $u^{n+1}$  terms are not split. For linear problems, (4.9) becomes

$$U^1 = u^n + \frac{1}{2}\Delta t [\mathsf{L}_D U^1 + \mathsf{L}_N u^n] - \frac{1}{2}\Delta t [\mathsf{L} u^{n+1}], \quad (4.10a)$$

$$u^{n+1} = u^n + \frac{1}{2}\Delta t [\mathsf{L}_D U^1 + \mathsf{L}_N u^n] + \frac{1}{2}\Delta t [\mathsf{L} u^{n+1}]. \quad (4.10b)$$

Note also in the special case when  $\mathsf{L}_D = 0$ , (4.10) decouples and (4.10b) is exactly the Crank–Nicolson method.

**Third-order DSRK (“DSRK3”)** This formally third-order DSRK scheme [BJZ94, BT97, itH96] is based on the underlying Runge–Kutta method:

$$\begin{array}{c|c} c & A \\ \hline & b^\top \end{array} = \begin{array}{c|ccc} 0 & \frac{5}{2} & -2 & -\frac{1}{2} \\ \frac{1}{2} & -1 & 2 & -\frac{1}{2} \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}, \quad \text{combined with } W = \begin{bmatrix} 0 & 0 & 0 \\ \frac{7}{24} & \frac{1}{6} & \frac{1}{24} \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{bmatrix}.$$

**Higher order DSRK schemes** Although unconditionally contractive second- and third-order DSRK methods such as DSRK2 and DSRK3 exist, so far no unconditionally contractive fourth-order DSRK methods have been found. In [itH96, proof of Theorem 2.4], the following necessary conditions are given for DSRK schemes to be unconditionally contractive in the maximum norm:

$$\text{all principal minors of } A - \mathbf{e}b^\top \text{ are nonnegative}, \quad (4.11a)$$

$$\text{for each } i \in \{1, 2, \dots, s\}, \det[(A \leftarrow_i b^\top)(\mathcal{I})] \geq 0 \quad (4.11b)$$

$$\text{for every } \mathcal{I} \subset \{1, 2, \dots, s\} \text{ with } i \in \mathcal{I},$$

where the notation “ $\leftarrow_i$ ” indicates replacing the  $i^{\text{th}}$  row. The notation  $\mathbf{M}(\mathcal{I})$  indicates the principal submatrix formed by selecting from  $\mathbf{M}$  only those rows and columns indexed by  $\mathcal{I}$ . These conditions are simpler than full necessary and sufficient conditions (e.g., [BT97, Theorem 3.3]) in part because they do not involve matrix inverses. This ensures the conditions can be written out as polynomial expressions which is ideal for the optimization software discussed next.

As a first step towards finding an unconditionally contractive four-stage fourth-order DSRK (DSRK44) scheme, we employ the proprietary Branch and Reduce Optimization Navigator (BARON) software [ST04] to search for DSRK44 satisfying conditions (4.11). In [Mac03, Ruu06, KMG08] the proprietary Branch and Reduce Optimization Navigator (BARON) software [ST04] was used to find optimal SSP Runge–Kutta schemes. Here we begin by searching for *any* feasible DSRK methods by imposing the 14 order conditions in Table 4.1 and the 48 necessary conditions (4.11) as constraints and minimizing the sum of the squares of the  $\mathbf{b}$  coefficients. BARON was interrupted after 30 days of calculation (on an Athlon MP 2800+ with 1 GiB of RAM) and was unable to find any feasible solutions. Constrained only by the order conditions, BARON was able to quickly find DSRK44 schemes; it was also able to quickly find five-stage fourth-order DSRK54 methods satisfying the order conditions and necessary conditions (4.11).

Altogether, this is a strong indication that unconditionally contractive DSRK44 methods do not exist. The question of the existence of unconditionally contractive DSRK54 schemes is left open, noting however that such schemes are still likely to suffer from the order reduction noted in Section 4.3.

### 4.2.3 Numerical implementation of DSRK

For linear problems, DSRK can be implemented using (4.8) by re-arranging all the unknowns into a larger linear system, in general  $(2sm) \times (2sm)$  where  $m$  is the size of the linear system (4.1) and  $s$  is the number of stages in the underlying Runge–Kutta scheme. However, particular choices of methods may result in smaller systems; for example, the two-stage DSRK2 (4.10) can be written as the  $2m \times 2m$  system

$$\left[ \begin{array}{c|c} 1 - \frac{1}{2}\Delta t L_D & \frac{1}{2}\Delta t L \\ \hline -\frac{1}{2}\Delta t L_D & 1 - \frac{1}{2}\Delta t L \end{array} \right] \begin{pmatrix} \mathbf{U}^1 \\ \mathbf{u}^{n+1} \end{pmatrix} = \begin{pmatrix} \mathbf{u}^n + \frac{1}{2}\Delta t L_N \mathbf{u}^n \\ \mathbf{u}^n + \frac{1}{2}\Delta t L_N \mathbf{u}^n \end{pmatrix},$$

where  $\mathbf{I}$  represents the  $m \times m$  identity. This linear system can then be solved to advance one time step. As is usually the case, nonlinear systems are considerably more difficult. For the non-linear problems, a numerical zero-finding method is used to solve the nonlinear equations.

All numerical computations are performed with Matlab versions 7.0 and 7.3 using double precision on x86 and x86-64 architectures. Linear systems are solved using MATLAB's `backslash` operator, whereas for the nonlinear problems in Sections 4.3.4 and 4.4.1, the diagonal splitting function (4.6) is implemented, and a black-box equation solver (MATLAB's `fsolve`) used directly on (4.7).

### 4.3 Numerical Results

The primary aim is to show that unconditionally contractive DSRK methods preserve the desired strong stability properties when applied to a variety of test cases. The numerical experiments are focused on three types of problems: convection, diffusion, and convection-diffusion. The SSP property is perhaps most important for convection driven problems, such as hyperbolic problems with discontinuous solutions. The methods have also been used to treat problems where the slope or some derivative of the solution is discontinuous and, for this reason, SSP schemes have been used widely to treat Hamilton–Jacobi equations (see, e.g., [OF03]). Many other problems of reaction-advection-diffusion type also can benefit from nonlinearly stable time stepping. For example, time stepping a spatially discretized Black–Scholes equation (an equation we consider in Section 4.3.3) can lead to spurious oscillations in the solution. These oscillations are particularly undesirable in option-pricing problems because they can lead to highly oscillatory results in the first and second spatial derivatives—known respectively as  $\gamma$  and  $\delta$  (“the Greeks”) in computational finance.

In the section, the time-step size  $\Delta t$  will often be measured relative to the forward Euler restriction  $\Delta t_{\text{FE}}$  using  $c$  as  $\Delta t = c\Delta t_{\text{FE}}$ .

#### 4.3.1 Convection-driven problems

An important prototype problem for SSP methods is the linear wave equation, or *advection equation*

$$u_t + au_x = 0, \quad 0 \leq x \leq 2\pi. \quad (4.12)$$

We consider (4.12) with  $a = -2\pi$ , periodic boundary conditions, and various initial conditions. A method-of-lines approach is used, discretizing the interval  $(0, 2\pi]$  into  $m$  points  $x_j = j\Delta x$ ,  $j = 1, \dots, m$ , and then discretizing  $-au_x$  with first-order upwind finite differences. The resulting linear system is solved using the time-stepping schemes described in Sections 4.1 and 4.2.

### Smooth initial conditions

To study the order of accuracy of the methods, we consider (4.12) with smooth initial conditions

$$u(0, x) = \sin(x).$$

Table 4.2 shows a numerical convergence study with fixed  $\Delta x$ . The implicit time-discretization methods used are backward Euler (BE), Crank–Nicolson (CN), DSRK2 and DSRK3. The system is also evolved with several explicit methods: forward Euler (FE), SSP(2,2), SSP(3,3) and SSP(5,4). To isolate the effect of the time-discretization error, the effect of the error associated with the spatial discretization is excluded by comparing the numerical solution to the exact solution of the ODE system (4.1), rather than to the exact solution of the underlying PDE. In lieu of the exact solution a very accurate numerical solution is obtained using MATLAB's `ode45` with minimal tolerances ( $\text{AbsTol} = 1 \times 10^{-14}$ ,  $\text{RelTol} = 1 \times 10^{-13}$ ). Table 4.2 shows that all the methods achieve their design order when  $\Delta t$  is sufficiently small. However, the errors from CN are typically smaller than the errors produced by the other implicit methods. For large  $\Delta t$ , the second- and third-order DSRK schemes are far worse than CN. If we broaden the experiments to include explicit schemes and take time steps which are within the stability time-step restriction, we obtain smaller errors still. Given the relatively inexpensive cost of explicit time stepping, it would appear that high-order explicit schemes (e.g., SSP(5,4)) are likely more efficient for this smooth problem, unless, perhaps, very large time steps are preferred over accuracy considerations.

### Discontinuous initial conditions

To study the nonlinear stability properties of the methods, we consider the case of advection of discontinuous data

$$u(x, 0) = \begin{cases} 1 & \text{if } \frac{\pi}{2} \leq x \leq \frac{3\pi}{2}, \\ 0 & \text{otherwise.} \end{cases} \quad (4.13)$$

Table 4.2: Numerical convergence study for the linear advection of a sine wave to  $t_f = 1$  using  $N$  time steps,  $m = 64$  points and a first-order upwinding spatial discretization. Here  $c$  measures the size of the time step relative to  $\Delta t_{FE}$ .

$c$	$N$	discrete error, $l_\infty$ -norm							
		BE	order	CN	order	DSRK2	order	DSRK3	order
4	16	0.518		0.0582		0.408		0.395	
2	32	0.336	0.62	0.0147	1.98	0.194	1.08	0.178	1.15
1	64	0.194	0.79	3.70e-3	2.00	0.0714	1.44	0.0590	1.59
$\frac{1}{2}$	128	0.105	0.89	9.25e-4	2.00	0.0223	1.68	0.0152	1.95
...	...	...	...	...	...	...	...	...	...
$\frac{1}{32}$	2048	7.04e-3		3.61e-6		1.09e-4		1.21e-5	
$\frac{1}{64}$	4096	3.53e-3	1.00	9.04e-7	2.00	2.74e-5	1.99	1.61e-6	2.91
$\frac{1}{128}$	8192	1.77e-3	1.00	2.26e-7	2.00	6.87e-6	1.99	2.09e-7	2.95
$c$	$N$	FE	order	SSP22	order	SSP33	order	SSP54	order
2	32	unstable		unstable		unstable		2.66e-5	
1	64	0.265		7.43e-3		1.82e-4		1.66e-6	4.00
$\frac{1}{2}$	128	0.122	1.12	1.85e-3	2.01	2.27e-5	3.00	1.03e-7	4.01

Figure 4.2 shows typical results. Note that oscillations are observed in the Crank–Nicolson results, while the DSRK schemes are free of such oscillations. In fact, Table 4.3 shows that for any time-step size BE, DSRK2 and DSRK3 preserve the TVD property of the spatial discretization coupled with forward Euler. In contrast, Crank–Nicolson exhibits oscillations for time steps larger than  $\Delta t = \frac{2}{|a|} \Delta x$  (i.e.,  $c > 2$ ). These results suggest that the unconditionally contractive DSRK schemes do preserve the strong stability properties of the ODE system.

$c$	$N$	$\max_t TV(\mathbf{u})$				
		exact	CN	BE	DSRK2	DSRK3
32	16	2	8.78	2	2	2
16	32	2	6.64	2	2	2
8	64	2	4.73	2	2	2
4	128	2	3.33	2	2	2
2	256	2	2	2	2	2
1	512	2	2	2	2	2

Table 4.3: Total variation of the solution for the advection of a square wave ( $N$  time steps,  $t_f = 1$ ). The spatial discretization uses  $m = 512$  points, first-order upwinding and periodic BCs.

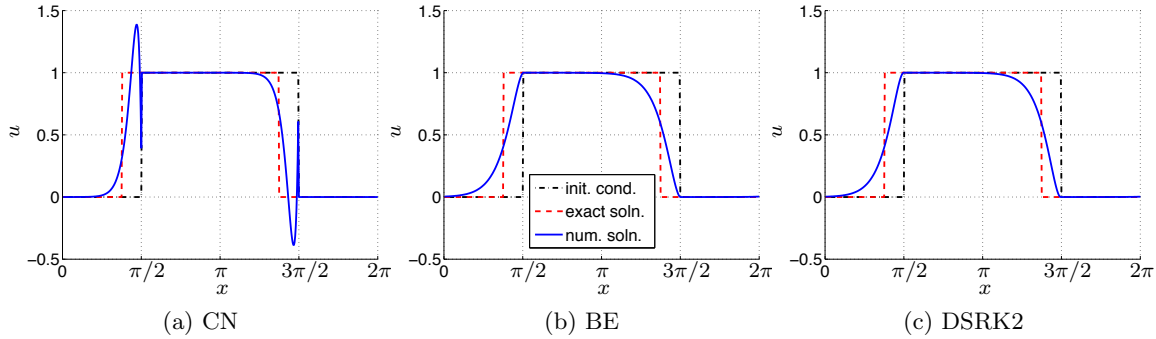


Figure 4.2: Advection of a square wave after two time steps, showing oscillations from Crank–Nicolson and none from backward Euler and DSRK2. Here  $c = 16$  and a first-order upwinding spatial discretization is used with  $m = 512$  points in space.

### Order reduction and scheme selection

We now delve deeper into the observed convergence rates of the smooth and nonsmooth problems. Figures 4.3 and 4.4 show that for large time steps, the DSRK methods exhibit behavior similar to backward Euler in that they exhibit large errors and as the size of the time steps are decreased, the error decreases at a rate which appears only first order. As the time steps are taken smaller still, the convergence rate increases to the design order of the DSRK schemes. In contrast, note that Crank–Nicolson shows consistent second-order convergence over a wide range of time steps.

On the discontinuous problem (Figure 4.4), note the DSRK schemes do not produce significantly improved errors over backward Euler until the time-step sizes are small enough that Crank–Nicolson no longer exhibits spurious oscillations ( $c = 2$  in Figure 4.4). In fact, once the time steps are small enough that DSRK are competitive, we are almost within the nonlinear stability constraint of explicit methods such as SSP(2,2) ( $c = 1$  in Figure 4.4).

Note that neither Figure 4.3 nor Figure 4.4 takes into account the differences in computational work required by the various methods. The costs for DSRK2 and DSRK3 are significantly larger than BE and CN, because the underlying systems are larger. In the linear case, the size of the DSRK2 system is  $2m \times 2m$  and the DSRK3 system is  $5m \times 5m$ , whereas the BE and CN systems are only  $m \times m$ . Even if the cost of solving the system rose only linearly with the size of the system, the cost is doubled for DSRK2 and increased five-fold for DSRK3. In reality, the cost may increase more rapidly, depending on the structure of the implicit system and the method used to solve the implicit equations. Furthermore,

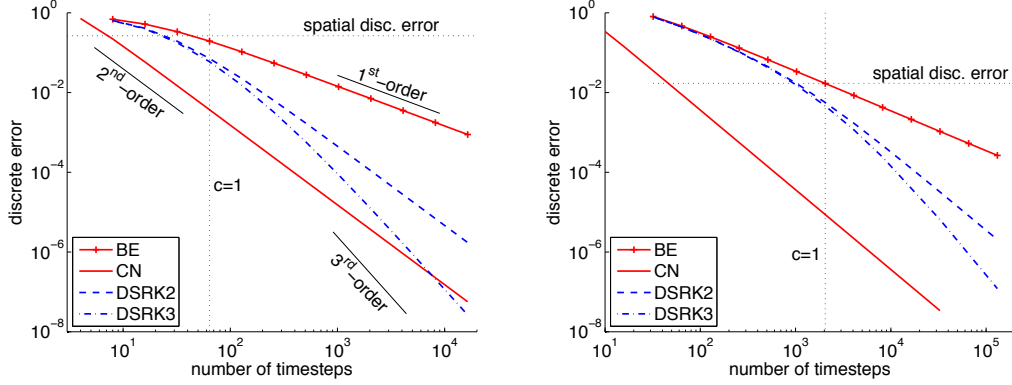


Figure 4.3: Numerical convergence study for linear advection of a sine wave to  $t_f = 1$ . The spatial discretization here is first-order upwinding with 64 points (left) and 2048 points (right). The spatial discretization error is determined experimentally and is indicated with a dotted horizontal line.

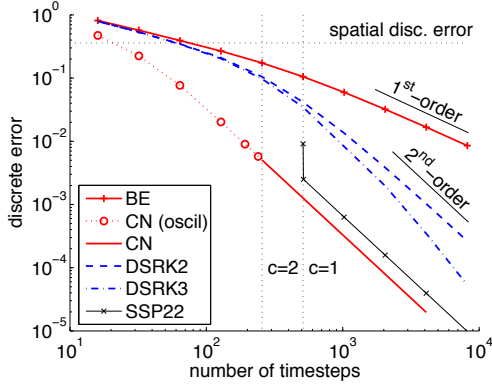


Figure 4.4: Numerical convergence study for linear advection of a square wave to time  $t_f = 1$  using first-order upwinding and 512 points in space. Note that Crank–Nicolson produces oscillations during the computation for  $c > 2$ . The spatial discretization error is indicated with a dotted horizontal line.

if a nonlinear system is solved, this cost may increase even further. It is even more difficult to quantify the increased cost of an implicit method over that of an explicit method. However, it is clear that implicit methods in general, and DSRK methods in particular, are significantly more costly than explicit methods.

Note that phase errors were also investigated to see if the DSRK schemes had improved phase error properties compared to BE, but they do not. In general, for large  $\Delta t$ , DSRK methods behave similarly in many aspects to backward Euler.

In summary, the results on the advection equation show that although the unconditionally contractive DSRK methods are formally high order, in practice a reduction of order is encountered for large time steps. If one requires large time steps, low to moderate accuracy,

and no oscillations, backward Euler is a good choice. If on the other hand, one requires higher accuracy, an explicit high-order SSP method is probably better suited. We will see that these results are typical for unconditionally contractive DSRK schemes. The order reduction is analyzed further in Section 4.4.

### 4.3.2 Diffusion driven problems

Consider the diffusion or heat equation

$$u_t = \nu u_{xx}, \quad (4.14)$$

with heat coefficient  $\nu$  on a periodic domain  $(0, 2\pi]$ . We begin by discretizing the  $u_{xx}$  term with second-order centered finite differences to obtain the ODE system (4.1).

In Figure 4.5 and Table 4.4, we consider (4.14) with smooth initial conditions

$$u(0, x) = \sin(x) + \cos(2x).$$

Once again, note that the DSRK schemes achieve their design order as  $\Delta t$  gets smaller, but for large time steps they exhibit large errors and reduced convergence rates.

Figure 4.6 shows that Crank–Nicolson produces spurious oscillations in the solution to the heat equation with discontinuous initial conditions (4.13). Also, Figure 4.6 shows that the DSRK schemes are not competitive with backward Euler until the time steps are smaller than the explicit stability limit (in this case, the restrictive  $\Delta t \leq \frac{\Delta x^2}{2\nu}$  shown by the dotted vertical line). Clearly, the unconditionally contractive DSRK methods exhibit order reduction for this parabolic problem as well.

### 4.3.3 The Black–Scholes equation

The Black–Scholes equation [BS73]

$$V_\tau = \frac{\sigma}{2} S^2 V_{SS} + r S V_S - r V, \quad (4.15)$$

is a PDE used in computational finance [For05] for determining the fair price  $V$  of an option at stock price  $S$ , where  $\sigma$  is the volatility and  $r$  is the risk-free interest rate. Note  $S$  is the independent (we can think “spatial”) variable on the positive half-line and  $\tau$  is a rescaled time (the actual time runs backwards from “final conditions”). The initial conditions, shown



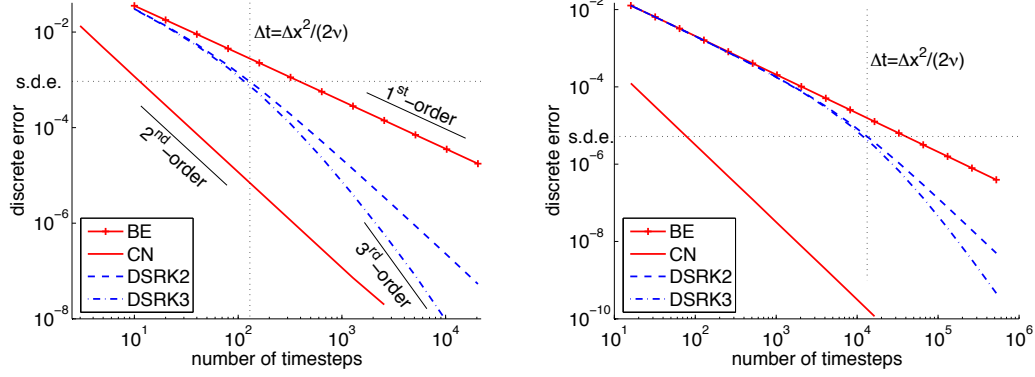


Figure 4.5: Numerical convergence studies for the heat equation with smooth initial conditions. Left:  $m = 64$ ,  $t_f = 10$ ,  $\nu = \frac{1}{16}$ . Right:  $m = 1024$ ,  $t_f = 1$ ,  $\nu = \frac{1}{4}$ . The spatial discretization uses second-order centered differences and the level of spatial discretization error is indicated by the horizontal dotted line labeled “s.d.e.”

$c$	$N$	discrete error $l_\infty$ -norm			
		BE	CN	DSRK2	DSRK3
830	16	0.0127	1.24e-4	0.0127	0.0127
415	32	0.00643	3.09e-5	0.00640	0.00640
...	...	...	...	...	...
12.97	1024	2.03e-4	3.02e-8	1.76e-4	1.74e-4
6.48	2048	1.02e-4	7.55e-9	7.77e-5	7.55e-5
1	13280	1.57e-5	1.80e-10	5.23e-6	4.28e-6
		FE	SSP22	SSP33	SSP54
2	6640	unstable	unstable	unstable	8.74e-13
1	13280	1.57e-5	3.59e-10	4.42e-13	1.32e-12

Table 4.4: Numerical convergence study for the heat equation with smooth initial conditions. Here  $\nu = 1/4$ ,  $m = 1024$ ,  $t_f = 1$ . The discrete error is computed against the ODE solution calculated with MATLAB’s `ode15s`. For comparison explicit methods are shown near their stability limits around  $c = 1$ .

Figure 4.6: Numerical convergence study for heat equation with discontinuous initial conditions using  $m = 512$ ,  $t_f = 1$  and  $\nu = \frac{1}{4}$ . The spatial discretization in this example is second-order centered differences.

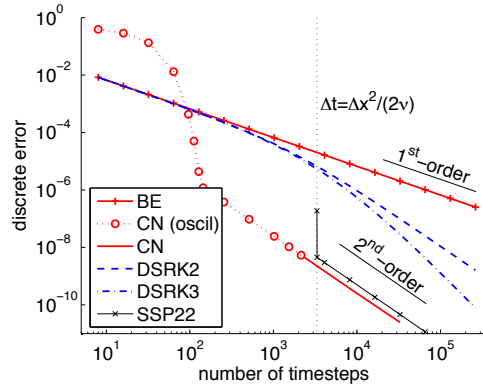
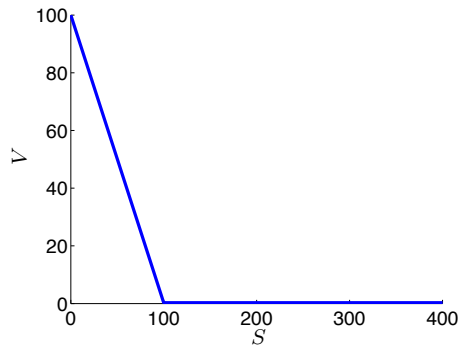


Figure 4.7: Computational domain and initial conditions for the Black-Scholes problem.



in Figure 4.7, have a discontinuity in the first derivative at  $S = 100$  (these initial conditions are known as a “put option” with a “strike price” of  $S = 100$ ).

Note that (4.15) is a linear non-constant coefficient advection-reaction-diffusion equation and can be treated as the ODE system (4.1) by approximating the  $V_S$  term with first-order upwind finite differences and the  $V_{SS}$  term with second-order centered finite differences. We use  $\sigma = 0.8$ ,  $r = 0.1$ , and for this choice no significant difference between upwind and centered differences for the advection term was observed. The right-hand boundary condition is an approximation to  $\lim_{S \rightarrow \infty} V(S) = 0$ , specifically  $V(S_{\max}) = 0$ . At the left-hand end of the domain, note that (4.15) reduces to

$$\dot{V}_0 = -rV_0,$$

and thus it is both natural and convenient to simply solve this ODE coupled with the other components  $V_j$  as part of the method-of-lines computation.

Figure 4.8 shows the problem of oscillations which show up in a Crank–Nicolson calculation of the Black–Scholes problem. The oscillations are amplified in “the Greeks” i.e., the first and second spatial derivatives. Note this is a well-known phenomenon [Col06] associated with the CN numerical solution of (4.15); in practice, Rannacher time stepping consisting of several initial steps of BE followed by CN steps [GC06] is often used to avoid these oscillations. DSRK schemes also avoid oscillations but are not likely competitive with Rannacher time stepping in terms of efficiency due to the order reduction illustrated in Table 4.5 as well as the greater expense of implementing the DSRK schemes. A great number of time steps ( $N = 17,800$  in the case considered in Table 4.5) are required before the Crank–Nicolson calculation is completely oscillation-free in “the Greeks”.

Note that explicit methods are not practical for this problem because of the severe linear stability restriction imposed by the diffusion term in (4.15). If an oscillation-free calculation is desired, then backward Euler is preferred over DSRK methods because DSRK methods cost more and offer essentially the same first-order convergence rates for time-step sizes of practical interest. Moreover, DSRK schemes can offer little practical advantage over current Rannacher time-stepping techniques which attempt to combine the best aspects of backward Euler and Crank–Nicolson.

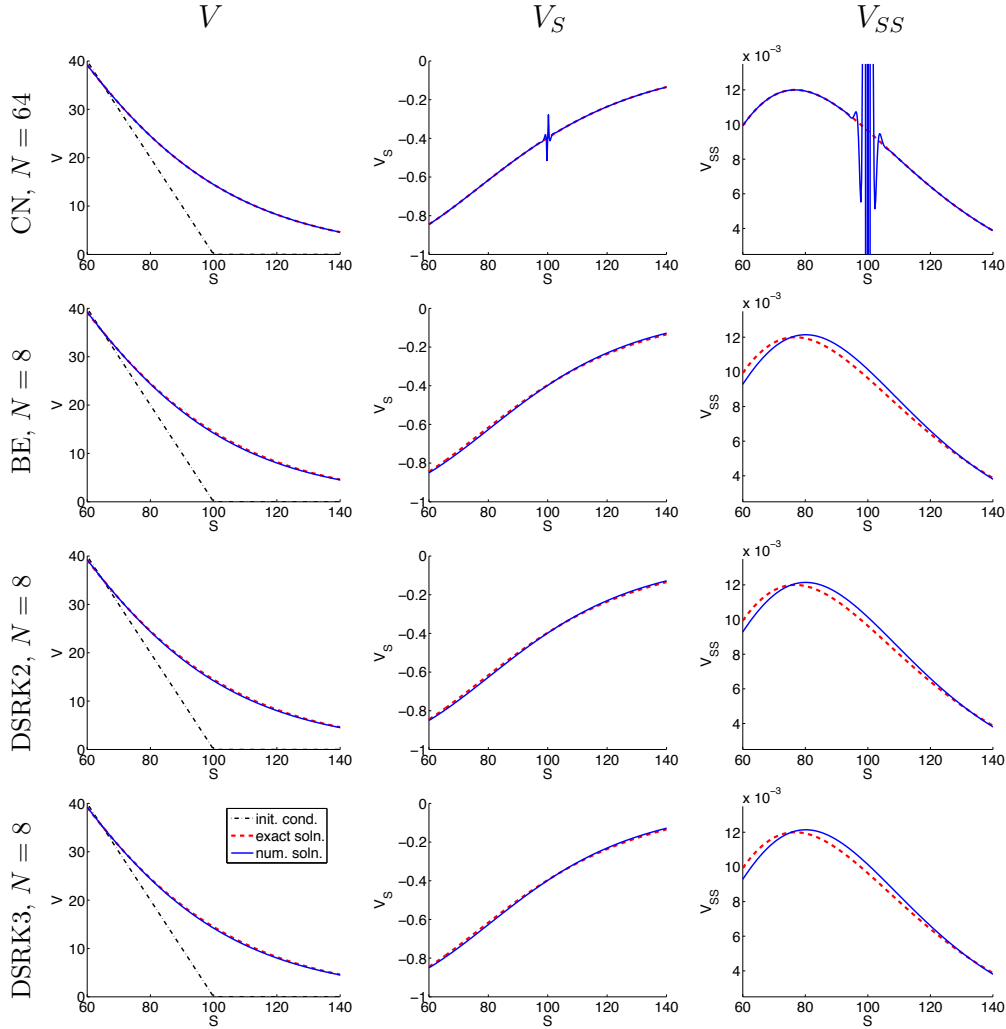


Figure 4.8: Numerical solutions of the Black–Scholes problem magnified near  $S = 100$  with  $m = 1600$ ,  $t_f = \frac{1}{4}$ ,  $\sigma = 0.8$ ,  $r = 0.1$  and  $S_{\max} = 400$  using  $N$  time steps. From left-to-right:  $V$ ,  $V_S$  and  $V_{SS}$ . Note that Crank–Nicolson exhibits oscillations with  $N = 64$  whereas BE and the DSRK schemes appear free of oscillation even with the larger time steps corresponding to  $N = 8$ .

Table 4.5: Black–Scholes numerical convergence study. \* indicates oscillations in  $V$ ,  $V_S$  or  $V_{SS}$ . Here,  $m = 1600$ ,  $S_{\max} = 400$ ,  $\Delta x = \frac{1}{4}$ ,  $t_f = \frac{1}{4}$ ,  $\sigma = 0.8$  and  $r = 0.1$ . The discrete error is calculated against a numerical solution from MATLAB’s `ode15s` with  $\text{AbsTol} = 1 \times 10^{-14}$ ,  $\text{RelTol} = 1 \times 10^{-13}$ .

$N$	discrete error $l_\infty$ -norm							
	BE	order	CN	order	DSRK2	order	DSRK3	order
32	0.0655		0.115 *		0.0654		0.0654	
64	0.0328	1.00	0.0452 *	1.35	0.0327	1.00	0.0326	1.00
128	0.0164	1.00	8.64e-3 *	2.39	0.0163	1.00	0.0163	1.00
256	8.21e-3	1.00	8.76e-5 *	6.62	8.07e-3	1.01	8.06e-3	1.02
512	4.10e-3	1.00	1.95e-6 *	5.49	3.97e-3	1.02	3.96e-3	1.03
1024	2.05e-3	1.00	4.88e-7 *	2.00	1.92e-3	1.05	1.91e-3	1.05
...	...		...		...		...	
8192	2.57e-4		7.62e-9 *		1.60e-4		1.51e-4	
16384	1.28e-4	1.00	1.90e-9 *	2.00	5.67e-5	1.50	4.98e-5	1.60
32768	6.41e-5	1.00	4.75e-10	2.00	1.78e-5	1.67	1.67e-5	1.58

#### 4.3.4 Hyperbolic conservation laws: Burgers’ equation

The examples so far have dealt exclusively with linear problems. In this section we consider Burgers’ equation

$$u_t = -f(u)_x = -\left(\frac{1}{2}u^2\right)_x,$$

with initial condition  $u(0, x) = \frac{1}{2} - \frac{1}{4}\sin(\pi x)$  on the periodic domain  $x \in [0, 2)$ . The solution is right-travelling and over time steepens into a shock. The right hand side  $-f(u)_x$  is discretized using a conservative simple upwind approximation

$$-f(u)_x \approx -\frac{1}{\Delta x} \left( \tilde{f}_{i+\frac{1}{2}} - \tilde{f}_{i-\frac{1}{2}} \right) = -\frac{1}{\Delta x} (f(u_i) - f(u_{i-1})).$$

Figure 4.9 shows that Crank–Nicolson produces spurious oscillations in the wake of the shock, for  $c = 8$  (in fact, oscillations can be observed from CN for  $c \geq 4$  as noted in Table 4.6). As expected, BE, DSRK2 and DSRK3 produce a non-oscillatory TVD solution. Table 4.6 shows a numerical convergence study for this problem which illustrates the familiar pattern of order reduction. Notice, in particular, that for any time-step size considered, one of BE or CN gives non-oscillatory results with similar or smaller errors than the DSRK schemes considered here. Furthermore, for small time steps, the explicit methods considerably outperform the other choices.

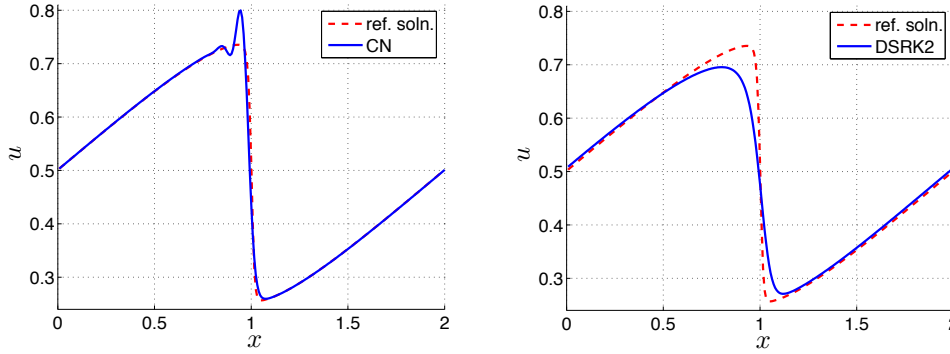


Figure 4.9: Burgers' equation with Crank–Nicolson (left) and DSRK2 (right) with  $m = 256$  spatial points and  $t_f = 2$ ,  $N = 32$  ( $c = 8$ ). For CN, the solution appears smooth until the shock develops; then an oscillation develops at the trailing edge of the shock. Note that DSRK2 appears overly dissipative. The reference solution is calculated with CN and  $N = 8192$ .

Table 4.6: Burgers' equation numerical convergence study. Values for which oscillations appear are indicated with \*. The setup here is the same as in Figure 4.9 except the reference solution is calculated with SSP(5,4) and  $N = 8192$ .

$c$	$N$	error ( $l_\infty$ -norm against ref. soln.)							
		BE	order	CN	order	DSRK2	order	DSRK3	order
16	16	0.192		0.193 *		0.195		0.195	
8	32	0.173	0.15	0.109 *	0.82	0.153	0.35	0.154	0.34
4	64	0.140	0.31	0.0399 *	1.45	0.110	0.47	0.114	0.43
2	128	0.0964	0.54	0.0124	1.68	0.0644	0.78	0.0673	0.76
1	256	0.0589	0.71	3.11e-3	2.00	0.0273	1.24	0.0249	1.43
0.5	512	0.0320	0.88	7.72e-4	2.01	8.72e-3	1.65	6.79e-3	1.87
0.25	1024	0.0165	0.96	1.90e-4	2.02	2.45e-3	1.83	1.39e-3	2.29
		FE	order	SSP22	order	SSP33	order	SSP54	order
4	64	unstable		unstable		unstable		unstable	
2	128	unstable		unstable		unstable		2.50e-4	
1	256	0.0880		5.98e-3		3.54e-4		1.36e-5	4.20
0.5	512	0.0377	1.22	1.45e-3	2.04	4.32e-5	3.03	7.63e-7	2.88
0.25	1024	0.0172	1.13	3.63e-4	2.00	5.34e-6	3.02	4.46e-8	4.10
0.125	2048	8.43e-3	1.03	9.08e-5	2.00	6.61e-7	3.01	2.68e-9	4.06

$\epsilon$	$y_1(0)$	$y_2(0)$
$1 \times 10^{-1}$	2	-0.65
$1 \times 10^{-2}$	2	-0.6654321
$1 \times 10^{-3}$	2	-0.66654321
$1 \times 10^{-4}$	2	-0.666654321
$1 \times 10^{-5}$	2	-0.6666654321
$1 \times 10^{-6}$	2	-0.66666654321
$1 \times 10^{-7}$	2	-0.666666654321

Table 4.7: Initial conditions for the van der Pol problem for various  $\epsilon$  (from [LM05, Table 5.1]).

## 4.4 Stage Order and Order Reduction

The numerical experiments have shown that the unconditionally contractive DSRK2 and DSRK3 methods preserve nonlinear stability properties when applied to our test cases in Section 4.3. However, the results also show that these methods suffer from order reduction: when large time steps  $\Delta t$  are taken, the form of the error appears larger than the design order term  $\mathcal{O}(\Delta t^p)$  would suggest, for example, because of large derivatives in stiff problems which may not be  $\mathcal{O}(1)$  when  $\Delta t$  is large [HW96]. The order reduction implies that the unconditionally contractive DSRK methods are not likely an appropriate choice for a time-stepping scheme, because they cannot compete with BE for large time steps or with SSP explicit methods for smaller time steps.

### 4.4.1 The van der Pol equation

To further investigate the order reduction observed in the previous numerical tests, we apply the DSRK methods to the van der Pol equation, a problem often used for testing for reduction of order (see, e.g., [LM05] and references therein). The problem can be written as an ODE initial value problem consisting of two components

$$y_1' = y_2, \tag{4.16a}$$

$$y_2' = \frac{1}{\epsilon} (-y_1 + (1 - y_1^2)y_2), \tag{4.16b}$$

with  $\epsilon$ -dependent initial conditions shown in Table 4.7. The problem becomes increasingly stiff as  $\epsilon$  is decreased. The solution is computed to  $t_f = \frac{1}{2}$ .

Figure 4.10 shows the distinctive “flattening” [LM05] that occurs during the numerical convergence studies whereby the error exhibits a region (depending on  $\epsilon$ ) of first-order

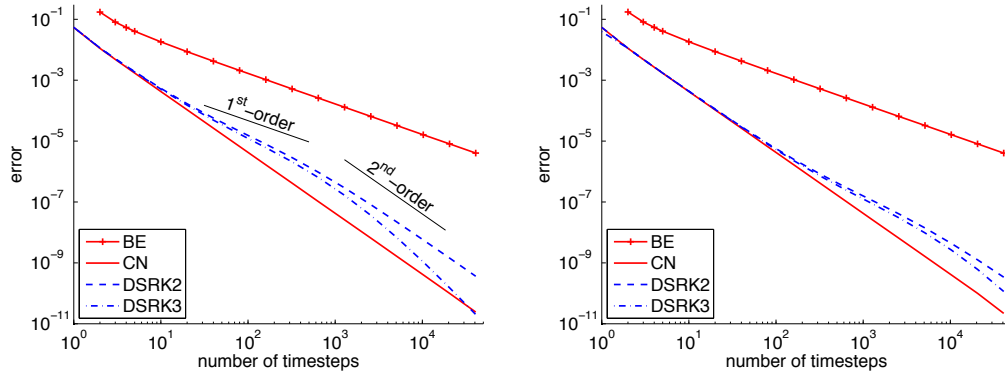


Figure 4.10: Numerical convergence study on the van der Pol equation for  $\epsilon = 1 \times 10^{-3}$  (left) and  $\epsilon = 1 \times 10^{-4}$  (right). Error shown is in the second component.

behaviour as the time-step size decreases before eventually approaching the design order of the method. This suggests that DSRK schemes suffer from order reduction whereas Crank–Nicolson does not. Before the flattened region, all the high-order methods produce similar errors. In particular DSRK3 does no better than the second-order Crank–Nicolson until after the flattening region. Note that this order reduction is noticeable despite the fact that the choices of  $\epsilon$  used do not correspond to particularly stiff systems.

#### 4.4.2 DSRK schemes with higher underlying stage order

The order reduction is not completely unexpected, as [itH96] showed that the underlying Runge–Kutta methods must have stage order at most one, and low stage order—at least in Runge–Kutta schemes—is known to lead to order reduction [HW96]. For comparison, consider a DSRK method which is based on the two-stage, second-order implicit Runge–Kutta method with stage order two

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} = \begin{array}{c|cc} \frac{1}{2} & \frac{3}{4} & -\frac{1}{4} \\ 1 & 1 & 0 \\ \hline & 1 & 0 \end{array}, \quad \text{combined with } W = \begin{bmatrix} \frac{1}{2} & 0 \\ 1 & 0 \end{bmatrix}. \quad (4.17)$$

We call this method DSRK2uso2. Because the underlying method has stage order larger than one (i.e., two), the DSRK2uso2 method cannot be unconditionally contractive [itH96].



#### 4.4.3 DSRK schemes with higher stage order

Figures 4.11 and 4.12 indicate that DSRK2uso2 also suffers from order reduction. Thus, it appears that higher stage order of the underlying Runge–Kutta scheme is not sufficient to avoid order reduction. Thus stage order properties of the DSRK scheme itself are investigated using the test problem of [PR74]

$$u' = \lambda(u - \phi(t)) + \phi'(t), \quad \text{Re}(\lambda) < 0, \quad u(t_0) = \phi(t_0).$$

However, because DSRK schemes reduce to Runge–Kutta schemes on scalar problems, we introduce a modified vector version

$$\mathbf{u}' = \mathbf{\Lambda}(\mathbf{u} - \phi(t)) + \phi'(t), \quad (4.18)$$

where  $\mathbf{u}(t_0) = \phi(t_0)$  and  $\mathbf{\Lambda}$  is negative semidefinite, where the exact solution is  $\mathbf{u}(t) = \phi(t)$ . Following the Runge–Kutta analysis in [HW96, Section IV.15], the general DSRK scheme (4.8) can be applied to the vector test problem to obtain

$$\begin{aligned} \mathbf{u}^{n+1} &= \mathbf{u}^n + \Delta t \sum_{j=1}^s b_j [\mathbf{\Lambda}_D(\mathbf{U}^j - \phi(t + c_j \Delta t)) + \mathbf{\Lambda}_N(\mathbf{Z}^j - \phi(t + c_j \Delta t)) + \phi'(t + c_j \Delta t)], \\ \mathbf{U}^i &= \mathbf{u}^n + \Delta t \sum_{j=1}^s a_{ij} [\mathbf{\Lambda}_D(\mathbf{U}^j - \phi(t + c_j \Delta t)) + \mathbf{\Lambda}_N(\mathbf{Z}^j - \phi(t + c_j \Delta t)) + \phi'(t + c_j \Delta t)], \\ \mathbf{Z}^i &= \mathbf{u}^n + \Delta t \sum_{j=1}^s w_{ij} [\mathbf{\Lambda}_D(\mathbf{U}^j - \phi(t + c_j \Delta t)) + \mathbf{\Lambda}_N(\mathbf{Z}^j - \phi(t + c_j \Delta t)) + \phi'(t + c_j \Delta t)]. \end{aligned}$$

Substituting the exact solution for  $\mathbf{u}^n$ ,  $\mathbf{u}^{n+1}$ ,  $\mathbf{U}^i$  and  $\mathbf{Z}^i$  results in

$$\begin{aligned} \phi(t_n + \Delta t) &= \phi(t_n) + \Delta t \sum_{j=1}^s b_j \phi'(t + c_j \Delta t) + \mathbf{d}_0, \\ \phi(t_n + c_i \Delta t) &= \phi(t_n) + \Delta t \sum_{j=1}^s a_{ij} \phi'(t + c_j \Delta t) + \mathbf{d}_1^i, \quad i = 1, \dots, s, \\ \phi(t_n + c_i \Delta t) &= \phi(t_n) + \Delta t \sum_{j=1}^s w_{ij} \phi'(t + c_j \Delta t) + \mathbf{d}_2^i, \quad i = 1, \dots, s, \end{aligned}$$

where  $\mathbf{d}_0$ ,  $\mathbf{d}_1^i$  and  $\mathbf{d}_2^i$  measure the resulting *defect* between the left- and right-hand sides. Taylor series expansions of the various  $\phi$  and  $\phi'$  terms about  $t_n$  can be used to determine the order of each defect,  $\mathbf{d}_0 = \mathcal{O}(\Delta t^{q_0+1})$ ,  $\mathbf{d}_1^i = \mathcal{O}(\Delta t^{q_1+1})$ , and  $\mathbf{d}_2^i = \mathcal{O}(\Delta t^{q_2+1})$ . The

values of  $q_0$ ,  $q_1$  and  $q_2$  depend on the coefficients of the DSRK scheme and are the largest values such the following relations hold:

$$\mathbf{b}^\top \mathbf{c}^{k-1} = \frac{1}{k}, \quad \text{for } k = 1, \dots, q_0, \quad (4.21a)$$

$$\mathbf{A} \mathbf{c}^{k-1} = \frac{\mathbf{c}^k}{k}, \quad \text{for } k = 1, \dots, q_1, \quad (4.21b)$$

$$\mathbf{W} \mathbf{c}^{k-1} = \frac{\mathbf{c}^k}{k}, \quad \text{for } k = 1, \dots, q_2, \quad (4.21c)$$

where the  $\mathbf{c}^k$  indicates component-wise exponentiation. We then define the *stage order* of the DSRK method as follows.

**Definition 4 (DSRK Stage Order)** *Consider a DSRK scheme with coefficients  $\mathbf{A}$ ,  $\mathbf{W}$ ,  $\mathbf{b}$  and  $\mathbf{c}$ . Let  $q_0$ ,  $q_1$  and  $q_2$  be the largest possible values such that the relations (4.21) hold. Then  $\min(q_0, q_1, q_2)$  is the stage order of the DSRK scheme.*

Note that  $\min(q_0, q_1)$  is the stage order of the underlying Runge–Kutta scheme [HW96, Section IV.15] and that  $q_0 \geq p$ , where  $p$  is the order of the DSRK scheme.

The scheme DSRK2uso2 has  $q_1 = 2$  and  $q_2 = 1$ . The DSRK2 scheme as  $q_1 = 1$  and  $q_2 = 2$ . It does not appear possible to create a two-stage second-order DSRK scheme with  $q_1 = q_2 = 2$ . However, we can find many three-stage second-order DSRK schemes with  $q_1 = q_2 = 2$ ; a particular example is the method we call DSRK32so2 with

$$\begin{array}{c|cc} \mathbf{c} & \mathbf{A} & \\ \hline \mathbf{b}^\top & & \end{array} = \begin{array}{c|ccc} 0 & \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 \\ 1 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \hline & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{array}, \quad \mathbf{W} = \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{12} & \frac{1}{12} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}.$$

Third-order, three-stage DSRK methods with  $q_1 = q_2 = 2$  can also be found, for example, DSRK33so2 with

$$\begin{array}{c|cc} \mathbf{c} & \mathbf{A} & \\ \hline \mathbf{b}^\top & & \end{array} = \begin{array}{c|ccc} 0 & \frac{1}{4} & -\frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{4} & \frac{1}{4} \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}, \quad \mathbf{W} = \begin{bmatrix} \frac{1}{3} & -\frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{12} & \frac{1}{12} \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{bmatrix}.$$

Recall that because the stage order of the underlying Runge–Kutta method exceeds one, none of these higher stage order schemes can be unconditionally contractive [itH96], and in

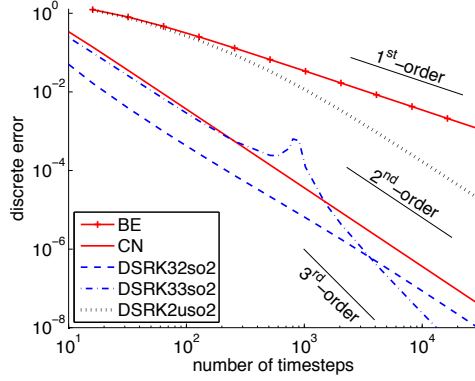


Figure 4.11: Stage order numerical convergence study for linear advection of a sine wave to  $t_f = 1$ . The spatial discretization here is first-order upwinding with  $m = 2048$  points.

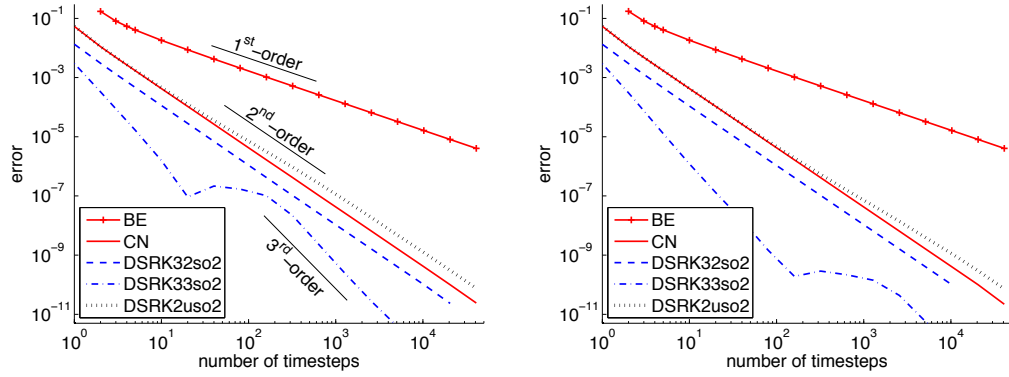


Figure 4.12: Stage order numerical convergence study on the van der Pol equation. Error shown is in the second component for  $\epsilon = 1 \times 10^{-3}$  (left) and  $\epsilon = 1 \times 10^{-4}$  (right).

numerical tests (not included) it was observed that indeed, DSRK2uso2, DSRK32so2 and DSRK33so2 violated the strong stability property for large enough  $\Delta t$ .

Figures 4.11 and 4.12 show that the DSRK32so2 scheme is free from order reduction. However, note that DSRK33so2 still exhibits order reduction as its stage order is one less than its design order.

The apparent importance of high stage order for DSRK schemes is intriguing, especially because order reduction is not observed when using implicit SSP schemes (which necessarily have stage order at most two) even when tested [KMG08] on some of the same test problems used here.

## Chapter 5

# Conclusions

The Closest Point Method is a recent technique for the solution of time dependent partial differential equations (PDEs) or other problems constrained to surfaces. The method propagates the solution of the PDE in time by alternating between an evolution step and an extension step in an embedding space surrounding the surface. This thesis contributed significantly to the development of the Closest Point Method in two main ways: first by applying the method to level set equations on surfaces; and second by deriving an implicit Closest Point Method.

Chapter 2 applied the Closest Point Method to level set equations on surfaces. The result is a robust technique for evolving interfaces on very general surfaces. In particular, the method retains the advantages of the level set method itself—i.e., it automatically handles self-intersecting interfaces and it makes use of standard high-order methods in the embedding space (typically  $\mathbb{R}^3$ ) to evolve the level set equations themselves. New in this chapter is the derivation of a Weighted Essentially Non-Oscillatory (WENO) based interpolation scheme suitable for use in the closest point extension step. The Closest Point Method, using this interpolation scheme together with standard Hamilton–Jacobi WENO discretizations for the evolution of the embedding level set equation, achieved fourth- and fifth-order results on convergence test problems for passive transport, normal flow and the reinitialization equation. Flows were computed on a sphere, a torus, a triangulation of the surface of a human hand and on the surface of the non-orientable, codimension-two Klein bottle, illustrating that the Closest Point Method is very flexible with respect to the geometry and dimension of the surface.

Chapter 3 outlined the development of a new implicit Closest Point Method. A new

notation expressing the evolution and extension steps of the original explicit Closest Point Method as matrix operations allowed the derivation of the implicit Closest Point Method essentially by taking the product of two matrices. The method then uses standard implicit linear multistep methods for time stepping and is well-suited to the numerical solution of PDEs with high-order differential terms. The implicit Closest Point Method allows large time steps, and example computations showed that it is well-suited for stiff problems involving the Laplace–Beltrami or in-surface biharmonic differential operators.

Chapters 2 and 3 both include numerical convergence studies of the Closest Point Method, indicating high-order results of up to fifth-order. These results are the first to demonstrate that the method is capable of solving surface problems with a high order of accuracy. That these results are obtained over a variety of examples including both hyperbolic and parabolic PDEs on surfaces ranging from simple spheres to Klein bottles and triangulated pigs is indicative of the broad applicability of the Closest Point Method.

Finally, Chapter 4 studied unconditionally contractive diagonally split Runge–Kutta (DSRK) methods. These high-order schemes appeared promising because they allow large time steps while maintaining nonlinear stability properties that might be desirable, for example, when applying the implicit Closest Point Method to nonlinear problems with nonsmooth solutions. However, using numerical test cases, it was shown that unconditionally contractive DSRK schemes suffer from severe order reduction. New DSRK stage order conditions were then derived and the order reduction of unconditionally contractive DSRK methods was shown to be associated with their necessarily low stage order. In conclusion, the class of unconditionally contractive DSRK methods does not produce viable alternatives to well-established conditionally SSP Runge–Kutta and linear multistep methods, be it for use with the implicit Closest Point Method or otherwise.

The Closest Point Method is very new and much work remains to be done on theory, implementation and applications. Applications of particular interest include image processing on surfaces, for example, image inpainting and segmentation. Other interesting problems include computation on moving surfaces, perhaps by combining the Closest Point Method with the method of [LZ08], and the computation of the harmonics or eigenmodes on a surface [Bra07].

## Appendix A

# Implicit Closest Point Method Algorithms

### A.1 Construction of the Computational Band

#### A.1.1 Pass one

Start with empty lists  $L_{\text{evolve}}$  and  $L_{\text{ghost}}$ . Given an arbitrary point  $\mathbf{x}$ , add the base point for  $\text{cp}(\mathbf{x})$  to  $L_{\text{evolve}}$  (we need a single point seeded within each non-connected part of the surface; for a connected surface, any point will do and the algorithm will find all appropriate points).

- Loop over each element  $\mathbf{x}$  of  $L_{\text{evolve}}$  (note that there is initially only one point in  $L_{\text{evolve}}$  but we will be adding elements as we go along: we stop when we actually get to the end of the list):
  - Loop over all the points  $\mathbf{y}$  in the evolution stencil surrounding  $\mathbf{x}$ :
    - Calculate  $\text{cp}(\mathbf{y})$ , and find  $\mathbf{b}$  the base point of the interpolation stencil.
    - For efficiency, if we've processed  $\mathbf{b}$  before then do not do anything (proceed to the next  $\mathbf{y}$ ); otherwise, process as follows.
    - Let  $S_{\text{interp}}^{\mathbf{b}} = \emptyset$ ; this list associated with  $\mathbf{b}$  will contain the indices of the points in the interpolation stencil for which  $\mathbf{b}$  is the base point.
    - For each  $\mathbf{z}$  in the interpolation stencil:
      - Look for  $\mathbf{z}$  in  $L_{\text{evolve}}$ , note its index if found.

- If  $z$  is not found, add it to  $L_{\text{evolve}}$  and note its index.
- Add the index of  $z$  to  $S_{\text{interp}}^b$ .

### A.1.2 Pass two

- Loop over each element  $x$  in  $L_{\text{evolve}}$ :
  - Set  $S_{\text{evolve}}^x = \emptyset$ ; this list associated with point  $x$  will contain the indices of the points in the evolution stencil at  $x$ .
  - Loop over all the points  $y$  in the evolution stencil surrounding  $x$ :
    - Find the index in  $L_{\text{evolve}}$  of the base point  $b$  corresponding to  $y$ . Save the base point index and the value of  $\text{cp}(y)$  with  $y$ .
    - Look for  $y$  in  $L_{\text{evolve}}$  and  $L_{\text{ghost}}$ , note its index if found.
    - If not found, add it to  $L_{\text{ghost}}$  and note its index.
    - Add the index of  $y$  and which band it is in to  $S_{\text{evolve}}^x$ .

## Appendix B

# WENO Schemes

### B.1 The Hamilton–Jacobi WENO Procedure

This appendix describes the fifth-order Hamilton–Jacobi WENO procedure following [OF03, Section 3.4]. Given seven data values  $\phi_{i-3}, \dots, \phi_{i+3}$  at seven equispaced grid points  $x_{i-3}, \dots, x_{i+3}$  with grid spacing  $\Delta x$ , the WENO procedure computes  $\phi_x^+$  and  $\phi_x^-$ , the right-biased and left-biased approximations to  $\phi_x(x_i)$ .

We begin by constructing the differences  $v_j$  from the seven  $\phi_j$  values as follows. To compute  $\phi_x^-$ , we use

$$v_1 = \frac{\phi_{i-2} - \phi_{i-3}}{\Delta x}, v_2 = \frac{\phi_{i-1} - \phi_{i-2}}{\Delta x}, v_3 = \frac{\phi_i - \phi_{i-1}}{\Delta x}, v_4 = \frac{\phi_{i+1} - \phi_i}{\Delta x}, v_5 = \frac{\phi_{i+2} - \phi_{i+1}}{\Delta x},$$

or to compute  $\phi_x^+$ , we use

$$v_1 = \frac{\phi_{i+3} - \phi_{i+2}}{\Delta x}, v_2 = \frac{\phi_{i+2} - \phi_{i+1}}{\Delta x}, v_3 = \frac{\phi_{i+1} - \phi_i}{\Delta x}, v_4 = \frac{\phi_i - \phi_{i-1}}{\Delta x}, v_5 = \frac{\phi_{i-1} - \phi_{i-2}}{\Delta x}.$$

Next we compute the smoothness indicators

$$\begin{aligned} S_1 &= \frac{13}{12} (v_1 - 2v_2 + v_3)^2 + \frac{1}{4} (v_1 - 4v_2 + 3v_3)^2, \\ S_2 &= \frac{13}{12} (v_2 - 2v_3 + v_4)^2 + \frac{1}{4} (v_2 - v_4)^2, \\ S_3 &= \frac{13}{12} (v_3 - 2v_4 + v_5)^2 + \frac{1}{4} (3v_3 - 4v_4 + v_5)^2. \end{aligned}$$

From the smoothness indicators, we compute

$$\alpha_1 = \frac{1}{10} \frac{1}{(\varepsilon + S_1)^2}, \quad \alpha_2 = \frac{6}{10} \frac{1}{(\varepsilon + S_2)^2}, \quad \alpha_3 = \frac{3}{10} \frac{1}{(\varepsilon + S_3)^2},$$



where  $\varepsilon$  prevents division-by-zero if one of the smoothness indicators is zero. The value  $\varepsilon = 1 \times 10^{-6}$  is used in the calculations in Chapter 2. From the  $\alpha$  values we compute the weights

$$w_1 = \frac{\alpha_1}{\alpha_1 + \alpha_2 + \alpha_3}, \quad w_2 = \frac{\alpha_2}{\alpha_1 + \alpha_2 + \alpha_3}, \quad w_3 = \frac{\alpha_3}{\alpha_1 + \alpha_2 + \alpha_3}.$$

Finally, the derivative is computed as

$$\phi_x^\pm = w_1 \frac{2v_1 - 7v_2 + 11v_3}{6} + w_2 \frac{-v_2 + 5v_3 + 2v_4}{6} + w_3 \frac{2v_3 + 5v_4 - v_5}{6}.$$

## Appendix C

# Explicit Strong Stability Preserving Runge–Kutta Schemes

### C.1 Common SSP Runge–Kutta Schemes

Some popular explicit SSP Runge–Kutta methods are given below. See [RS02, SR03, Got05, Ket08] for details.

**Two-stage, second-order SSP Runge–Kutta (SSP(2,2))** An optimal second-order SSP Runge–Kutta method given by

$$\begin{aligned}\bar{\mathbf{u}}^{(1)} &= \mathbf{u}^n + \Delta t \mathbf{f}(\mathbf{u}^n), \\ \mathbf{u}^{n+1} &= \frac{1}{2}\mathbf{u}^n + \frac{1}{2}\bar{\mathbf{u}}^{(1)} + \frac{1}{2}\Delta t \mathbf{f}(\bar{\mathbf{u}}^{(1)}).\end{aligned}$$

The time-step restriction for this scheme is  $\Delta t \leq \Delta t_{\text{FE}}$ , which means that it has a SSP coefficient of  $\mathcal{C} = 1$ . However, because it is a two-stage method, the computational work required per step is doubled compared to forward Euler so  $\mathcal{C}_{\text{eff}} = 0.5$ .

**Three-stage, third-order SSP Runge–Kutta (SSP(3,3))** An optimal third-order SSP Runge–Kutta method given by

$$\begin{aligned}\bar{\mathbf{u}}^{(1)} &= \mathbf{u}^n + \Delta t \mathbf{f}(\mathbf{u}^n), \\ \bar{\mathbf{u}}^{(2)} &= \frac{3}{4}\mathbf{u}^n + \frac{1}{4}\bar{\mathbf{u}}^{(1)} + \frac{1}{4}\Delta t \mathbf{f}(\bar{\mathbf{u}}^{(1)}), \\ \mathbf{u}^{n+1} &= \frac{1}{3}\mathbf{u}^n + \frac{2}{3}\bar{\mathbf{u}}^{(2)} + \frac{2}{3}\Delta t \mathbf{f}(\bar{\mathbf{u}}^{(2)}).\end{aligned}$$

The time-step restriction for this method is  $\Delta t \leq \Delta t_{\text{FE}}$ , so it has a value of  $\mathcal{C} = 1$ . However, the computational work per step in this method is three times that of forward Euler so  $\mathcal{C}_{\text{eff}} = \frac{1}{3}$ . This method is very commonly used and is often referred to as the third-order TVD Runge-Kutta scheme or the Shu–Osher method.

**Five-stage, fourth-order SSP Runge–Kutta (SSP(5,4))** An optimal method developed in [SR02, Ruu06, Kra91] with coefficients expressed to 15 digits is

$$\begin{aligned}
 \bar{\mathbf{u}}^{(1)} &= \mathbf{u}^n + 0.391752226571890\Delta t \mathbf{f}(\mathbf{u}^n), \\
 \bar{\mathbf{u}}^{(2)} &= 0.444370493651235\mathbf{u}^n + 0.555629506348765\bar{\mathbf{u}}^{(1)} + 0.368410593050371\Delta t \mathbf{f}(\bar{\mathbf{u}}^{(1)}), \\
 \bar{\mathbf{u}}^{(3)} &= 0.620101851488403\mathbf{u}^n + 0.379898148511597\bar{\mathbf{u}}^{(2)} + 0.251891774271694\Delta t \mathbf{f}(\bar{\mathbf{u}}^{(2)}), \\
 \bar{\mathbf{u}}^{(4)} &= 0.178079954393132\mathbf{u}^n + 0.821920045606868\bar{\mathbf{u}}^{(3)} + 0.544974750228521\Delta t \mathbf{f}(\bar{\mathbf{u}}^{(3)}), \\
 \mathbf{u}^{n+1} &= 0.517231671970585\bar{\mathbf{u}}^{(2)} + 0.096059710526146\bar{\mathbf{u}}^{(3)} + 0.063692468666290\Delta t \mathbf{f}(\bar{\mathbf{u}}^{(3)}) \\
 &\quad + 0.386708617503269\bar{\mathbf{u}}^{(4)} + 0.226007483236906\Delta t \mathbf{f}(\bar{\mathbf{u}}^{(4)}).
 \end{aligned}$$

The time-step restriction for this method is approximately  $\Delta t \leq 1.508\Delta t_{\text{FE}}$ , thus it has  $\mathcal{C} \approx 1.508$ . The computational work per step in this method is five times that of forward Euler, with  $\mathcal{C}_{\text{eff}} \approx 0.301$ ,

Finally note the recently proposed 10-stage, fourth-order SSP(10,4) scheme [Ket08] which has simple rational coefficients and improves considerably on the efficiency of SSP(5,4) with  $\mathcal{C}_{\text{eff}} = 0.6$  ( $\mathcal{C} = 6$ ).

# Bibliography

- [ABP<sup>+</sup>07] Helen Alexander, Anna Belkin, Chris Poss, Weining Wang, and Colin Macdonald. Modelling forest fires using level set equations. Technical report, MITACS BC Industrial Math Summer School, 2007.
- [ARS97] Uri M. Ascher, Steven J. Ruuth, and Raymond J. Spiteri. Implicit-explicit Runge–Kutta methods for time-dependent partial differential equations. *Appl. Numer. Math.*, 25(2-3):151–167, 1997.
- [ARW95] Uri M. Ascher, Steven J. Ruuth, and Brian T. R. Wetton. Implicit-explicit methods for time-dependent partial differential equations. *SIAM J. Numer. Anal.*, 32(3):797–823, 1995.
- [BCOS01] Marcelo Bertalmío, Li-Tien Cheng, Stanley Osher, and Guillermo Sapiro. Variational problems and partial differential equations on implicit surfaces. *J. Comput. Phys.*, 174(2):759–780, 2001.
- [BF01] Richard L. Burden and J. Douglas Faires. *Numerical Analysis*. Brooks/Cole, seventh edition, 2001.
- [BJZ94] A. Bellen, Z. Jackiewicz, and M. Zennaro. Contractivity of waveform relaxation Runge–Kutta iterations and related limit methods for dissipative systems in the maximum norm. *SIAM J. Numer. Anal.*, 31(2):499–523, 1994.
- [Bra07] Jeremy Brandman. A level-set method for computing the eigenvalues of elliptic operators defined on compact hypersurfaces. CAM Report 07-24, UCLA, 2007.
- [BS73] F. Black and M. Scholes. The Pricing of Options and Corporate Liabilities. *The Journal of Political Economy*, 81(3):637–654, 1973.
- [BT97] A. Bellen and L. Torelli. Unconditional contractivity in the maximum norm of diagonally split Runge–Kutta methods. *SIAM J. Numer. Anal.*, 34(2):528–543, 1997.
- [BT04] Jean-Paul Berrut and Lloyd N. Trefethen. Barycentric Lagrange interpolation. *SIAM Rev.*, 46(3):501–517, 2004.

- [But03] John C. Butcher. *Numerical Methods for Ordinary Differential Equations*. Wiley, 2003.
- [CBMO02] Li-Tien Cheng, Paul Burchard, Barry Merriman, and Stanley Osher. Motion of curves constrained on surfaces using a level-set approach. *J. Comput. Phys.*, 175(2):604–644, 2002.
- [CL84] M. G. Crandall and P.-L. Lions. Two approximations of solutions of Hamilton–Jacobi equations. *Math. Comp.*, 43(167):1–19, 1984.
- [Col06] Thomas Coleman. Option pricing: The hazards of computing delta and gamma. [http://www.fenews.com/fen49/where\\_num\\_matters/numerics.htm](http://www.fenews.com/fen49/where_num_matters/numerics.htm), 2006. Accessed 2006-08-08.
- [CT08] Li-Tien Cheng and Yen-Hsi Tsai. Redistancing by flow of time dependent eikonal equation. *J. Comput. Phys.*, 227(8):4002–4017, 2008.
- [FAMO99] Ronald P. Fedkiw, Tariq Aslam, Barry Merriman, and Stanley Osher. A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *J. Comput. Phys.*, 152(2):457–492, 1999.
- [Feh70] Erwin Fehlberg. Klassische Runge–Kutta-Formeln vierter und niedrigerer Ordnung mit Schrittweiten-Kontrolle und ihre Anwendung auf Wärmeleitungsprobleme. *Computing (Arch. Elektron. Rechnen)*, 6:61–71, 1970.
- [For05] P.A. Forsyth. An introduction to computational finance without agonizing pain. Available on author’s website, <http://www.cs.uwaterloo.ca/~paforsyt/agon.pdf>, February 2005. Accessed 2005-09-27.
- [FS04] L. Ferracina and M. N. Spijker. Stepsize restrictions for the total-variation-diminishing property in general Runge–Kutta methods. *SIAM J. Numer. Anal.*, 42(3):1073–1093, 2004.
- [FS05] L. Ferracina and M. N. Spijker. An extension and analysis of the Shu–Osher representation of Runge–Kutta methods. *Math. Comp.*, 74(249):201–219, 2005.
- [FS08] L. Ferracina and M. N. Spijker. Strong stability of singly-diagonally-implicit Runge–Kutta methods. *Appl. Numer. Math.*, 2008. To appear, doi:10.1016/j.apnum.2007.10.004.
- [GC06] Michael B. Giles and Rebecca Carter. Convergence analysis of Crank–Nicolson and Rannacher time-marching. *Journal of Computational Finance*, 9(4):89–112, 2006.
- [Gea71] C.W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice Hall, 1971.

- [GMR06] Sigal Gottlieb, Julia S. Mullen, and Steven J. Ruuth. A fifth order flux implicit WENO method. *J. Sci. Comput.*, 27(1-3):271–287, 2006.
- [Got05] Sigal Gottlieb. On high order strong stability preserving Runge–Kutta and multi step time discretizations. *J. Sci. Comput.*, 25(1-2):105–128, 2005.
- [Gre06] John B. Greer. An improvement of a recent Eulerian method for solving PDEs on general geometries. *J. Sci. Comput.*, 29(3):321–352, 2006.
- [GS98] Sigal Gottlieb and Chi-Wang Shu. Total variation diminishing Runge–Kutta schemes. *Math. Comp.*, 67(221):73–85, 1998.
- [GST01] Sigal Gottlieb, Chi-Wang Shu, and Eitan Tadmor. Strong stability-preserving high-order time discretization methods. *SIAM Rev.*, 43(1):89–112, 2001.
- [GVL96] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1996.
- [Hea97] Michael T. Heath. *Scientific Computing: An Introductory Survey*. McGraw-Hill, 1997.
- [Hig04] Inmaculada Higuera. On strong stability preserving time discretization methods. *J. Sci. Comput.*, 21(2):193–223, 2004.
- [Hig05] Inmaculada Higuera. Representations of Runge–Kutta methods and strong stability preserving methods. *SIAM J. Numer. Anal.*, 43(3):924–948, 2005.
- [HNW93] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving ordinary differential equations I: Nonstiff problems*, volume 8 of *Springer Series in Computational Mathematics*. Springer-Verlag, second edition, 1993.
- [Hor98] Zoltán Horváth. Positivity of Runge–Kutta and diagonally split Runge–Kutta methods. *Appl. Numer. Math.*, 28(2-4):309–326, 1998.
- [HR06] Willem Hundsdorfer and Steven J. Ruuth. On monotonicity and boundedness properties of linear multistep methods. *Math. Comp.*, 75(254):655–672, 2006.
- [HRS03] Willem Hundsdorfer, Steven J. Ruuth, and Raymond J. Spiteri. Monotonicity-preserving linear multistep methods. *SIAM J. Numer. Anal.*, 41(2):605–623, 2003.
- [Hui08] Annie Hui. “Annie Hui’s pig”, the AIM@SHAPE shape repository. <http://shapes.aimatshape.net>, 2008. Accessed 2008-04-09.
- [HW96] E. Hairer and G. Wanner. *Solving ordinary differential equations. II: Stiff and differential-algebraic problems*, volume 14 of *Springer Series in Computational Mathematics*. Springer-Verlag, second edition, 1996.

- [itH96] K. J. in 't Hout. A note on unconditional maximum norm contractivity of diagonally split Runge–Kutta methods. *SIAM J. Numer. Anal.*, 33(3):1125–1134, 1996.
- [JOP<sup>+</sup>01] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001. <http://www.scipy.org>.
- [JP00] Guang-Shan Jiang and Danping Peng. Weighted ENO schemes for Hamilton–Jacobi equations. *SIAM J. Sci. Comput.*, 21(6):2126–2143, 2000.
- [JS96] Guang-Shan Jiang and Chi-Wang Shu. Efficient implementation of weighted ENO schemes. *J. Comput. Phys.*, 126(1):202–228, 1996.
- [Ket08] David I. Ketcheson. Highly efficient strong stability-preserving Runge–Kutta methods with low-storage implementations. *SIAM J. Sci. Comput.*, 30(4):2113–2136, 2008.
- [KMG08] David I. Ketcheson, Colin B. Macdonald, and Sigal Gottlieb. Optimal implicit strong stability preserving Runge–Kutta methods. *Appl. Numer. Math.*, 2008. To appear, doi:10.1016/j.apnum.2008.03.034.
- [Kra91] J. F. B. M. Kraaijevanger. Contractivity of Runge–Kutta methods. *BIT*, 31(3):482–528, 1991.
- [Küh05] W. Kühnel. *Differential Geometry: Curves – Surface – Manifolds*. American Mathematical Society, second edition, 2005.
- [Lan98] Culbert B. Laney. *Computational gasdynamics*. Cambridge University Press, 1998.
- [Len89] H. W. J. Lenferink. Contractivity preserving explicit linear multistep methods. *Numer. Math.*, 55(2):213–223, 1989.
- [Len91] H. W. J. Lenferink. Contractivity-preserving implicit linear multistep methods. *Math. Comp.*, 56(193):177–199, 1991.
- [LM05] Anita T. Layton and Michael L. Minion. Implications of the choice of quadrature nodes for Picard integral deferred corrections methods for ordinary differential equations. *BIT*, 45(2):341–373, 2005.
- [LOC94] Xu-Dong Liu, Stanley Osher, and Tony Chan. Weighted essentially non-oscillatory schemes. *J. Comput. Phys.*, 115(1):200–212, 1994.
- [LZ08] Shingyu Leung and Hongkai Zhao. A grid based particle method for moving interface problems. CAM Report 08-08, UCLA, 2008.

- [Mac03] Colin B. Macdonald. Constructing high-order Runge–Kutta methods with embedded strong-stability-preserving pairs. Master’s thesis, Simon Fraser University, August 2003.
- [MGR08] Colin B. Macdonald, Sigal Gottlieb, and Steven J. Ruuth. A numerical study of diagonally split Runge–Kutta methods for PDEs with discontinuities. *J. Sci. Comput.*, 2008. To appear, doi:10.1007/s10915-007-9180-6.
- [Mit04] Ian Mitchell. A toolbox of level set methods. Technical Report TR-2004-09, University of British Columbia Department of Computer Science, July 2004. <http://www.cs.ubc.ca/~mitchell/ToolboxLS/Papers/Toolbox/toolboxLS-1.0.pdf>.
- [MR] Barry Merriman and Steven J. Ruuth. Embedding methods for the numerical solution of PDEs on manifolds. In preparation.
- [MR07] Barry Merriman and Steven J. Ruuth. Diffusion generated motion of curves on surfaces. *J. Comput. Phys.*, 225(2):2267–2282, 2007.
- [MR08] Colin B. Macdonald and Steven J. Ruuth. Level set equations on surfaces via the Closest Point Method. *J. Sci. Comput.*, 2008. To appear, doi:10.1007/s10915-008-9196-6.
- [Mur03] J.D. Murray. *Mathematical Biology II: Spatial Models and Biomedical Applications*, volume 18 of *Interdisciplinary Applied Mathematics*. Springer, third edition, 2003.
- [OF03] Stanley Osher and Ronald Fedkiw. *Level set methods and dynamic implicit surfaces*, volume 153 of *Applied Mathematical Sciences*. Springer-Verlag, 2003.
- [Oli07] Travis E. Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007.
- [OS88] Stanley Osher and James A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton–Jacobi formulations. *J. Comput. Phys.*, 79(1):12–49, 1988.
- [OS91] Stanley Osher and Chi-Wang Shu. High-order essentially nonoscillatory schemes for Hamilton–Jacobi equations. *SIAM J. Numer. Anal.*, 28(4):907–922, 1991.
- [PL68] I. Prigogine and R. Lefever. Symmetry breaking instabilities in dissipative systems. II. *The Journal of Chemical Physics*, 48(4):1695–1700, 1968.
- [Pow99] D.L. Powers. *Boundary value problems*. Academic Press, 1999.
- [PR74] A. Prothero and A. Robinson. On the stability and accuracy of one-step methods for solving stiff systems of ordinary differential equations. *Math. Comp.*, 28:145–162, 1974.



- [RM08] Steven J. Ruuth and Barry Merriman. A simple embedding method for solving partial differential equations on surfaces. *J. Comput. Phys.*, 227(3):1943–1961, 2008.
- [RS00] Giovanni Russo and Peter Smereka. A remark on computing distance functions. *J. Comput. Phys.*, 163(1):51–67, 2000.
- [RS02] Steven J. Ruuth and Raymond J. Spiteri. Two barriers on strong-stability-preserving time discretization methods. *J. Sci. Comput.*, 17(1-4):211–220, 2002. Proceedings of the Fifth International Conference on Spectral and High Order Methods (ICOSAHOM-01).
- [Run01] Carl Runge. Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten. *Zeit. für Math. und Phys.*, 46:224–243, 1901.
- [Ruu95] Steven J. Ruuth. Implicit-explicit methods for reaction-diffusion problems in pattern formation. *J. Math. Bio.*, 34(2):148–176, 1995.
- [Ruu06] Steven J. Ruuth. Global optimization of explicit strong-stability-preserving Runge–Kutta methods. *Math. Comp.*, 75(253):183–207, 2006.
- [SAA07] L. Saboret, M. Attene, and P. Alliez. “Laurent’s Hand”, the AIM@SHAPE shape repository. <http://shapes.aimatshape.net>, 2007. Accessed 2007-02-16.
- [Set99] J. A. Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*, volume 3 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, second edition, 1999.
- [Shu88] Chi-Wang Shu. Total-variation-diminishing time discretizations. *SIAM J. Sci. Statist. Comput.*, 9(6):1073–1084, 1988.
- [Shu97] Chi-Wang Shu. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws. NASA CR-97-206253 ICASE Report 97-65, Institute for Computer Applications in Science and Engineering, 1997.
- [SML98] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit: An Object-Oriented Approach To 3D Graphics*. Prentice Hall, 1998.
- [SO88] Chi-Wang Shu and Stanley Osher. Efficient implementation of essentially nonoscillatory shock-capturing schemes. *J. Comput. Phys.*, 77(2):439–471, 1988.
- [Spi83] M. N. Spijker. Contractivity in the numerical solution of initial value problems. *Numer. Math.*, 42(3):271–290, 1983.

- [SR02] Raymond J. Spiteri and Steven J. Ruuth. A new class of optimal high-order strong-stability-preserving time discretization methods. *SIAM J. Numer. Anal.*, 40(2):469–491, 2002.
- [SR03] Raymond J. Spiteri and Steven J. Ruuth. Non-linear evolution using optimal fourth-order strong-stability-preserving Runge–Kutta methods. *Math. Comput. Simulation*, 62(1-2):125–135, 2003.
- [SS03] Kurt Sebastian and Chi-Wang Shu. Multidomain WENO finite difference method with interpolation at subdomain interfaces. *J. Sci. Comput.*, 19(1-3):405–438, 2003.
- [SSO94] M. Sussman, P. Smereka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flow. *J. Comput. Phys.*, 114(1):146–159, 1994.
- [ST04] N. V. Sahinidis and M. Tawarmalani. *BARON 7.2: Global Optimization of Mixed-Integer Nonlinear Programs*, User’s Manual, 2004. Available at <http://www.gams.com/dd/docs/solvers/baron.pdf>, Accessed 2005-01-17.
- [TB97] L.N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, 1997.
- [Tho95] J.W. Thomas. *Numerical Partial Differential Equations: Finite Difference Methods*, volume 22 of *Texts in Applied Mathematics*. Springer, 1995.
- [TL94] Greg Turk and Marc Levoy. The Stanford Bunny, the Stanford 3D scanning repository. <http://www-graphics.stanford.edu/data/3Dscanrep>, 1994. Accessed 2008-06-18.
- [Tur91] Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *Computer Graphics*, 25(4):289–298, 1991.
- [vR<sup>+</sup>91] Guido van Rossum et al. The Python programming language, 1991. <http://www.python.org>.
- [Wik07] Wikipedia contributors. Klein bottle. Wikipedia, the free encyclopedia, [http://en.wikipedia.org/w/index.php?title=Klein\\_bottle&oldid=133679151](http://en.wikipedia.org/w/index.php?title=Klein_bottle&oldid=133679151), May 2007. Accessed 2007-05-29.
- [YZE04] Lingfa Yang, Anatol M. Zhabotinsky, and Irving R. Epstein. Stable squares and other oscillatory Turing patterns in a reaction-diffusion model. *Phys. Rev. Lett.*, 92(19):198–303, May 2004.
- [Zen93] M. Zennaro. Contractivity of Runge–Kutta methods with respect to forcing terms. *Appl. Numer. Math.*, 11(4):321–345, 1993.

# Index

- algorithm, 106
- backward difference formula, 6–8
- backward Euler, 6, 88
- barycentric Lagrange formula, 11
- barycentric Lagrange interpolation, 10
- BDF, 6
- BDF-2, 6
- BDF-3, 7
- biharmonic, 70
- biharmonic operator, 70
- Black–Scholes, 92
- Black–Scholes equation, 87
- Burgers’ equation, 97
- closest point extension, 15, 49
- closest point function, 14
- closest point representation, 13
- consistency, 70
- contractivity, 83
- Crank–Nicolson, 6, 80, 85, 88
- diagonal splitting function, 77, 82
- diagonally split Runge–Kutta methods, 77, 82
- discrete extension operator, 49
- dissipative system, 83
- DSRK, 84
- DSRK method, 82
- effective SSP coefficient, 80
- eigenmodes, 56
- eigenvalues, 56
- embedding methods, 13
- Embedding methods, 13
- evolution stencil, 50
- extension operator, 70
- forward Euler, 5, 80
- GMRES, 61, 66, 74
- Hamiltonian, 3
- IMEX Euler, 8
- IMEX scheme, 7
- IMEX schemes, 75
- implicit backward, 80
- implicit representation, 14
- implicit-explicit schemes, 75
- interface, 104
- interpolant, 10
- Interpolation, 9
- interpolation stencil, 48
- intrinsic, 12
- Laplace–Beltrami, 13, 16, 50, 70
- level set method, 104
- Level set methods, 22
- linear multistep method, 84
- linear multistep methods, 6, 52
- linear stability, 46
- Matlab, 61, 74, 87
- Matlab backslash operator, 61, 66, 73, 87
- numerical convergence study, 88
- numerical Hamiltonian, 4
- NumPy, 67
- order reduction, 99, 100
- pattern formation, 74

- Python, 61, 67
- reaction-diffusion equation, 74
- rounding error, 74
- Runge–Kutta, 5, 46
- SBDF-2 IMEX scheme, 8
- SciPy, 67
- spatial discretization, 77
- spectrum, 56
- spherical harmonics, 66, 74
- SSP, 5
- SSP(3,3), 5
- stability, 70, 73
- stage order, 99, 100, 102
- stiff, 46
- stiffness, 6, 7, 99, 100
- strong stability preserving, 84
- strong stability, 79
- strong stability preserving, 5, 9
- surface gradient, 70
- TVD spatial discretizations, 80
- unconditional contractivity, 83
- unconditionally contractive, 99
- van der Pol equation, 99
- Weighted essentially non-oscillatory, *see* WENO
- WENO, 108
  - Hamilton–Jacobi, 4, 23
  - interpolation, 12, 21, 25, 39