Non-oscillatory interpolation for the Semi-Lagrangian scheme

Tomos Wyn Roberts

Abstract

In this dissertation we are concerned with the study of various interpolation methods for use with the semi-lagrangian scheme. In particular we are interested in the limited form of the divided difference interpolation method as suggested by M.Berzins [1], because of its parallels with ENO type numerical schemes in reducing the oscillations in a solution.

It is found that this new interpolation compares favourably with standard polynomial interpolation when approximating Runge's function on a standard mesh. Moving on to semi-lagrangian schemes we see how the new divided difference interpolation method offers no improvement over existing methods when modelling a square wave with passive advection in 1-D.

In the final chapter we examine two methods of departure point calculation for the semi-lagrangian scheme. When modelling a non-linear equation with a smooth initial condition we see that Berzins' interpolation performs rather poorly if used in conjunction with these methods and the semi-lagrangian scheme. If we change the initial condition for a function that has discontinuities we see that it performs rather better.

Finally we provide a summary and offer some suggestions for further work.

Acknowledgements

Many thanks to my supervisor Mike Baines for all his patience and help along the way. Thanks also to Amos Lawless for sharing his knowledge on semi-lagrangian schemes. I would also like to thank NERC for their financial support while completing my masters.

Declaration

I confirm that this is my own work and the use of all material from other sources has been properly and fully acknowledged.

Signed .....

Contents

Chapter 1

Introduction

In this dissertation we compare different interpolation methods for use with the semi-lagrangian scheme, a type of numerical advection scheme used extensively in weather forecasting. In particular we look for an interpolation method that will successfully reduce oscillations in our solution. We are mainly concerned with a certain interpolation method put forward by Berzins [1] and how it behaves in relation to other well-known interpolation schemes when modelling simple advection problems.

1.1 The Semi-Lagrangian method

The semi-lagrangian scheme is a type of numerical advection scheme that is often used in weather forecasting. Suppose we wish to model the flow of some arbitrary fluid. We can choose to simulate the flow using a variety of numerical methods.

Schemes that treat the flow from an Eulerian viewpoint would use a fixed computational grid, where each gridpoint has associated values for certain variables. We would solve the problem from a fixed frame of reference, observing the flow. These schemes are unstable and inaccurate for large timesteps.

For Lagrangian schemes on the other hand, the frame of reference moves with the flow and maps the trajectory of each individual particle. Lagrangian

schemes are stable for large timesteps, but the particles may spread out over a large area, or become 'tangled' in a small area, making it difficult to estimate gradients etc. which results in a less accurate model.

Semi-Lagrangian schemes are a combination of the above schemes, where we try to take the best properties from the two. We keep the fixed computational grid from the Eulerian frame of reference schemes but still have stability for large timesteps, as for Lagrangian schemes. The basic principle involves using a different set of particles for each timestep, and using values at the gridpoints from the previous timestep to approximate the gridpoint values for each new timestep.

We can use the semi-lagrangian scheme to solve a variety of advection equations.

1.1.1 1-D Advection Equation

The 1-D advection equation with constant velocity a is given by

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0 \qquad \text{with } u(0) = u_0 \tag{1.1}$$

where $u = u(x, t)$.

We can solve this equation using the method of characteristics, which gives two further equations:

$$\frac{dx}{dt} = a \qquad \Rightarrow \qquad x = x_0 + at \text{ (these are the characteristic curves)}$$
$$\frac{du}{dt} = 0 \qquad \Rightarrow \qquad u(x(t), t) = constant \text{ along the characteristics.}$$

The second equation shows that u is constant along the characteristics found by solving the first equation.

Suppose we know the solution u to equation (1.1) at time $t = t_n$, and we wish to find the solution at the next timestep $t = t_n + \Delta t = t_{n+1}$ at some $x = x_a$ (known as the arrival point.) We draw a characteristic line back to $t = t_n$, where $x = x_d$ (known as the departure point). We can visualise this by looking at the diagram found in [4], see figure(1.1).

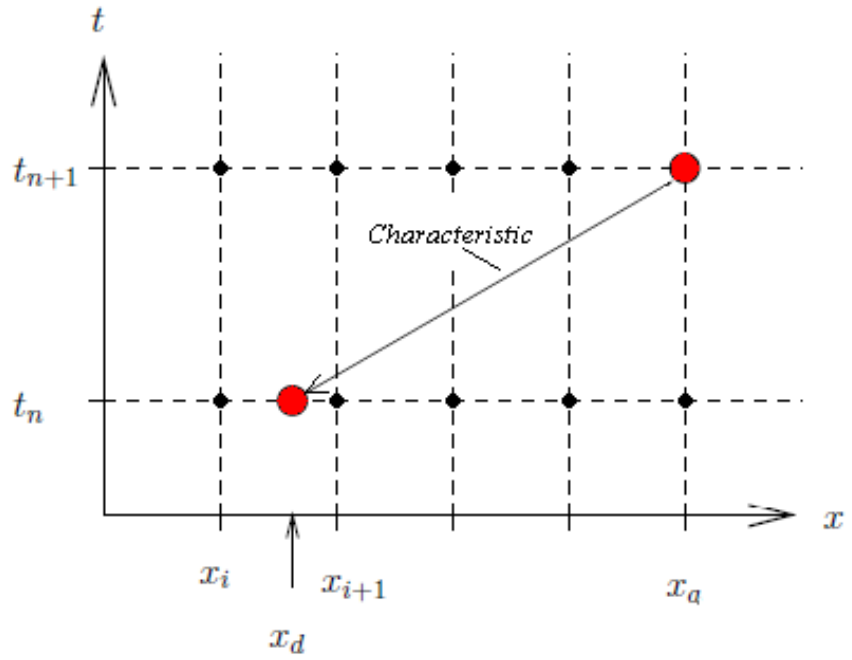

Figure 1.1: Diagram showing a brief outline of the semi-lagrangian scheme.

Since u is constant along the characteristic, the solution $u(x_a, t_{n+1})$ will be the same as $u(x_d, t_n)$. So all we need to solve the equation (1.1) at $x = x_a$ is the value of u at $x = x_d$.

If $u(x_d, t_n)$ happens to lie on a gridpoint, then since we know the solution at the gridpoints at $t = t_n$ then we have our answer to $u(x_a, t_{n+1})$, as it is simply equal to $u(x_d, t_n)$.

If $u(x_d, t_n)$ does not lie on a gridpoint (as is mostly the case) we must estimate its value by using the values that we already know.
Thus we interpolate the value of u at $x = x_d$ at time t_n by using the values of u at neighbouring gridpoints. The better our interpolation method, the more accurate our solution will be to the advection equation.
This then motivates us to study different types of interpolation methods to

see which one can give us the best results for this scheme.

In chapter two we look various types of well known interpolation methods and introduce an interpolation scheme suggested by Berzins [1]. In chapter three we compare results for the various interpolation methods when approximating a function on a standard mesh and also when modelling a 1-D advection problem with the semi-lagrangian scheme. Chapter four will be concerned with the modelling of a non-linear equation using the semi-lagrangian scheme along with various interpolation methods. We introduce two different methods for departure point calculation, the midpoint method and the Shu-Osher Runge-Kutta method. We finish with a summary of the work undertaken.

Chapter 2

Interpolation Methods

Interpolation is a process where given a set of function values at unique data points, we estimate the values of the function at a new set of data points. The new data points must be within the range of the original set.

2.1 Linear Interpolation

One of the simplest methods of interpolating a given data set is by linear interpolation. Given two points in the $x-y$ plane, (x_1, y_1) and (x_2, y_2), with $x_1 \neq x_2$ then we can form a first order polynomial (a straight line) between the two points. We call this polynomial the interpolant. Our approximation to the function at any intervening point (x, y) is then given by

$$y = y_1 + (x - x_1)\frac{(y_2 - y_1)}{(x_2 - x_1)}$$

For example if we know that a function has values of 1 and 6 at $x = 0, 2$ then our linear interpolation estimation to the value of the function at $x = 1$ is

$$y = 1 + (1 - 0)\frac{(6 - 1)}{(2 - 0)} = 3.5.$$

The disadvantages of linear interpolation are that it is not very accurate, and we cannot differentiate the interpolant at the data points. Nevertheless it is easy to use and can provide a quick solution to a problem.

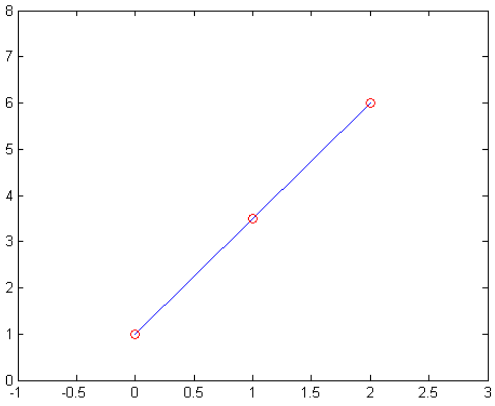

Figure 2.1: Linear interpolation between the points $(0, 1)$ and $(2, 6)$

2.2 Polynomial Interpolation

We can extend linear interpolation to more than two data points. Given a set of n data points where function values are defined, (x_k, y_k) $k = 1, \ldots, n$, there exists a unique polynomial, $P(x)$ (again called the interpolant), of order less than n that passes exactly through each point, i.e.

$$P(x) = y_k, \;\; k = 1, \ldots n \tag{2.1}$$

At any point (x, y) that is within the range of the original data points $P(x)$ is given by the Lagrangian interpolating polynomial

$$\begin{aligned} P(x) \quad = \quad & \left(\frac{x - x_2}{x_1 - x_2}\right)\left(\frac{x - x_3}{x_1 - x_3}\right) \times \ldots \times \left(\frac{x - x_n}{x_1 - x_n}\right) \times y_1 \\ + \quad & \left(\frac{x - x_1}{x_2 - x_1}\right)\left(\frac{x - x_3}{x_2 - x_3}\right) \times \ldots \times \left(\frac{x - x_n}{x_2 - x_n}\right) \times y_2 + \ldots \\ \ldots \quad + & \left(\frac{x - x_1}{x_n - x_1}\right)\left(\frac{x - x_2}{x_n - x_2}\right) \times \ldots \times \left(\frac{x - x_{n-1}}{x_n - x_{n-1}}\right) \times y_n \end{aligned}$$

or

$$P(x) = \sum_k \left(\prod_{j \neq k} \frac{x - x_j}{x_k - x_j} \right) y_k.$$

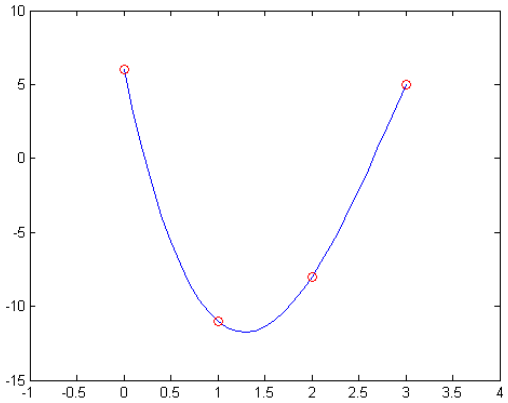

Figure 2.2: Polynomial interpolation using the data set $x = (0, 1, 2, 3),\ y = (6, -11, -8, 5)$

As an example consider the following data set:

$$x = (0, 1, 2, 3),\ y = (6, -11, -8, 5)$$

Using these points the interpolating polynomial is given by

$$\begin{aligned} P(x) &= \frac{(x-1)(x-2)(x-3)}{(-6)}(6) + \frac{x(x-2)(x-3)}{(2)}(-11) \\ &+ \frac{x(x-1)(x-3)}{(-2)}(-8) + \frac{x(x-1)(x-2)}{(6)}(5) \end{aligned}$$

or, written in power form (see below)

$$P(x) = -\frac{5}{3}x^3 + 15x^2 - \frac{91}{3}x + 6$$

The power form of an $n-1^{th}$ degree polynomial takes the form

$$P(x) = a_n x^{n-1} + a_{n-1}x^{n-2} + a_{n-2}x^{n-3} + \ldots + a_2 x + a_1$$

Substituting this into (2.1) we get a system of simultaneous linear equations, which we can write in matrix form as

$$\begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \ldots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \ldots & x_2 & 1 \\ \vdots & \vdots & \vdots & & \vdots \\ x_n^{n-1} & x_n^{n-2} & \ldots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

The matrix on the left is known as the Vandermonde Matrix. This matrix may be very badly conditioned leading to very large errors when we try to solve the above system using standard methods such as Gaussian elimination. We might use polynomial interpolation to solve a problem given a handful of evenly spaced data points, but we will experience difficulties if we try to use it as a general method.

2.3 Piecewise Linear Interpolation

Piecewise linear interpolation is an extension of linear interpolation to n points, where $n > 2$. Given a data set (x_k, y_k) $k = 1, \ldots, n$ we perform linear interpolation between each point within the set. To interpolate the value of y at any intervening point x, we must firstly locate it within our data set, i.e. find k where

$$x_k \leq x < x_{k+1}.$$

k is referred to as the interval index. We then proceed to find the distance s from x to the data point directly to the left of it

$$s = x - x_k$$

and then the first divided difference

$$\delta_k = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

Note that s and δ_k are unique to every interval within the data set. They are thus referred to as local variables

Once these three quantities are known, the interpolating value at x is given by

$$\begin{aligned} P(x) &= y_k + (x - x_k)\frac{y_{k+1} - y_k}{x_{k+1} - x_k} \\ &= y_k + s\delta_k \end{aligned}$$

which gives a straight line that passes through (x_k, y_k) and (x_{k+1}, y_{k+1}), with

$$P(x_k) = y_k \qquad P(x_{k+1}) = y_{k+1}$$

P is a continuous function but its derivative P' is discontinuous at the data points.

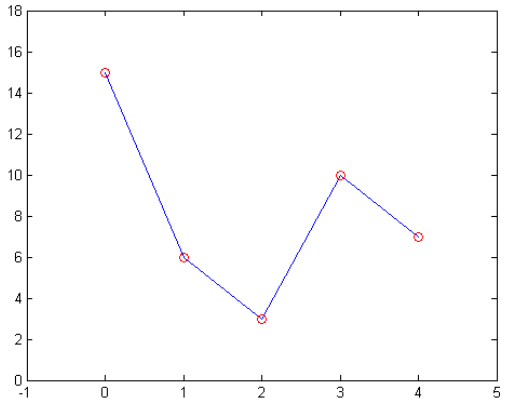

Figure 2.3: Piecewise linear interpolation using the data set $x = (0, 1, 2, 3, 4)$, $y = (15, 6, 3, 10, 7)$

2.4 Cubic Hermite Interpolation

As we can see from the graph at the end of the previous section, piecewise linear interpolation does not give a 'smooth' graph; it has jagged corners at the data points. We would prefer it if our interpolant was 'smoother', i.e. had a continuous first derivative.

Let us consider piecewise cubic Hermite interpolation. For a set of data points (x_k, y_k) $k = 1, \ldots, n$ piecewise linear interpolation satisfies

$$P(x_k) = y_k \qquad P(x_{k+1}) = y_{k+1}$$

at every point. As well as satisfying the above, Hermite cubics also satisfy conditions on the derivative of the interpolant at the data points.

$$P'(x_k) = y'_k \qquad P'(x_{k+1}) = y'_{k+1}$$

Whereas with piecewise linear interpolation two conditions led to a linear interpolant, we now have four interpolating conditions which gives a cubic interpolant.

Interpolating using three nodes would give six interpolation conditions, resulting in a quintic interpolant. As we can see the more points that are used the higher the order of the interpolating polynomial.

We will concentrate on piecewise cubic Hermite interpolation, and so will only be concerned with two neighbouring data points at a time, x_k and x_{k+1}.

The cubic Hermite interpolant at any point x, with $x_k < x < x_{k+1}$, takes the form

$$P(x) = \frac{3hs^2 - 2s^3}{h^3} y_{k+1} + \frac{h^3 - 3hs^2 + 2s^3}{h^3} y_k + \frac{s^2(s-h)}{h^2} d_{k+1} + \frac{s(s-h)^2}{h^2} d_k$$

where

$$\begin{aligned} h &= x_{k+1} - x_k \\ \delta_k &= \frac{y_{k+1} - y_k}{h_k} \\ d_k &= P'(x_k) \\ s &= x - x_k \end{aligned}$$

Piecewise cubic Hermite interpolation can prove very useful if we are given known function values and the first derivatives at a set of data points. If we are not provided with derivative values at these points and still wish to use piecewise cubic Hermite interpolation, then we need a way of defining the first derivatives ourselves.

2.5 Shape-Preserving Piecewise Cubic (pchip)

Shape-preserving piecewise cubic interpolation (called in Matlab using the **pchip** command) takes the same form as piecewise cubic Hermite interpolation, but differs in that the first derivatives d_k at the nodes (x_k, y_k) $k = 1, \ldots n$ are defined in a special way. This new method ensures that the value of the interpolant stays within the range of the local data points, i.e. the new function values do not overshoot the function values at the ends of each interval.

Again we form the first divided difference

$$\delta_k = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}.$$

If the intervals between the data points are all of the same length, then for the inner points (x_k, y_k), $k = 2, \ldots n-1$, the derivative is given by the

'harmonic mean' of the two differences δ_k and δ_{k-1}

$$\frac{1}{d_k} = \frac{1}{2}\left(\frac{1}{\delta_{k+1}} + \frac{1}{\delta_k}\right)$$

If δ_k and δ_{k-1} have different signs or if one is zero, then we set $d_k = 0$, since this means that (x_k, y_k) will be a stationary point.

If δ_k and δ_{k-1} have the same sign but the intervals are not of the same lengths, then d_k is given by the 'weighted harmonic mean'

$$\frac{w_1 + w_2}{d_k} = \frac{w_1}{\delta_{k-1}} + \frac{w_2}{\delta_k}$$

with weights

$$w_1 = 2k_k + h_{k-1}, \qquad w_2 = k_k + 2h_{k-1}$$

where

$$h_k = x_{k+1} - x_k, \qquad h_{k-1} = x_k - x_{k-1}.$$

A different non-centred shape preserving 3-point formula is used to define the derivatives at the endpoints (x_1, y_1) and (x_n, y_n).

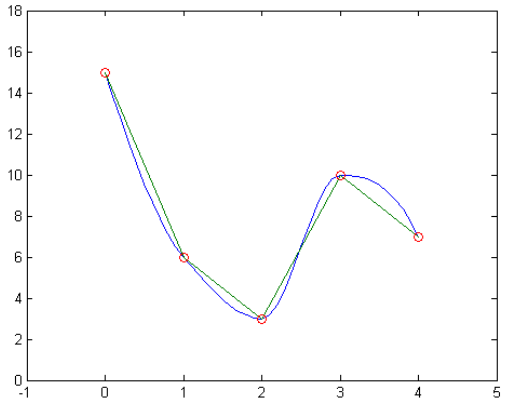

Figure 2.4: pchip (blue) and piecewise linear interpolation (green) using the data set $x = (0, 1, 2, 3, 4),\ y = (15, 6, 3, 10, 7)$

Figure (2.4) shows how pchip (blue line) compares with piecewise linear interpolation (green line). pchip produces a smooth interpolant that has a

continuous first derivative at the data points, as compared to the 'jagged' linear interpolant. We can also see that the the pchip interpolant does not overshoot the data points at the ends of each interval.

2.6 Divided Difference Polynomial Interpolation

We now proceed to look the interpolation method described by M.Berzins in his paper on Adaptive Polynomial Interpolation in SIAM Review, vol. 49 [1].

These methods are motivated by the ENO (Essentially non-Oscillatory) schemes for the numerical solution solution of hyperbolic conservation laws.

Consider a set of $n+1$ data points $(x_i, y_i), \qquad i = 0, \ldots n$, where $y_i = U(x_i)$ on the interval $[-1, 1]$. Let $U^L(x)$ be an approximating polynomial to the data set.
For this interpolation method we will use the standard notation for divided differences:

$$U[x_i] = U(x_i) \qquad \text{and} \qquad U[x_i, x_{i+1}] = \frac{U[x_{i+1}] - U[x_i]}{x_{i+1} - x_i}$$

with higher order differences given by

$$U[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{U[x_{i+1}, x_{i+2}, \ldots, x_{i+k}] - U[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}$$

For example for $i = 0$ the first few divided differences are

$$\begin{aligned} U[x_0] &= U(x_0) \\ U[x_0, x_1] &= \frac{U[x_1] - U[x_0]}{x_1 - x_0} \\ U[x_0, x_1, x_2] &= \frac{U[x_1, x_2] - U[x_0, x_1]}{x_2 - x_0} \\ &= \frac{\frac{U[x_2]-U[x_1]}{x_2-x_1} - \frac{U[x_1]-U[x_0]}{x_1-x_0}}{x_2 - x_0} \\ &\textit{etc.} \end{aligned}$$

It is often easier to picture divided differences if we form a difference table

$$\begin{array}{llll}
x_0 & U[x_0] & & \\
 & & U[x_0,x_1] & \\
x_1 & U[x_1] & & U[x_0,x_1,x_2] \\
 & & U[x_1,x_2] & & U[x_0,x_1,x_2,x_3] \\
x_2 & U[x_2] & & U[x_1,x_2,x_3] \\
 & & U[x_2,x_3] & \\
x_3 & U[x_3] & &
\end{array}$$

For meshpoints $x_i, x_{i+1}, \ldots, x_{i+n}$ with corresponding solution values $U[x_i], \ldots, U[x_{i+n}]$ the standard divided difference interpolating polynomial (also known as the Newton polynomial) is given by

$$U^L(x) = U[x_i] + \pi_{1,i}(x)\, U[x_i, x_{i+1}] + \pi_{2,i}(x)\, U[x_i, x_{i+1}, x_{i+2}] + \ldots + \pi_{n,i}(x)\, U[x_i, \ldots, x_{i+n}] \tag{2.2}$$

where the π-functions are defined as

$$\begin{aligned}
\pi_{1,i}(x) &= (x - x_i) \\
\pi_{2,i}(x) &= (x - x_i)(x - x_{i+1}) \\
\pi_{3,i}(x) &= (x - x_i)(x - x_{i+1})(x - x_{i+2}) \\
&\vdots
\end{aligned}$$

Note that for each extra term we use the next mesh point to the right. There is no need to adhere to this rule, in fact we may use successive meshpoints to the left or right of x_i as we please.

For example, suppose $i > 1$ and we want a quadratic interpolant on the interval $[x_i, x_{i+1}]$. One way of constructing the interpolant is by using the first three terms of (2.2)

$$U^L(x) = U[x_i] + \pi_{1,i}\, U[x_i, x_{i+1}] + \pi_{2,i}\, U[x_i, x_{i+1}, x_{i+2}] \tag{2.3}$$

where for each successive term we use the next meshpoint to the right.

Alternatively an equally valid interpolant is given by

$$U^L(x) = U[x_i] + \pi_{1,i}\, U[x_i, x_{i+1}] + \pi_{2,i}\, U[x_{i-1}, x_i, x_{i+1}] \tag{2.4}$$

Note that the π-functions are the same for both interpolants since divided differences are invariant if we change the order in which the x_i's are written, i.e.

$$U[x_{i-1}, x_i, x_{i+1}] = U[x_i, x_{i+1}, x_{i-1}].$$

So using the definitions above we use the same π-function for $U[x_{i-1}, x_i, x_{i+1}]$ (rewritten as $U[x_i, x_{i+1}, x_{i-1}]$) as we did for $U[x_i, x_{i+1}, x_{i+2}]$, since the π-function only depends on the first two gridpoints written in the difference.

To try to reduce the oscillations in our answer (as seen with polynomial interpolation) we choose the interpolant with the smallest divided difference, i.e. if

$$|U[x_{i-1}, x_i, x_{i+1}]| < |U[x_i, x_{i+1}, x_{i+2}]|$$

then we use (3.3), and vice-versa.

Unfortunately, when using evenly spaced meshpoints this type of interpolation does not guarantee a data bounded interpolant.

Definition 2.1 *An interpolating polynomial $U^L(x)$ is **data bounded** on the interval $[x_i, x_{i+1}]$ if:*

$$\begin{aligned} U^L(x_i) &= U(x_i) \\ U^L(x_{i+1}) &= U(x_{i+1}) \\ min(U(x_i), U(x_{i+1})) \leq\ & U^L(x) \leq max(U(x_i), U(x_{i+1})) \qquad x \in [x_i, x_{i+1}] \end{aligned}$$

We would prefer a data bounded interpolant since a motivation for this work comes from fluid dynamics, where positive and data bounded solutions are required for physical quantities. We would also prefer an interpolant that is monotonic on each local interval, since this would eliminate the oscillations in our semi-lagrangian solution.

2.6.1 Limited Form of the Divided Difference Interpolating Polynomial

So far we have seen how we can choose the smallest divided difference at each stage to reduce the oscillations in our solution. Alas this method can lead to higher order differences being much larger than differences used in previous stages. Berzins [1] refers to a paper by Harten (1995) [2] who said that large jumps in the size of consecutive differences can lead to a poor approximation when using the above method.

The problem arises when forming divided differences using two lower order differences of opposite sign.

Consider the divided difference

$$U[x_{i-1}, x_i, x_{i+1}] = \frac{U[x_i, x_{i+1}] - U[x_{i-1}, x_i]}{(x_{i+1} - x_{i-1})}$$

and suppose that $U[x_{i-1}, x_i] = -\epsilon U[x_i, x_{i+1}]$ where ϵ is some positive constant.

Rewriting the above we get

$$U[x_{i-1}, x_i, x_{i+1}] = (1 + \epsilon)\frac{U[x_i, x_{i+1}]}{x_{i+1} - x_{i-1}}$$

and if $(x_{i+1} - x_{i-1}) < 1$ then

$$\frac{1 + \epsilon}{x_{i+1} - x_{i-1}}$$

is an amplification factor and so we have

$$U[x_{i-1}, x_i, x_{i+1}] > U[x_i, x_{i+1}].$$

This shows how using a higher order interpolating polynomial can produce jumps in the sizes of the differences used which in turn gives a poorer approximation.

To solve this problem Berzins suggests that we form an interpolant on each interval within the data set and set any divided differences formed using differences of opposite signs to zero.

For example suppose we decided to use equation (2.3) to form a quadratic interpolant on the interval $[x_i, x_{i+1}]$

$$U^L(x) = U[x_i] + \pi_{1,i}\, U[x_i, x_{i+1}] + \pi_{2,i}\, U[x_i, x_{i+1}, x_{i+2}].$$

We know that $U[x_i, x_{i+1}, x_{i+2}]$ is formed using $U[x_i, x_{i+1}]$ and $U[x_{i+1}, x_{i+2}]$

$$U[x_i, x_{i+1}, x_{i+2}] = \frac{U[x_i, x_{i+1}] - U[x_{i+1}, x_{i+2}]}{x_i - x_{i+2}}$$

and so if $U[x_i, x_{i+1}]$ and $U[x_{i+1}, x_{i+2}]$ are of opposite sign we set $U[x_i, x_{i+1}, x_{i+2}]$ to zero.

This leaves us with the linear interpolant

$$U^L(x) = U[x_i] + \pi_{1,i}(x)\ U[x_i, x_{i+1}].$$

Note how we have locally altered the degree of the interpolating polynomial in order to reduce the error in our solution.
The new interpolant is referred to as the **limited form** of the divided difference interpolating polynomial.
This new approach gives a data bounded monotone interpolant for the cases which we shall consider.

Chapter 3

Results

3.1 Results for Runge's function

3.1.1 Runge's function with polynomial interpolation

A famous example from numerical analysis involves Runge's function

$$f(x) = \frac{1}{1 + 25x^2}. \qquad x \in [-1, 1]$$

In 1901 Carl David Tolm Runge found that accuracy is lost when approximating this function using high-order polynomial interpolation with evenly spaced data points (see section 2.2). In fact convergence is never observed as the order of the polynomial is increased.

This is shown in figure(3.1), which confirms Runge's findings.

In the first plot ($n = 4$ or 3^{rd} order polynomial) we have a smooth, albeit inaccurate interpolant. Doubling the number of data points to 8 produces some oscillation in the solution and by the time we have increased the order of the interpolating polynomial to 15 ($n = 16$) in the final plot we see marked oscillations in the solutions, reaching roughly twice the height of the original function.
So the approximating polynomial does not actually converge as the degree tends to infinity.

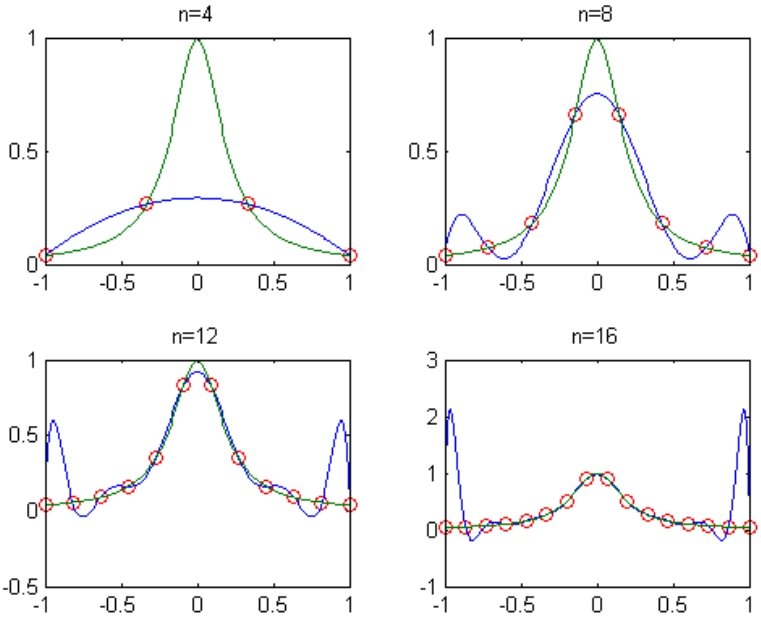

Figure 3.1: Polynomial interpolation (blue) plotted against Runge's function (green) and data points (red) with increasing order. n indicates the number of data-points used.

3.1.2 Runge's function with standard divided difference interpolation

We now investigate how the standard form of the divided difference interpolation (section 2.6) behaves when we use it to interpolate Runge's function. We can write a Matlab program to emulate the results on the standard form found in Berzin's paper [1].
We use 7 evenly spaced data points on the interval $[-1, 1]$. The graph is shown in figure(3.2).

As we can see from the oscillations in the graph the standard form of the divided difference interpolant does not provide an accurate approximation to Runge's function when using 7 data points.

To show how the standard form behaves as the number of data points used

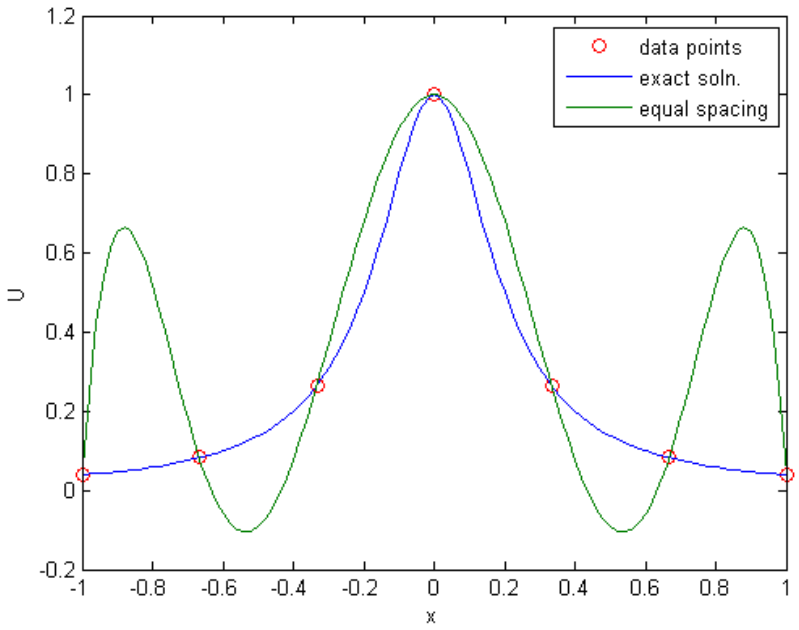

Figure 3.2: Standard divided difference interpolation using 7 evenly spaced data points (green) plotted with Runge's function (blue) and data points (red).

is increased, we recreate the above plot using 3,5,7 and 9 data points. The four graphs are shown in figure(3.3).

We can see that the standard form of the divided difference interpolant gives wilder oscillations (and thus a poorer approximation) when the number of data points used is increased, as we saw was the case for polynomial interpolation (see previous subsection). These results confirm the theory on the standard form seen in section (2.6), where we saw how using higher order divided differences (i.e. more data points) can produce large errors in the solution, due to divided differences being formed using lower order differences of opposite sign.

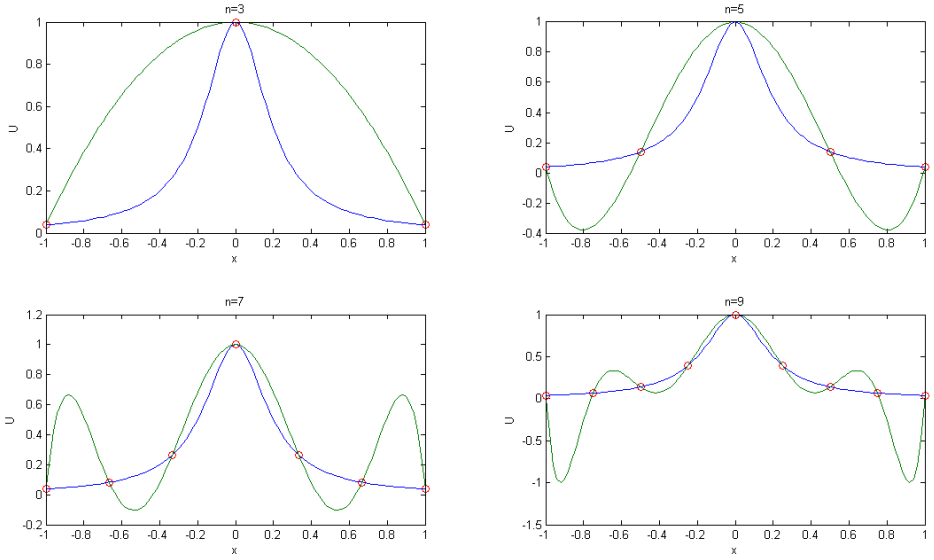

Figure 3.3: Runge's function (blue) plotted with the standard divided difference interpolant (green) for increasing number of data points (red).

3.1.3 Runge's function with the limited form of divided difference interpolation

We can now show that the limited form of the divided difference interpolant can produce a better approximation to Runge's function than the methods discussed so far in this chapter. The graphs in figure(3.4) show Runge's function plotted against the limited form of the divided difference interpolant using an increasing number of data points.

From the graphs we see that using the limited form of the divided difference interpolant gives a drastic improvement in our approximation to Runge's function when compared to the methods seen previously in the chapter. We no longer see large oscillations in our solution, and increasing the number of data points seems to improve the accuracy of the interpolant as opposed to increasing the error. We can see that the interpolants are data bounded (definition 2.1), i.e. the value of each interpolant does not overshoot the data points on each local interval. Also each interpolant is monotonic on each interval.

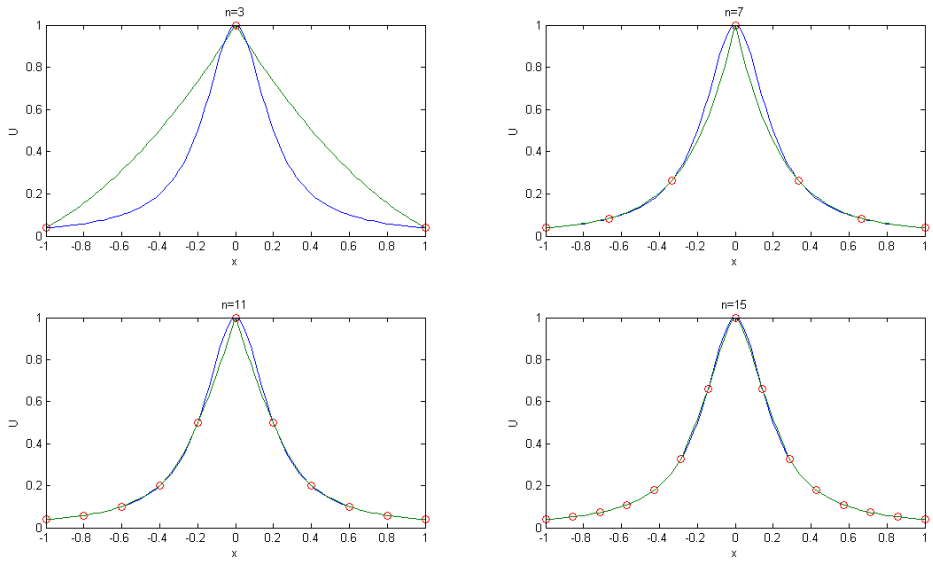

Figure 3.4: Runge's function (blue) plotted with the limited form interpolant (green) and data points (red) with increasing n (number of data points).

The results show that using Harten's approach (subsection (2.6.1)) and throwing away any divided differences formed using differences of opposite signs in the interpolating polynomial reduces the oscillations and thus improves the accuracy of our approximation.

3.2 Results for the Semi-Lagrangian scheme

We now compare the limited form of divided difference interpolation with two other interpolation methods, standard piecewise cubic and pchip (Hermite cubic interpolation with nodal slopes calculated by harmonic means), when used with a semi-lagrangian scheme.

We shall use the 1-d Advection Equation

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0 \qquad \text{with } u(0) = u_0$$

(see subsection (1.1.1)) to model a square wave travelling to the right in the region $x = [-10, 10]$ with periodic boundary conditions.

We choose u_0 to be a square wave with height 10 and width 2, centred at the origin. So our initial condition is

$$u = \begin{cases} 10 & -1 \leq x \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$

A square wave is notoriously difficult to interpolate because of the discontinuities that occur, in our case at $x = -1, 1$.

To begin we choose a time step $dt = 1$, spatial step $dx = 0.2$ and velocity $a = 0.7$, and run the scheme for 999 time steps. The initial plot and final plot for the three types of interpolation are shown in figures (3.5),(3.6) and (3.7).

Figure (3.5) shows how standard piecewise cubic interpolation behaves in this instance. On the final plot we see oscillations either side of the wave where the interpolant 'undershoots' the x-axis. Also the wave has a rounded peak (it is no longer square) that reaches above 10. Despite this, the width of the wave at the final time is well-preserved.

Figure (3.6) shows pchip interpolation. Again, the wave has become 'smoothed' at the final time. The pchip interpolant has no oscillations at the final time, but the peak of the wave has fallen from 10 to roughly 8.

In figure (3.7) we see the limited form of the divided difference interpolant at the initial and final times. Again, the interpolant has failed to capture the discontinuities in the wave. We have no oscillations, but the peak of the interpolant has fallen well below 10 to less than than half its original height. The wave has also become spread out over an area roughly five times larger than for the initial square wave. This is due to the fact that we are using high order divided differences from an area wider than the wave profile to form the interpolant, and since the function is zero at points outside the profile we get a 'damping' effect in our solution.

We can conclude that the best type of non-oscillatory interpolation to use for the set of parameters chosen is pchip interpolation, since it has no oscillations and captures the square wave more accurately than the limited form of

the the divided difference interpolant. The worst type interpolation from a non-oscillatory point of view in this instance is piecewise cubic interpolation because of the oscillations that arise at the final time.

We now increase the number of iterations to 9999 and observe the effect this has on the interpolants. The graphs are shown in figures (3.8),(3.9) and (3.10).

Running the scheme for ten times longer confirms what we saw above. Although each interpolant has lost height and spread out over a wider area after 9999 time steps, we still see that pchip gives the best non-oscillatory approximation to the square wave. Piecewise cubic interpolation still produces worse oscillations at the final time. The pchip interpolant has lost considerable height at the the final time, but not as much as the limited form divided difference interpolant which has spread out most and whose peak does not rise above 2.

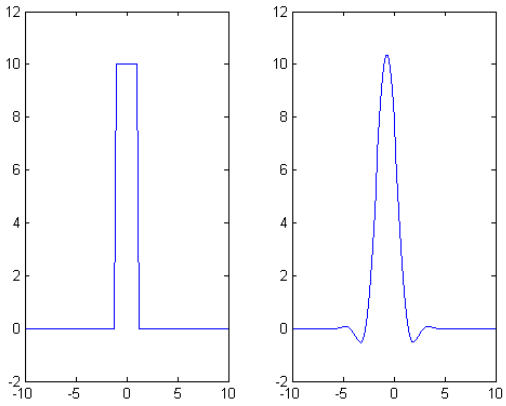

Figure 3.5: Standard piecewise cubic piecewise interpolation (dt=1, a=0.7, run for 999 time steps).

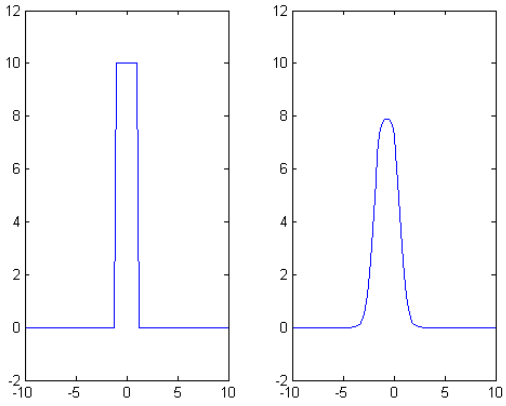

Figure 3.6: pchip interpolation (dt=1, a=0.7, run for 999 time steps).

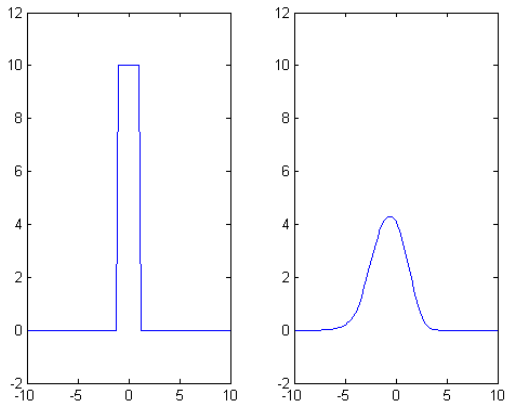

Figure 3.7: Limited form of divided difference interpolation (dt=1, a=0.7, run for 999 time steps).

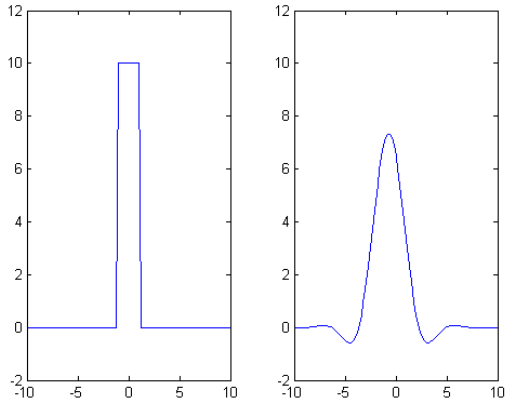

Figure 3.8: Standard piecewise cubic piecewise interpolation (dt=1, a=0.7, run for 9999 time steps).

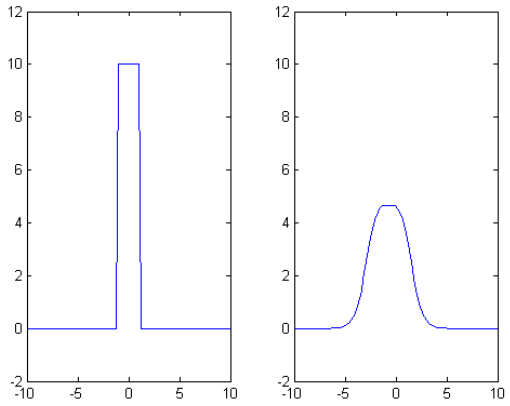

Figure 3.9: pchip interpolation (dt=1, a=0.7, run for 9999 time steps).

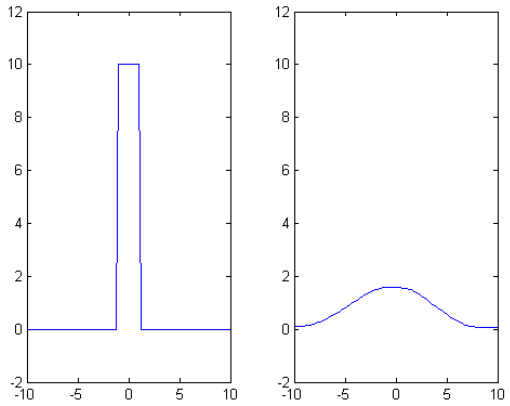

Figure 3.10: Limited form of divided difference interpolation (dt=1, a=0.7, run for 9999 time steps).

Chapter 4

A non-linear equation

4.1 The inviscid form of Burgers' equation

4.1.1 Exact Solution

ENO (Essentially non-Oscillatory) schemes were originally developed for use with non-linear equations. One such equation is the inviscid form of Burgers' equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0 \qquad u(x,0) = u_0(x). \tag{4.1}$$

We shall use the initial condition found on page 77 of Smith [4]

$$u_0 = \begin{cases} cos^2\left[\frac{\pi}{2}\left(\frac{x-3}{2}\right)\right] & |x-3| \leq 2 \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

The characteristic curves of 4.1 are given by

$$\frac{dx}{dt} = u(x(t),t) \tag{4.3}$$

along which we know that the solution u is constant.

Hence the solution $u(x(t),t)$ will be the same as the solution at x_0, where x_0 is the point where the characteristic that passes through $u(x(t),t)$ arrives at $t = 0$. So we may rewrite 4.3 as

$$\frac{dx}{dt} = u(x_0,0) = u_0(x_0).$$

Integrating we get

$$x = u_0(x_0)t + x_0 \tag{4.4}$$

So given any point (t_{n+1}, x_a) in (t, x) space we can obtain the solution u at this point by solving 4.4 for x_0.

To do this we rewrite 4.4 as

$$F(x_0) = x - u_0(x_0)t - x_0$$

and use the Newton-Rhaphson method:

$$x_0^{new} = x_0^{old} - \frac{F(x_0)}{F'(x_0)}$$

with initial guess x_0=x_a.
This method should converge fairly quickly to give us the original value of x_0.

Figure(4.1) shows the exact solution to the above problem at times $t = 0$ and $t = 1$.

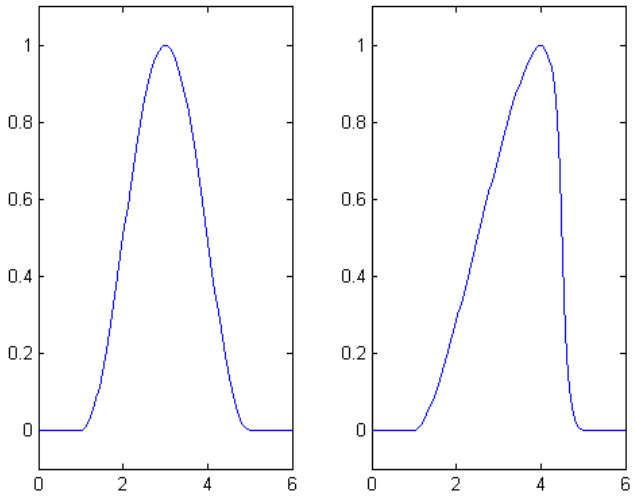

Figure 4.1: The exact solution to equation 4.1 at times $t = 0$ and $t = 1$

Notice the parallels between the above method the the semi-lagrangian scheme seen in previous chapters. In both cases, we travel back along the characteristic curves (in the above case to the start time $t = 0$ as opposed to the

previous timestep as with the semi-lagrangian scheme) and use the fact that the solution u is constant along these characteristics to obtain a solution at the new time.

The method we have used in this subsection for finding the exact solution is very useful for simple problems such as the inviscid Burgers' equation, but we will run into difficulties if we attempt to use it for more complicated equations. We would then have to use the numerical semi-lagrangian scheme.

4.1.2 The semi-lagrangian scheme with the inviscid form of Burgers' equation

We now solve the inviscid form of Burgers' equation (4.1) with initial condition (4.2) using the semi-lagrangian scheme.

As with the 1-D advection equation where $u = a$ (1.1.1) we assume that we know the solution u at time $t = t_n$.

Suppose we want the solution at some point (x_a, t_{n+1}), which we shall call the arrival point. We follow the characteristic curve that passes through (x_a, t_{n+1}) back to time $t = t_n$ and call this point x_d (the departure point). Using the fact that the solution is constant along the characteristic curves we conclude that the solution at the two points must be equal, i.e.

$$u(x_a, t_{n+1}) = u(x_d, t_n).$$

So all we need to find $u(x_a, t_{n+1})$ is x_d.

We now run into difficulties. Since our velocity $a = u(x, t)$ is a function of x and t we do not know the exact form of the characteristic curve that passes through (x_a, t_{n+1}). The usual procedure is to approximate the characteristic curve with a straight line by extrapolating the solution at time $t = t_{n+\frac{1}{2}}$ using the solutions at times $t = t_n$ and $t = t_{n-1}$.
This is done by means of the iterative midpoint method.

We wish to find α where $\alpha = x_a - x_d$. The midpoint method proceeds as follows.

$$\alpha^{(0)} = 0 \qquad \text{initial guess}$$

$$\begin{aligned} \alpha^{(k+1)} &= \Delta t u^*(x_a - \frac{\alpha^{(k)}}{2}, t_n + \frac{\Delta t}{2}) \\ \text{with} \quad u^*(x_i, t_n + \frac{\Delta t}{2}) &= \frac{3}{2}u(x_i, t_n) - \frac{1}{2}u(x_i, t_n - \Delta t) \end{aligned}$$

We use two to three iterations of the midpoint method to give us a value for α, from which we can find x_d (since $x_d = x_a - \alpha$).

If x_d lies on a grid point we have the solution immediately since $u(x_a, t_{n+1}) = u(x_d, t_n)$. If x_d does not coincide with a gridpoint, then we use an interpolation method to approximate $u(x_d, t_n)$ using local nodal values at time $t = t_n$ which gives us $u(x_a, t_{n+1})$.

4.1.3 Results using the midpoint method

We use the semi-lagrangian scheme with the midpoint method to solve the inviscid form of Burgers' equation (4.1) using initial condition (4.2) and parameters $\Delta t = 0.005$, $\Delta x = \frac{1}{15}$. We run the scheme for 120 timesteps, ensuring that the profile does not form a shock before our final time ($t = 0.6$).

Figure (4.2) shows the exact solution plotted at the final time along with the results for the semi-lagrangian scheme using three different interpolation methods. *Limited Form* denotes the limited form of the divided difference interpolant (see (2.6.1)), *pchip* denotes piecewise cubic Hermite interpolation with derivatives at the nodes approximated by harmonic means (see (2.5)) and *Polyinterp* denotes a piecewise cubic polynomial interpolation (see (2.2)).

In figure (4.2) we see that the plots for the semi-lagrangian solution lie slightly to the right of the exact solution. This means that the semi-lagrangian solution is slightly 'ahead' of the true solution i.e. it approximates the wave to be travelling slightly faster than it actually is. The plots show no 'undershoots', indeed we have no oscillations using any of the interpolation methods. This is probably due to the fact that the function which we are modelling is much 'smoother' than the square wave profile seen in previous chapters. The only difference between the interpolation methods seems to be that the limited form of the divided difference interpolant produces a much steeper slope at the edges of the wave, and that the pchip interpolant forms a slight 'hump' at the peak of the wave. We can therefore conclude that the best interpolation method to use in this case is standard piecewise cubic polynomial interpolation.

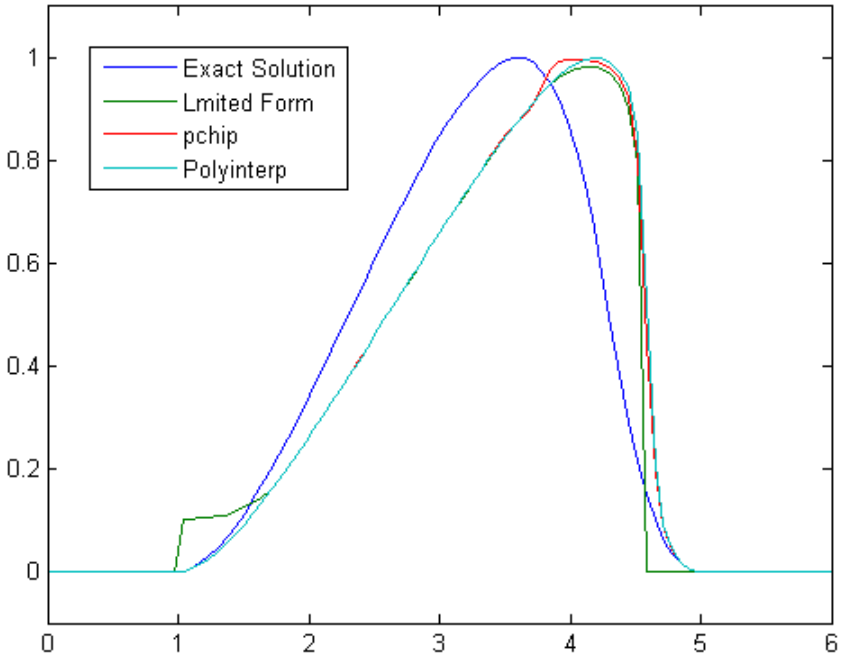

Figure 4.2: The exact solution to (4.1) at $t = 0.6$ plotted with the semi-lagrangian solution using the midpoint method and three different interpolation methods

4.1.4 The Shu-Osher Runge-Kutta method

As well as using polynomials with least variation to interpolate given data values, ENO schemes make extensive use of the 3^{rd} order Shu-Osher Runge-Kutta method to increase their order of accuracy.

We can adopt this ENO approach for use with our semi-lagrangian scheme when solving the inviscid form of Burgers' equation (4.1) with initial condition 4.2. As usual we suppose we know the solution u at time $t = t_n$, and that we wish to find u at a certain point x_a at time $t = t_{n+1}$.

As opposed to using the midpoint method (see (4.1.2) and (4.1.3) for calculation of our departure point x_d at time $t = t_n$, we can use a slightly modified version of the Shu-Osher method.

Suppose, as an initial guess, we take our departure point x_d to be the x-coordinate of the arrival point x_a. The Shu-Osher method then proceeds as follows.

$$\begin{aligned} x_d^{(1)} &= x_a - \Delta t\ u(x_a, t_n) \\ x_d^{(2)} &= \frac{3}{4}x_a + \frac{1}{4}x_d^{(1)} - \frac{1}{4}\Delta t\ u(x_d^{(1)}, t_n) \\ x_d^{(3)} &= \frac{1}{3}x_a + \frac{2}{3}x_d^{(2)} - \frac{2}{3}\Delta t\ u(x_d^{(2)}, t_n) \\ x_d &= x_d^{(3)} \end{aligned}$$

Note that since $x_d^{(1)}$ and $x_d^{(2)}$ will not necessarily lie on gridpoints we will have to use an interpolation method to approximate the values of $u(x_d^{(1)}, t_n)$ and $u(x_d^{(2)}, t_n)$. In our case we shall use cubic polynomial interpolation.

Figure (4.3) shows the exact solution plotted at the final time along with our results for the semi-lagrangian scheme using three different interpolation methods (limited form of the divided difference interpolant, pchip and cubic polynomial interpolation). We have used the same parameters as when using the midpoint method for calculating the departure point, namely $\Delta t = 0.005$, $\Delta x = \frac{1}{15}$ and a final time of 0.6.

We can see a marked difference between the results for the midpoint method and the Shu-Osher Runge-Kutta method. When using the midpoint method the semi-lagrangian solutions seemed to be travelling faster than the exact solution. With the Shu-Osher method the semi-lagrangian solutions are closer to the exact solution at the final time, and as they lie slightly to the left of the exact solution it seems that they are travelling slower than the exact solution. Since these solutions are closer to the exact solution than when using the midpoint method we conclude that, in this case, using the Shu-Osher Runge-Kutta method as opposed to the midpoint method for the calculation of the departure point x_d improves our solution.

As with the midpoint method, the limited form of the divided difference interpolant forms very steep gradients at the edges of the wave profile. These might be to do with the fact that we are throwing away divided differences formed using differences of opposite signs. Near an extremal point (such as the one in the function which we are approximating) this might cause many differences to be thrown away, resulting in a linear interpolant. More is said on this point in the section on further work.

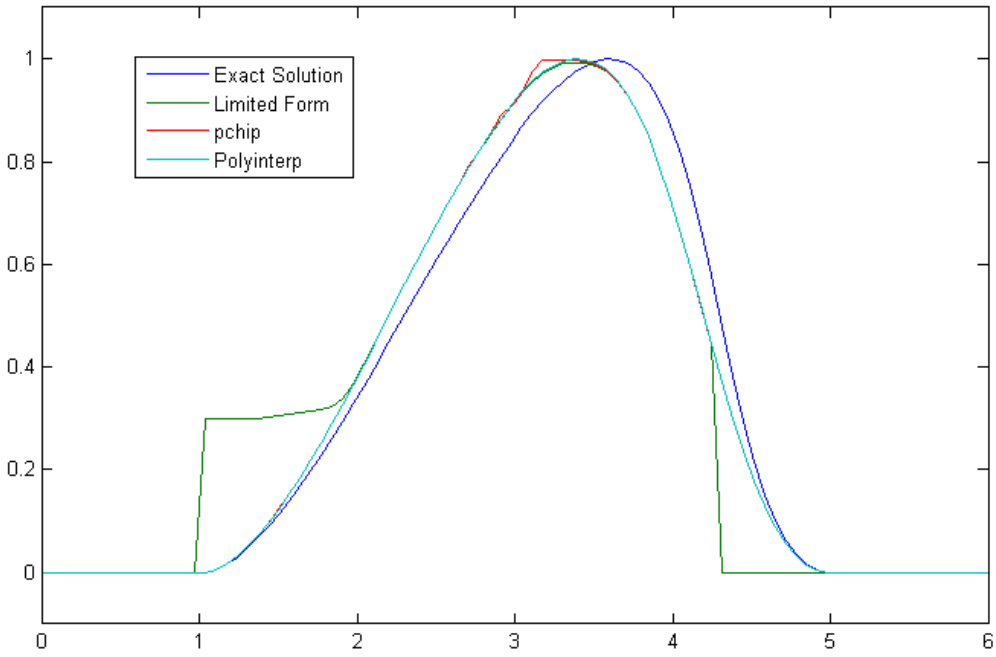

Figure 4.3: The exact solution to (4.1) at $t = 0.6$ plotted with the semi-lagrangian solution using the Shu-Osher Runge-Kutta method and three different interpolation methods

Again we see that pchip produces a slight 'hump' near the peak of the wave at the final time. Therefore we must again conclude that of the three interpolation methods used, standard piecewise cubic polynomial interpolation seems best in this instance.

4.1.5 Changing the initial condition

So far in this chapter, we have been solving equation (4.1) using initial condition (4.2) given by Smith [4]. Since this initial condition has a fairly smooth profile, we saw no oscillations in the semi-lagrangian solutions at the final time using our various interpolation methods.

Changing the initial condition to a function that has discontinuities might

give some oscillations in the semi-lagrangian solutions, and may give us a better way to distinguish which interpolation method and which method for finding the departure point works best in our case.

We therefore change our initial condition to a simple 'step function'

$$u_0 = \begin{cases} 1 & x < 0 \\ 2 & x \geq 0 \end{cases} \tag{4.5}$$

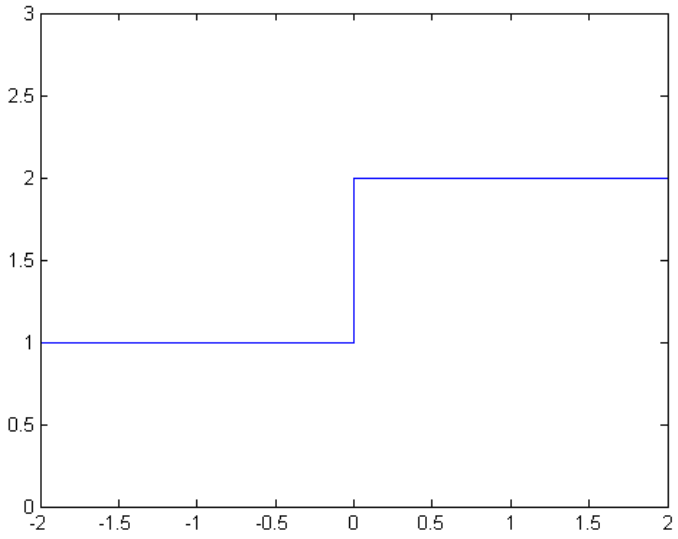

Figure 4.4: Plot of the step function used as initial condition (4.5)

We shall examine how each interpolation method fares when solving the inviscid form of Burgers' equation (4.1) with this new initial condition along with the two different methods for obtaining the departure point at each gridpoint, the midpoint method and the Shu-Osher Runge-Kutta method.

Figure (4.5) shows the exact solution at the final time $t = 0.5$ along with the semi-lagrangian solutions for the three interpolation methods using the midpoint method for calculation of the departure point. Figure (4.6) shows the same plots when we use the Shu-Osher Runge-Kutta methods for calculating the midpoint. The parameters used are $\Delta t = 0.005$, $\Delta x = \frac{1}{15}$. *Limited Form* denotes the limited form of the divided difference interpolant (see (2.6.1)),

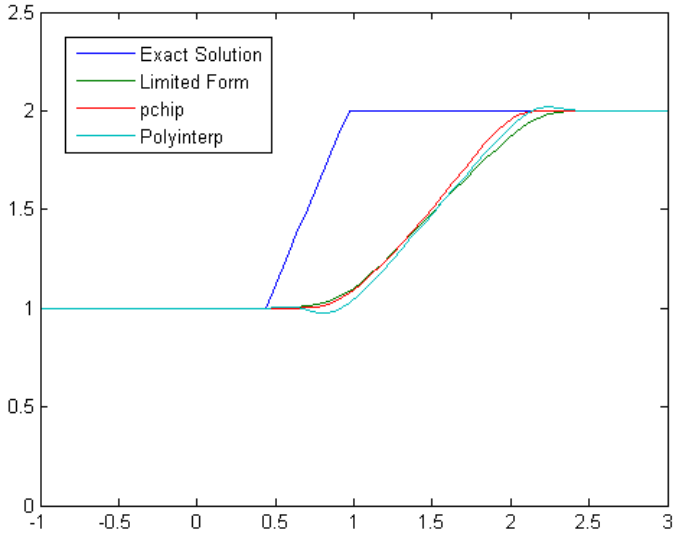

Figure 4.5: Solving equation (4.1) with initial condition (4.5) using the semi-lagrangian scheme with various interpolation methods using the midpoint method for calculation of the departure point.

pchip denotes piecewise cubic Hermite interpolation with derivatives at the nodes approximated by harmonic means (see (2.5)) and *Polyinterp* denotes a piecewise cubic polynomial interpolation (see (2.2)).

From the figures we that for both methods of departure point calculation, piecewise cubic polynomial interpolation produces 'undershoots' in the semi-lagrangian solution. These oscillations are not present when using pchip or the limited form of the divided difference interpolation due to the various techniques that these interpolation methods use to eliminate variations in the solution. From a non-oscillatory perspective this discredits piecewise cubic polynomial interpolation, even though it performed best when used to model a smooth function (see subsections (4.1.3) and (4.1.4)). There is no discernible difference between the divided difference interpolation and pchip in the first figure, although in figure (4.6) the divided difference interpolant seems closer to the exact solution at the final time.

As seen when modelling the smooth function in the previous two subsections, when we use the midpoint method for calculation of the departure point the

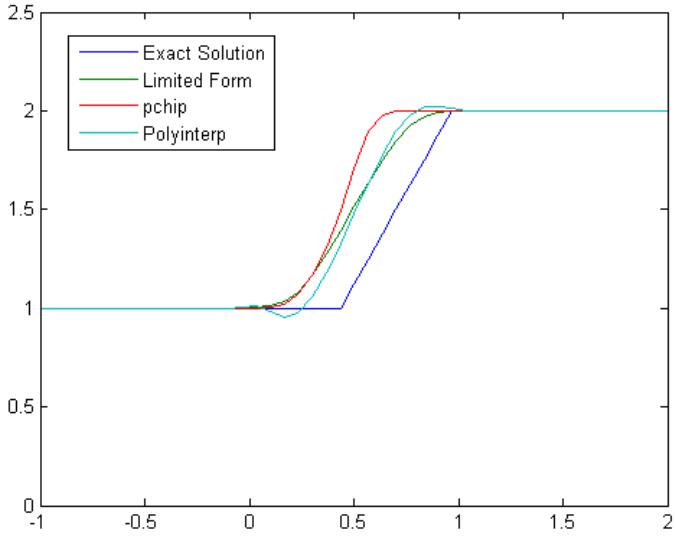

Figure 4.6: Solving equation (4.1) with initial condition (4.5) using the semi-lagrangian scheme with various interpolation methods using the Shu-Osher Runge-Kutta method for calculation of the departure point.

semi-lagrangian solutions appear to be travelling ahead of the exact solution, while when the Shu-Osher Runge-Kutta method is used the semi-lagrangian solutions are seen to be trailing the exact solution. Again the solutions for the Shu-Osher method appear much closer to the exact solution at the final time than those for the midpoint method, from which we conclude that the Shu-Osher method is better for this purpose.

Summary and further work

Summary

In this dissertation we have compared various interpolation methods for use with the semi-lagrangian scheme.

Chapter one gave a brief introduction to the semi-lagrangian scheme and some motivation for the work.

Chapter two introduced some well known interpolation methods, along with a new interpolation method suggested by Berzins [1], namely the limited form of divided difference interpolation. We also explained how the motivation for this work comes from problems in the physical world, demanding solutions that are positive and data-bounded. This meant that that we would require little or no oscillations in our results, and from that point would compare interpolation methods from a non-oscillatory viewpoint.

In chapter three we saw how Berzins' interpolation outperformed standard piecewise cubic interpolation when approximating Runge's function on a standard mesh. We also looked at some results for the semi-lagrangian scheme. Again Berzins' interpolation proved to be more useful for our purpose than standard piecewise cubic polynomial interpolation because of its non-oscillatory nature, but it did not seem to be as good as pchip, a special form of piecewise cubic Hermite interpolation.

Chapter four introduced a non-linear equation, the inviscid form of Burgers' equation. We saw how we would run into difficulties when using the semi-lagrangian scheme to model this equation since we would no longer know the departure points for each gridpoint. Therefore we looked at two methods for calculating the departure point and found that the Shu-Osher Runge-Kutta

method proved to be more useful for our case than the traditionally used midpoint method.
Finally we used the semi-lagrangian scheme in conjunction with the departure point calculation methods to model the non-linear equation. At first we used a smooth initial condition and found, rather surprisingly, that standard piecewise cubic polynomial interpolation behaved better than both Berzins' interpolation and pchip. We than changed the initial condition to a function that contained a discontinuity and found that standard piecewise cubic polynomial interpolation produced oscillations in the final solution and that Berzins' interpolation performed slightly better of the three.

Further work

An option for further work would be to investigate a more recent paper by Berzins [8], where he goes on to introduce a technique for dealing with extrema. He states that the limiting technique of keeping the divided difference interpolant bounded between the values at the endpoints of an interval could prove to be poor when the function has an extremal value in that interval. Berzins suggests a way of detecting these extrema and states that the limiting property should be turned off for an interval if an extrema is found to exist in that interval. It would be interesting from our perspective to see whether introducing this technique would improve the results for Berzins' interpolation, especially when modelling wave like functions such as Runge's function, since these functions have extremal points at their centres.

Bibliography

[1] M.Berzins: Adaptive Polynomial Interpolation on Evenly Spaced Meshes, *SIAM Review*, **49**, 2007, 604–627.

[2] A.Harten: Multiresolutional algorithms for the numerical solution of hyperbolic conservation laws, *Comm. Pure Appl. Math.*, **48**, 1995, 1304–1342.

[3] Andrew Staniforth and Jean Cote: Semi-Lagrangian Integration Schemes For Atmospheric Models - A Review, *Monthly Weather Review*, **119**, 1990, 2206–2223.

[4] Chris Smith: The Semi-Lagrangian Method in Atmospheric Modelling, *PhD Thesis*, **University of Reading**, 2000

[5] Amos S Lawless: Development of Linear Models for Data Assimilation in Numerical Weather Prediction, *PhD Thesis*, **University of Reading**, 2001

[6] Cleve Moler: *Numerical Computing with Matlab*, Society for Industrial and Applied Mathematics, 2004.

[7] Jan Hesthaven, David Gottlieb and Sigal Gottlieb: *Spectral methods for time-dependent problems*, Cambridge Monographs on Applied and Computational Mathematics, 2007.

[8] M.Berzins: Data Bounded Polynomials and Preserving Positivity in High Order ENO and WENO Methods, *SCI Report UUSCI*, 2009