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 di erence 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 di erence interpolation method o ers no improvement over existing methods when modelling a square wave with passive advection in 1-D.

In the nal 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 o er 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 nancial support while completing my masters.

Declaration

I con rm 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 concelme that
of1[(tes)-653(used)-682(in)-653(w)27(lat)1ther forecasting.Suprpuse we ishd to moded h(e)-653i
We ain(co)-27(use)-337(to)-38((sin)27(t)1ola(e)-38((the)-373i ow)27wd)-373uslingadietby(of)]TJ 0
comurtatiocalgrid,e whareeasclgridrpienthastasshodtuest(for)-03((certain)]TJ 0 -14.446 Td [v)564(aiac

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 di cult 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 xed 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 di erent 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

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

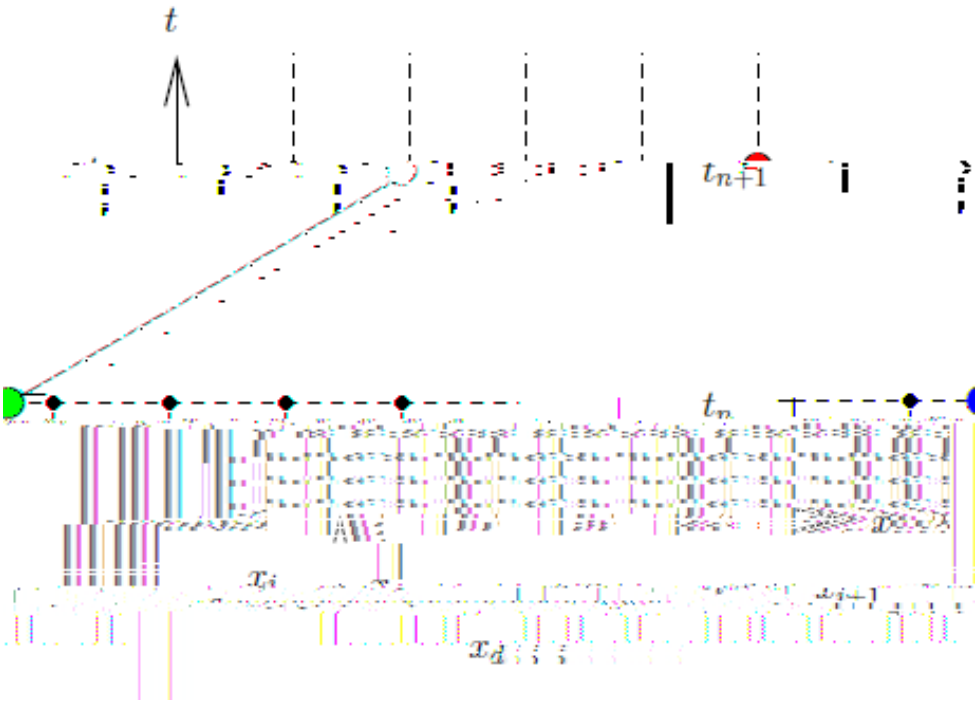


Figure 1.1: Diagram showing a brief outline of the semi-lagrangian scheme.

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 di erent 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 rst 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

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

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

$$y = 1 + (1 \quad 0)\frac{(6 \quad 1)}{(2 \quad 0)} = 3:5:$$

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

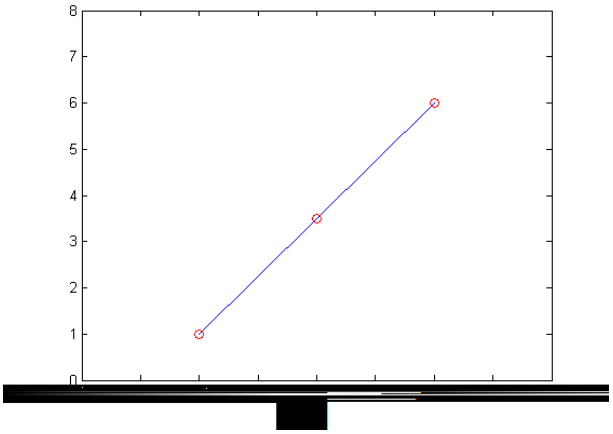


Figure 2.1: Linear interpolation between the points $(0;1)$ and $(2;6)$

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 de ned, $(x_k;y_k)$ $k = 1;:::;n$, there exists a unique polynomial, $P(x)$ (again called the interpolant), of order less than n that passes exactly through each point, i.e.

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

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

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

or

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

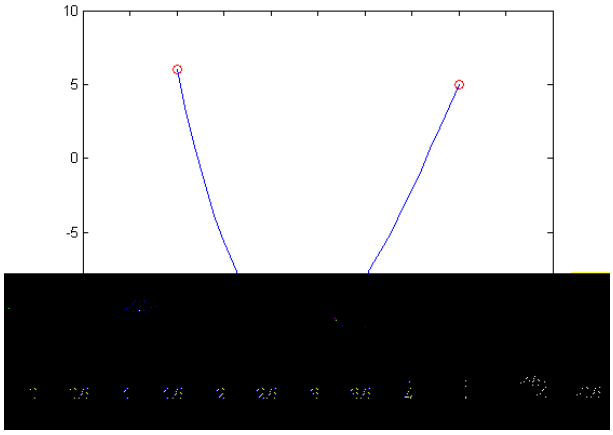


Figure 2.2: Polynomial interpolation using the data set $x = (0; 1; 2; 3)$; $y = (6; \ 11; \ 8; 5)$

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

$$P(x) = \frac{(x \ \ 1)(x \ \ 2)(x \ \ 3)}{(\ \ 6)}(6) + \frac{x(x \ \ 2)(x \ \ 3)}{(2)}(\ \ 11)$$
$$+ \frac{x(x \ \ 1)(x \ \ 3)}{(\ \ 2)}(\ \ 8) + \frac{x(x \ \ 1)(x \ \ 2)}{(6)}(5)$$

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

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

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

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

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 di culties 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 $(x_k; y_k)$ $k = 1; : : : ; n$ we perform linear interpolation between each point within the set. To interpolate the value of y at any intervening point x, we must rstly locate it within our data set, i.e. nd k where

$$x_k \quad x < x_{k+1}:$$

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

$$s = x \quad x_k$$

and then the rst divided di erence

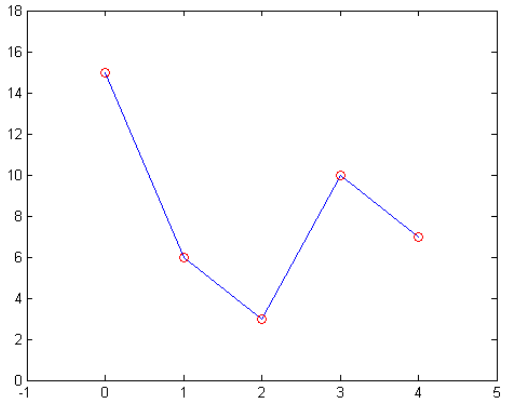
$$_k = \frac{y_{k+1} \quad y_k}{x_{k+1} \quad x_k}$$

Note that s and $_k$ 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

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

which gives a straight line that82nua$x$


18
16
14
12
10
8
6
4
2
0
-1
0
1
2
3
4
5

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

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

where

$$h = x_{k+1} \quad x_k$$
$$_k = \underline{y_{k+1} \quad y_k}$$

`harmonic mean' of the two di erences $_k$ and $_{k-1}$

$$\frac{1}{d_k} = \frac{1}{2}\left(\frac{1}{_{k+1}} + \frac{1}{_k}\right)$$

If $_k$ and $_{k-1}$ have di erent 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 $_k$ and $_{k-1}$ have the same sign but the intervals are not of the same lengths, then d_k is given by the `weighted harmonic mean'

$$\frac{w_1 + w_2}{d_k} = \frac{w_1}{_{k-1}} + \frac{w_2}{_k}$$

with weights

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

where

$$h_k = x_{k+1} \quad x_k; \qquad h_{k-1} = x_k \quad x_{k-1}:$$

A di erent non-centred shape preserving 3-point formula is used to de ne the derivatives at the endpoints $(x_1; y_1)$ and $(x_n; y_n)$.

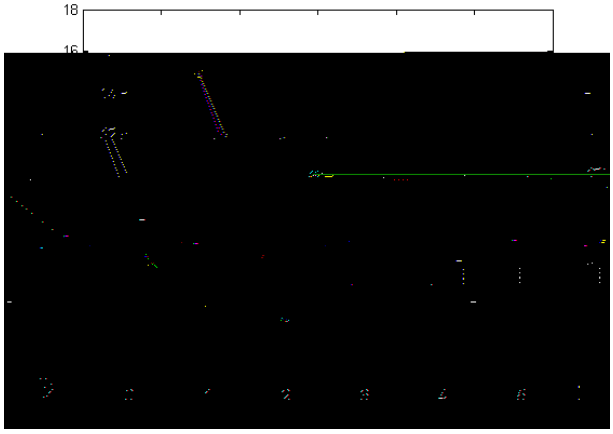


Figure 2.4: pchip (blue) and piecewise linear interpolation (green) using the data set $x = (0; 1; 2; 3; 4)$; $y = (15; 6; 3; 10; 7)$

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 $(x_i; y_i); \quad i = 0; : : : n$, 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:

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

with higher order differences given by

$$U[x_i; x_{i+1}; : : : ; x_{i+k}] = \frac{U[x_{i+1}; x_{i+2}; : : : ; x_{i+k}] \quad U[x_i; x_{i+1}; : : : ; x_{i+k-1}]}{x_{i+k} \quad x_i}$$

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

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

It is often easier to picture divided di erences if we form a di erence table

$$
\begin{array}{llllll}
x_0 & U[x_0] & & & & \\
 & & U[x_0; x_1] & & & \\
x_1 & U[x_1] & & U[x_0; x_1; x_2] & & \\
 & & U[x_1; x_2] & & U[x_0; x_1; x_2; x] & U
\end{array}
$$

So using the de nitions above we use the same -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 -function only depends on the rst two gridpoints written in the di erence.

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

$$jU[x_{i-1}; x_i; x_{i+1}]j < jU[x_i; x_{i+1}; x_{i+2}]j$$

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.

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

$$\begin{aligned} U^L(x_i) &= U(x_i) \\ U^L(x_{i+1}) &= U(x_{i+1}) \\ min(U(x_i); U(x_{i+1})) \quad U^L(x) \quad max(U(x_i); U(x_{i+1})) &\qquad x \ 2 \ [x_i; x_{i+1}] \end{aligned}$$

We would prefer a data bounded interpolant since a motivation for this work comes from uid 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 Di erence Interpolating Polynomial

So far we have seen ho

Consider the divided di erence

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

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

Rewriting the above we get

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

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

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

is an ampli cation 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 di erences 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 di erences 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 $[x_i; x_{i+1}]$

$$U^L(x) = U[x_i] + \ _{1;i}\ U[x_i; x_{i+1}] + \ _{2;i}\ U[x_i; x_{i+1}; x_{i+2}]:$$

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

$$U[x_i; x_{i+1}; x_{i+2}] = \frac{U[x_i; x_{i+1}] \quad U[x_{i+1}; x_{i+2}]}{x_i \quad x_{i+2}}$$

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.

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

$$f(x) = \frac{1}{1 + 25x^2}: \qquad x \; 2 \; [\;\; 1; 1]$$

In 1901 Carl David Tolm Runge found that accuracy is lost when approxi-

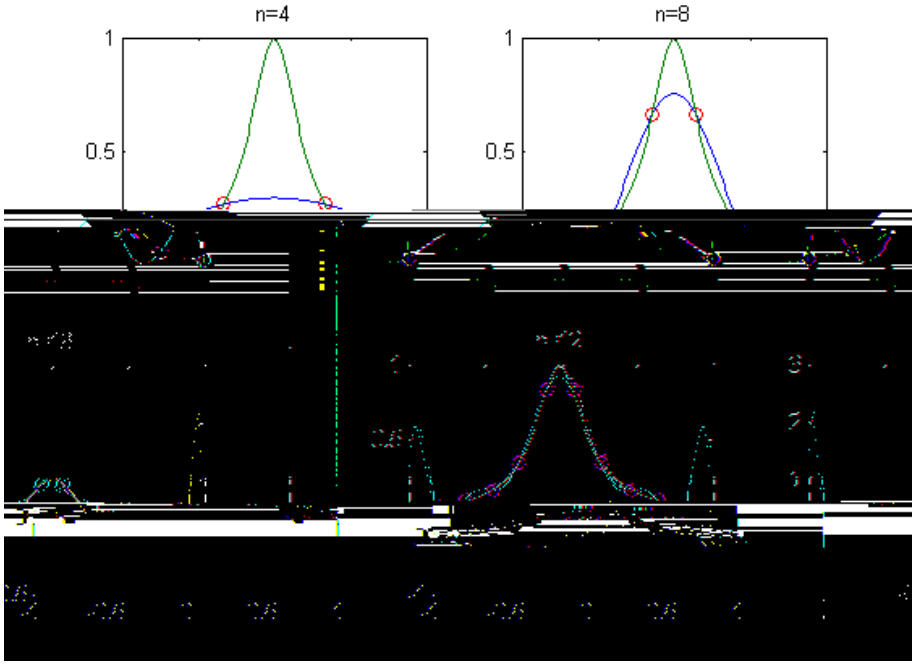


Figure 3.1: Polynomial interpolation (blue) plotted against Runge's function (green) and data points (red) with increasing order. *n* indicates the number of data-points used.

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

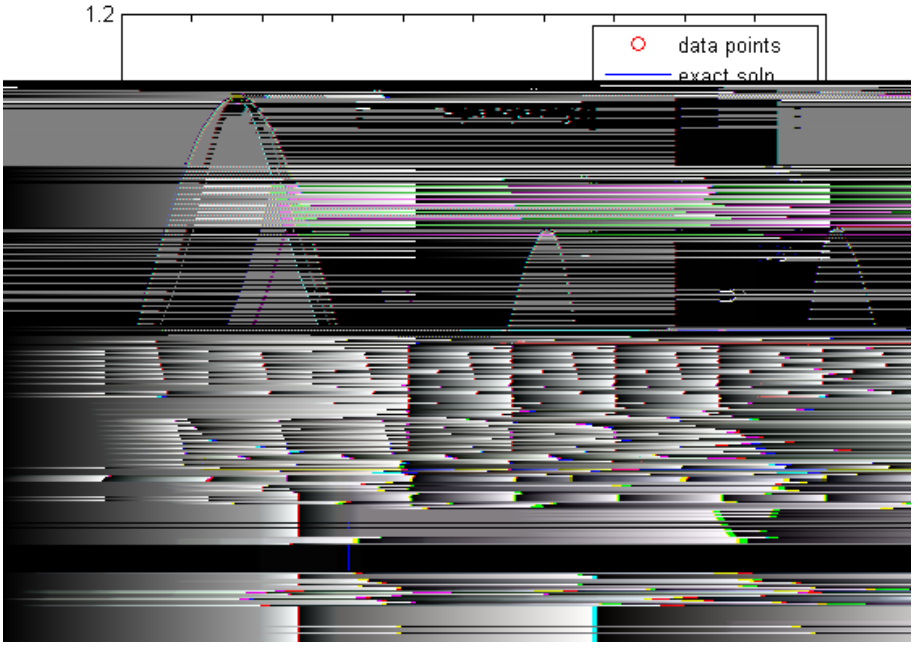


Figure 3.2: Standard divided di erence interpolation using 7 evenly spaced data points (green) plotted with Runge's function (blue) and data points (red).

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

We can see that the standard form of the divided di erence 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 con rm the theory on the standard form seen in section (2.6), where we saw how using higher order divided di erences (i.e. more data points) can produce large errors in the solution, due to divided di erences being formed using lower order di erences of opposite sign.

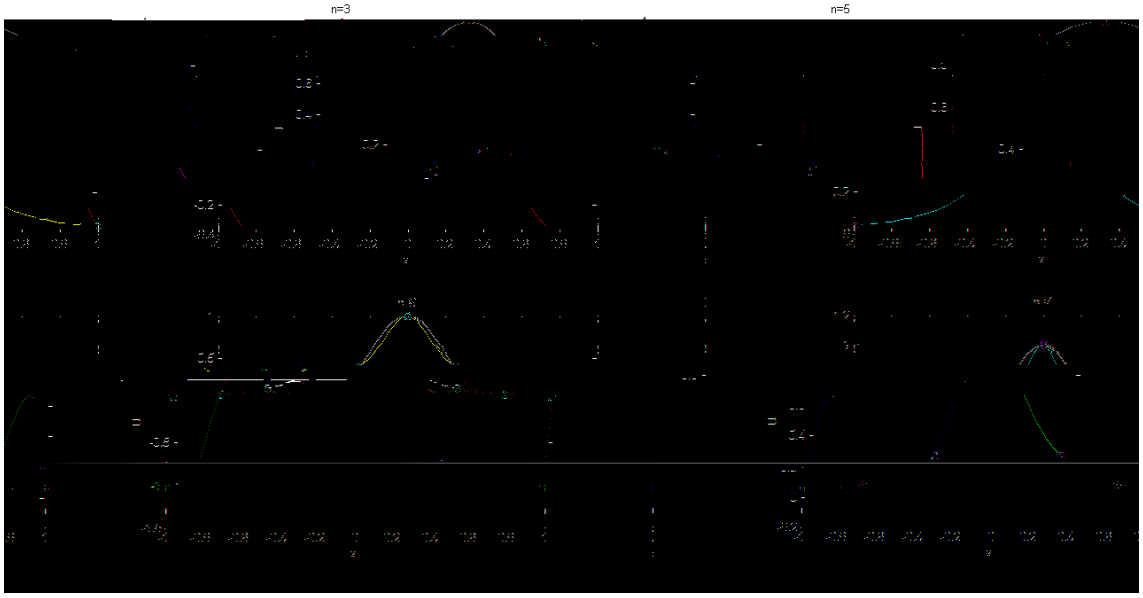


Figure 3.3: Runge's function (blue) plotted with the standard divided di erence interpolant (green) for increasing number of data points (red).

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

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

From the graphs we see that using the limited form of the divided di erence 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

(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

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

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 fali1e10 So roghsly

gure (3.7 e we the fome of the di herance inte(p)-28(olan)28(,)]TJ 0 -14.456 Td [(a
the disconi(n)27(uities)-218(in)-27[(the)-386(w)27(a)27(v)28(n.)-43[(W)82(e)-386h(a)28(v)27(
rscmiy nt aras wi(rs)-876(th1e)-827(t)1(he)-817(w)27(a)27(v)27(e)-817pro(i)1((e)-817(to

We cin cocluide that the bsat type ofoscillaoary interpolation toe
[illegible] interpolation, iance te has oscilli-
[illegible] the square wavecc u(nt)).915(the)-210limpidnd fome of

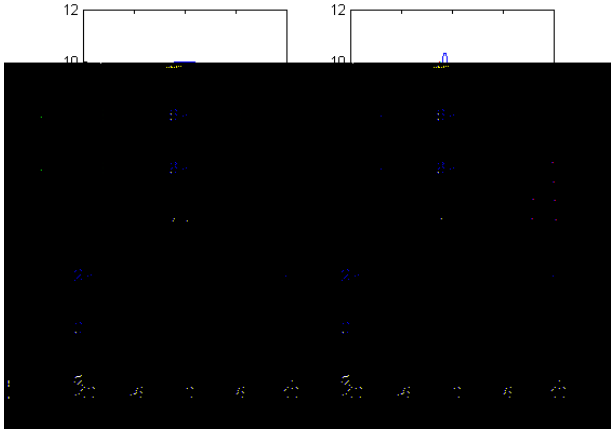


Figure 3.5: Standard piecewise cubic piecewise interpolation (dt=1, a=0.7, run for 999 time steps).

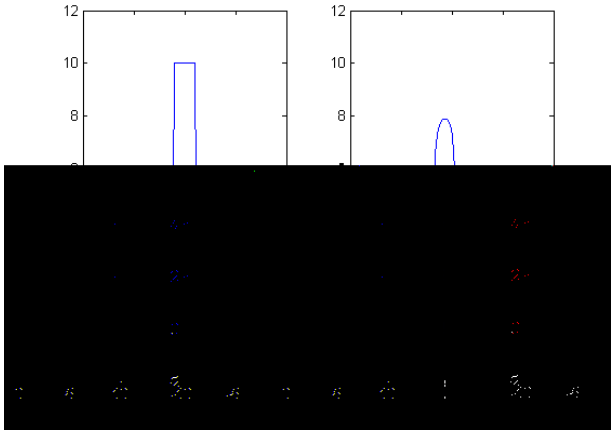


Figure 3.6: pchip interpolation (dt=1, a=0.7, run for 999 time steps).

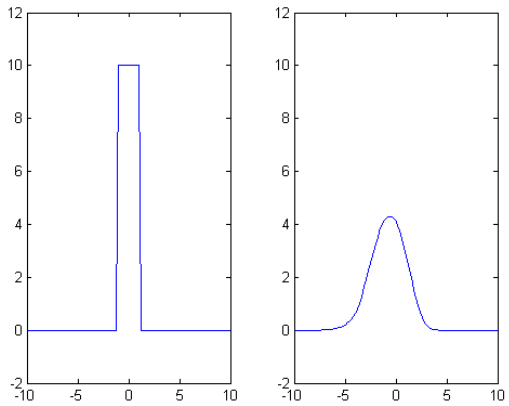


Figure 3.7: Limited form of divided di erence interpolation (dt=1, a=0.7, run for 999 time steps).

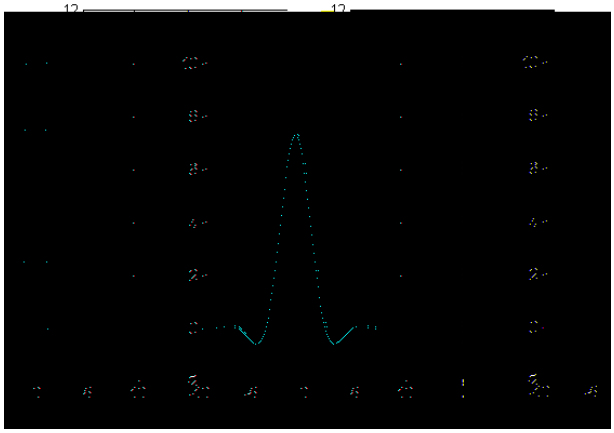


Figure 3.8: Standard piecewise cubic piecewise interpolation (dt=1, a=0.7, run for 9999 time steps).

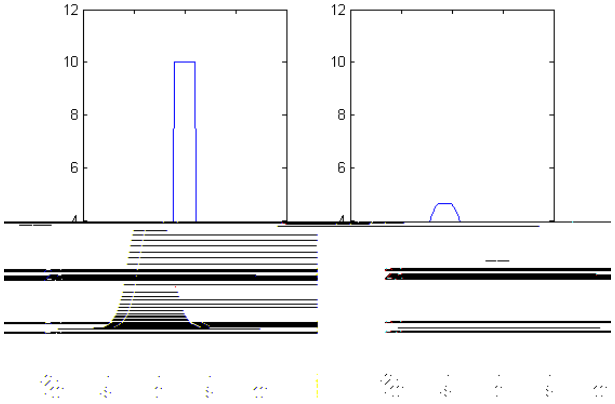


Figure 3.9: pchip interpolation (dt=1, a=0.7, run for 9999 time steps).

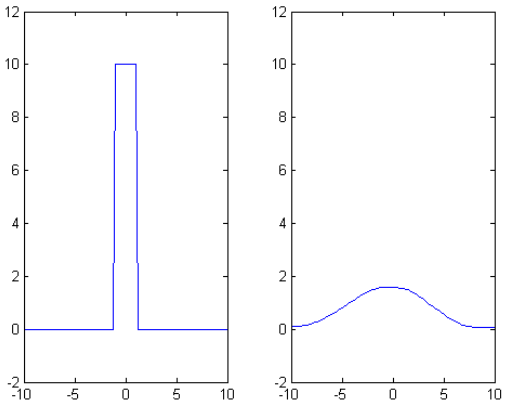


Figure 3.10: Limited form of divided di erence interpolation (dt=1, a=0.7, run for 9999 time steps).

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

$$\frac{@u}{@t} + u\frac{@u}{@x} = 0 \qquad u(x;0) = u_0(x): \tag{4.1}$$

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

$$u_0 = \begin{cases} \cos^2 \frac{\frac{x-3}{2}}{2} & jx \quad 3j \quad 2 \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

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

$$F(x_0) = x \quad u_0(x_0)t \quad x_0$$

and use the Newton-Rhaphson method:

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

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 $t = 1$.

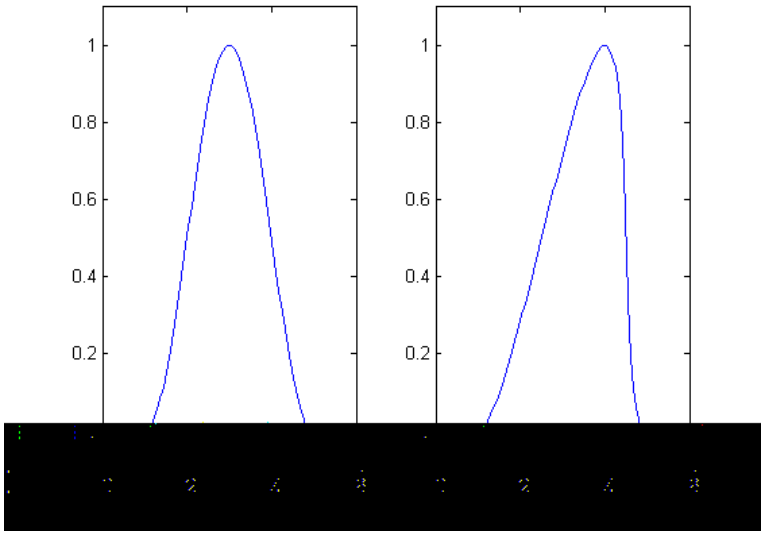


Figure 4.1: The exact solution to equation 4.1 at times $t = 0$ and $t = 1$

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 condi-

$$^{(k+1)} = tu^*(x_a \quad \frac{^{(k)}}{2}; t_n + \frac{t}{2})$$

$$\text{with} \quad u^*(x_i; t_n + \frac{t}{2}) = \frac{3}{2}u(x_i; t_n) \quad \frac{1}{2}u(x_i; t_n \qquad t)$$

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

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 t = 0:00526(initia)1(l)-326(condition)-326(())]TJ 0.00 0.00 0.5 rg 0.00 0.00 0.5

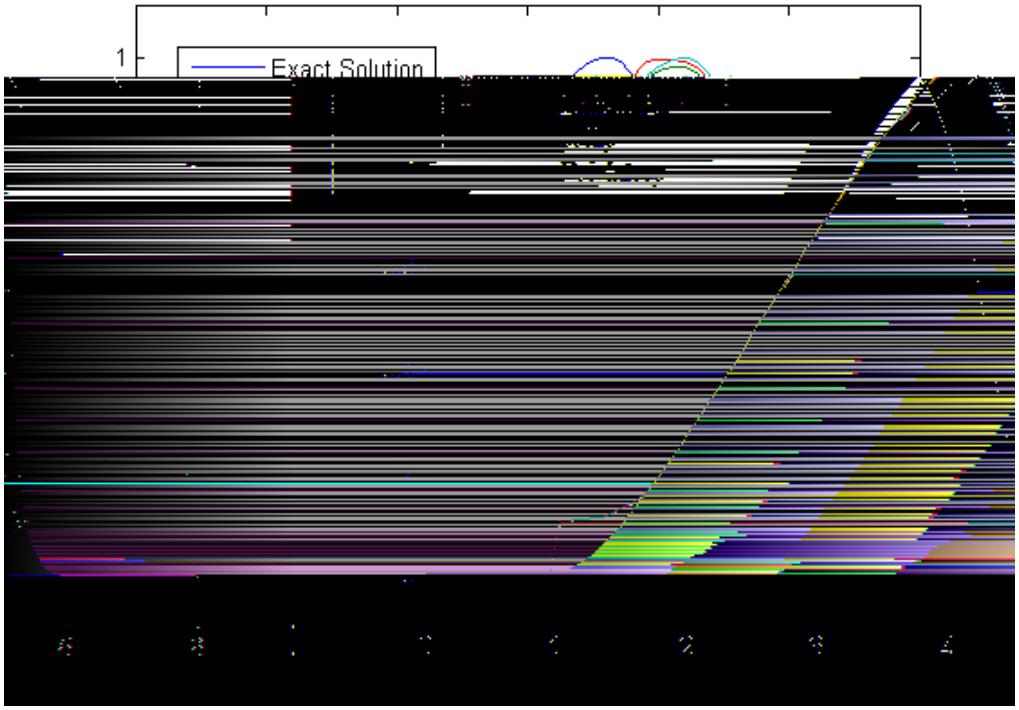


Figure 4.2: The exact solution to (4.1) at $t = 0.6$ plotted with the semi-lagrangian solution using the midpoint method and three different interpolation methods

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.

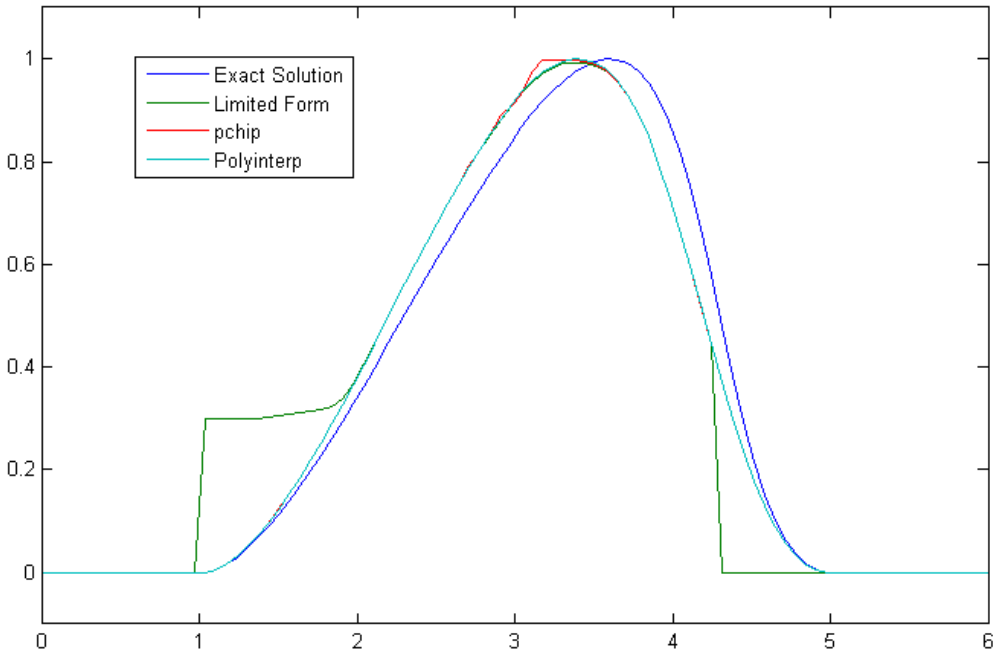


Figure 4.3: The exact solution to (4.1) at $t = 0.6$ plotted with the semi-lagrangian solution using the Shu-Osher Runge-Kutta method and three different interpolation methods

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'

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

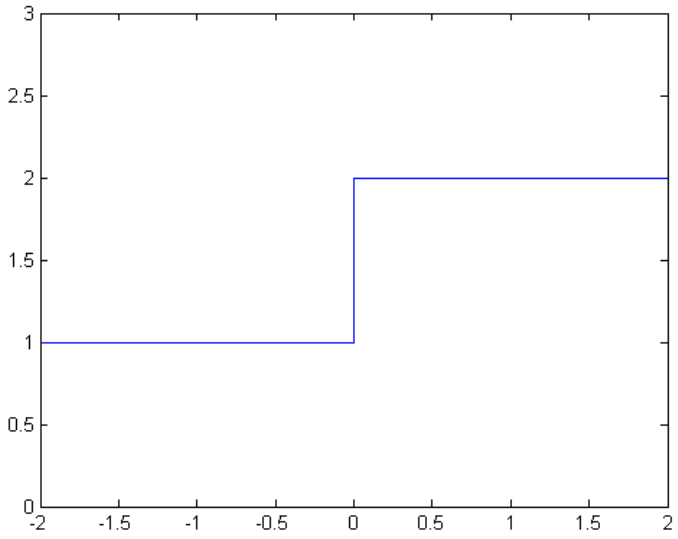


Figure 4.4: Plot of the step function used as initial condition (4.5)

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 $t = 0.005$, $x = \frac{1}{15}$. *Limited Form* denotes the limited form of the divided difference interpolant (see (2.6.1)),

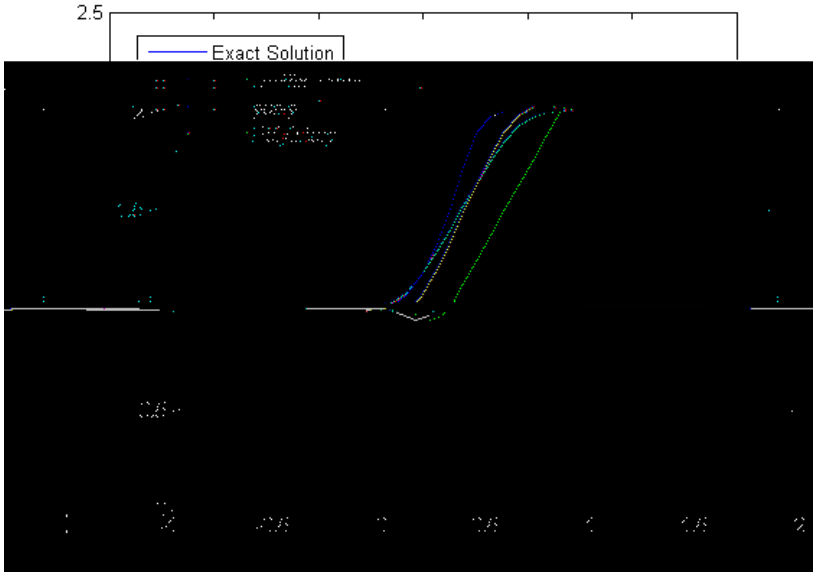


Figure 4.6: Solving equation (4.1) with initial condition (4.5) using the semi-lagrangian scheme with various interpolation methods using the Shu-Osher Runge-Kutta method for calculation of the departure point.

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

Summary and further work

Summary

In this dissertation we have compared various interpolation methods for use

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 func-

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