High Order Blobs and Particle Methods for Dispersive Partial Differential Equations

Jamie A. Burke
University of California, Los Angeles

Christopher N. Gallo
The University of Chicago

Alejandro F. Perez
University of California, Santa Barbara

July 30, 2002

Abstract

We present a numerical study of the dispersion-velocity method, introduced by Chertock and Levy (J.Com Phys. 171, 708-730(2001)). The method is implemented with high order blobs for a nonlinear dispersive PDE in 1 dimension. We present an error analysis for blobs up to order 14 that shows the convergence rates when the blob parameter, $\epsilon$, is taken as a multiple of the square root of the inter-particle spacing.

We also explore the difficulties in implementing dispersion-velocity method for a linear equation, and thus turn to the Particle Strength Exchange(PSE) Method. As above, we utilize kernels of high orders to reach high convergence rates, for a linear PDE.

1 Introduction

Numerical methods for solving partial differential equations (PDE’s) are a necessary tool in applied mathematics. Their importance stems from the fact that relatively few such equations can be solved explicitly. The following research focuses on the use of one such method in the analysis of two dispersive PDE’s in one spatial dimension.

Dispersive PDE’s are equations relating time derivatives of a function to odd ordered derivatives with respect to the spatial variable. The specific equations that we analyze below are the linear Airy Equation,

$$u_t(x,t) - u_{xxx}(x,t) = 0$$  \hspace{1cm} (1)

and the nonlinear equation

$$u_t + (u^2)_x + (u(u)_xx)_x = 0.$$  \hspace{1cm} (2)
For both of the equations periodic boundary conditions are assumed. Also, for both of the above equations, exact solutions are known for certain classes of initial data, thus facilitating error computation in the methods we use below.

Our research is motivated by the work of Chertock and Levy in [1], where they introduce a new method for numerically solving dispersive PDE’s. They adapt the diffusion-velocity method of Degond and Mustieles (1990) and develop a particle method they call the dispersion-velocity method. Examples of particle methods consist of the random vortex method and the diffusion-velocity method. In these methods, the solution of the PDE is represented as a collection of particles located at points $x_i$ with masses $w_i$.

Consider equation (2), with initial data $u(x, 0) = u_0(x)$. For notation purposes, we will let $u(x, t) = u$ such that $u_t(x, t) = u_t$ and $u_{xx}(x, t) = u_{xx}$. We can write equation (2) in the form of a linear transport equation

$$u_t(x, t) + (a(u)u(x, t))_x = 0,$$

where the velocity is taken as

$$a(u) = u_{xx} + u.$$  \hspace{1cm} (4)

The dispersion-velocity method uses the fact that equations in the conservation form of (3) will translate initial data in the form of delta functions as delta functions with velocity given by (4) [4]. Thus, if we are careful to choose initial data that can be represented as the sum of smooth approximations of delta functions, also known as blobs, we can expect them to translate as blobs with velocity in the form of (4).

Complications arise because the expression of the velocity in (4) depends on the solution to the equation and a derivative of that solution. We must first find a way to approximate the solution at a given time and use that approximation to find the velocity for a short time interval. We then translate the particles in our approximation and use the new positions of the particles to again find the velocity of the particles over the next short time interval. Bootstrapping thusly we progress to a solution of the equation at a given time.

In order to do this a method of approximating $a(u(x_i, t))$ is needed. This constitutes approximating the solution and its second derivative, with respect to location at a given time. Since we know that our solution translates delta functions in a known way, we use convolutions with blobs to approximate our solution at a given time, such as:

$$u^\tau_N(x, t) = (u_N * \phi_\tau)(x, t) = \sum_{i=1}^{N} \omega_i \phi_\tau(x - x_i(t)),$$

where the $x_i^\tau$s are the solutions to

$$\frac{dx_i}{dt} = a(u(x_i, t)), \quad x_i(0) = x_i^0$$

and $\phi_\tau$ is a specialized blob, which we describe below.
The dispersion-velocity method outlined in [1] uses such an approximation to also approximate $u_{xx}$, allowing us to solve equation (4), and effectively making (6) a solvable system of ODE’s at each time.

Before using such an approximation, it is useful to rigorously demonstrate how these convolutions approximate a function, $f(x)$, and the form of the error term.

![Some Blobs with Various Parameter Values](image)

Figure 1: 2nd Order Blobs with various parameter values in Matlab

## 2 Approximating Solutions Using Blobs

### 2.1 Approximating Functions with Convolutions

The justification for approximating $a(u(x,t))$ in equation (4) using a convolution of the function $u(x,t)$ with a blob follows directly from basic principles of calculus. The following method is taken largely from [3].

A blob is a kernel, $\phi(x)$ satisfying $\int_{-\infty}^{\infty} \phi(x)dx = 1$. It is useful for our purposes to use a parameter in scaling the blob, $\phi_{\epsilon}(x) = \frac{1}{\epsilon} \phi(\frac{x}{\epsilon})$, so that we can control the approximate width of these blobs. Notice that given a blob $\phi(x)$, $\phi_{\epsilon}(x)$ is also a blob. The following theorem describes a characteristic of scaled blobs that allows us to use it in the dispersion-velocity method.

**Theorem 1**

$$\lim_{\epsilon \to 0} \phi_{\epsilon}(x) = \delta(x)$$

where $\delta(x)$ is the Dirac Delta Function (distribution).

In choosing this function for approximating the exact solution to (2), we require the additional condition that the blob be even, for later use in error minimization.
We consider a blob with up to \( n \) bounded moments, where the function, \( f(x) \), satisfies
\[
f(x) \in C^n(\mathbb{R})
\] (7)

The convolution is defined as the following integral:
\[
(f * \phi_\epsilon)(x) = \int_{-\infty}^{\infty} f(y)\phi_\epsilon(x - y)dy
\] (8)

Utilizing the convolution of \( f(x) \) with the blob is accurate because most of the area below the curve is mapped out in the region \( x \in [-\epsilon, \epsilon] \), i.e.
\[
(f * \phi_\epsilon)(x) \approx \int_{x-\epsilon}^{x+\epsilon} f(y)\phi_\epsilon(x - y)dy \approx \int_{-\epsilon}^{\epsilon} f(x - z)\phi_\epsilon(z)dz \approx f(x) \int_{-\epsilon}^{\epsilon} \phi_\epsilon(z)dz \approx f(x)
\]

To be more precise, we generate an explicit approximation of the exact solution \( f(y) \), using a Taylor approximation of \( f(y) \) about the point \( x \):
\[
f(y) = f(x) + \sum_{k=1}^{n-1} \frac{1}{k!} (y - x)^k f^{(k)}(x) + \frac{1}{n!} (y - x)^n f^{(n)}(\xi)
\]

where \( n \) is determined by (7). We now distribute the blob with this expansion to obtain the following expression for the convolution:
\[
(f * \phi_\epsilon)(x) = \int_{-\infty}^{\infty} f(y)\phi_\epsilon(x - y)dy
\]
\[
= f(x) \int_{-\infty}^{\infty} \phi_\epsilon(x - y)dy + \sum_{k=1}^{n-1} \frac{1}{k!} f^{(k)}(x) \int_{-\infty}^{\infty} (y - x)^k \phi_\epsilon(x - y)dy
\]
\[
+ \frac{1}{n!} \int_{-\infty}^{\infty} f^{(n)}(\xi)(y - x)^n \phi_\epsilon(x - y)dy
\]
\[
= f(x) + \sum_{k=1}^{n-1} \frac{1}{k!} f^{(k)}(x) \int_{-\infty}^{\infty} (y - x)^k \phi_\epsilon(x - y)dy
\]
\[
+ \frac{1}{n!} \int_{-\infty}^{\infty} f^{(n)}(\xi)(y - x)^n \phi_\epsilon(x - y)dy
\]

(9)

We continue with a change of variables \( z = (x - y)/\epsilon \) and use the fact that \( \phi_\epsilon(x - y) = \frac{1}{\epsilon} \phi(z) \). Now we have
\[
\int_{-\infty}^{\infty} (y - x)^k \phi_\epsilon(x - y)dy = (-1)^k \epsilon^k \int_{-\infty}^{\infty} z^k \phi(z)dz \equiv (-1)^k \epsilon^k M_k(\phi),
\]

where the \( k^{th} \) moment of the function \( \phi \), \( M_k \), is defined as \( M_k(\phi) = \int_{-\infty}^{\infty} z^k \phi(z)dz \).

The moments become an essential tool in selecting a particular blob, as shown below.
Thus our convolution satisfies the following:

\[(f \ast \phi_{\epsilon})(x) = f(x) + \sum_{k=1}^{n-1} \frac{(-1)^k}{k!} \epsilon^k f^{(k)}(x) M_k(\phi) + O(\epsilon^n).\] (10)

This demonstrates that the convolution gives us an approximation of the function, with error in the form determined by (10).

### 2.2 Using Convolutions to Solve a Nonlinear Equation

Having shown that it is reasonable to use the convolution of a function with a blob to get an approximation of the function, we must now approximate the convolution itself, as defined in (8). We approximate this expression using the trapezoidal rule, giving us that

\[u_N'(x, t) = \sum_{j=1}^{N} \omega_j \phi(x - x_j)\] (11)

where the \(\omega_j\)'s represent the product of the weight of the particles and the spacing between them. It should be noted that to achieve better approximations special attention must be paid to the relationship between the spacing of the points and the width parameter of the scaled blobs. We discuss this later.

Now with an explicit way of approximating the function at a given time we need only find an expression for the second spatial derivative to begin solving (6), where \(a(u)\) is determined by (4). The *dispersion-velocity method* suggests using

\[(u_N'')'(x, t) = \sum_{j=1}^{N} \omega_j (\phi_{xx})(x - x_j).\] (12)

With these expressions we can complete the bootstrapping method described above, using the velocity expression to find the motion of the particles over a small time interval, and then using the new positions of the particles to solve (11). Prior to this a specific blob must be found to use in these approximations.

### 3 Constructing Higher Order Blobs

Recall that a blob \(\phi(x) : \mathbb{R} \to \mathbb{R}\) in one dimension, satisfies \(\int_{-\infty}^{\infty} \phi(x) dx = 1\), and is smooth. Here we will use the general even function

\[\phi(x) = \left(\sum_{r=0}^{s} \Gamma_r x^{2r}\right) \frac{e^{-x^2}}{\sqrt{\pi}},\]

where the \(\Gamma_r\)'s are constant.

Recall also, that the convolution is

\[(f \ast \phi_{\epsilon})(x) = f(x)M_0(\phi) - \epsilon f(x)M_1(\phi) + \frac{1}{2!} \epsilon^2 f''(x) M_2(\phi) - \ldots + \frac{(-1)^n}{n!} \epsilon^n f^{(n)}(x) M_n(\phi).\]
Because we select an even function as our blob, all odd moments are zero. Thus our convolution becomes

\[(f\ast\phi)(x) = f(x)M_0(\phi) + \frac{1}{2!}e^2 f^{II}(x)M_2(\phi) + \frac{1}{4!}e^4 f^{IV}(x)M_4(\phi) + \ldots + \frac{(-1)^n}{n!}e^n f^{(n)}(x)M_n(\phi).\]

We commence the discussion on the specific conditions a blob must satisfy to be considered of order 2, 4, \ldots , n.

For a 2\textsuperscript{nd} Order Blob,

\[
\phi_2(x) = \Gamma_0 \frac{e^{-x^2}}{\sqrt{\pi}}
\]

we only have one undetermined constant.
1.) The 0\textsuperscript{th} moment is equivalent to one, \(M_0 = 1\), which is the definition of a blob. Thus we have

\[
\int_{-\infty}^{\infty} \Gamma_0 \frac{e^{-x^2}}{\sqrt{\pi}} = 1
\]

After solving for the constant \(\Gamma_0\), we obtain the the 2\textsuperscript{nd} order blob

\[
\phi_2(x) = \frac{e^{-x^2}}{\sqrt{\pi}}.
\]

For a 4\textsuperscript{th} Order Blob, two conditions can be satisfied:
1.) The 0\textsuperscript{th} moment is equal to one, \(M_0 = 1\).
2.) The 2\textsuperscript{nd} moment is equal to zero, \(M_2 = 0\)

We now consider a 4\textsuperscript{th} Order Blob

\[
\phi_4 = (\Gamma_0 + \Gamma_1 x^2) \frac{e^{-x^2}}{\sqrt{\pi}}
\]

that satisfies the conditions \(\int_{-\infty}^{\infty} (\Gamma_0 + \Gamma_1 x^2) \frac{e^{-x^2}}{\sqrt{\pi}} = 1\) and \(\int_{-\infty}^{\infty} x^2 (\Gamma_0 + \Gamma_1 x^2) \frac{e^{-x^2}}{\sqrt{\pi}} = 0\).

Thus we have

\[
\phi_4 = \left(\frac{3}{2} - x^2\right) \frac{e^{-x^2}}{\sqrt{\pi}}.
\]

We notice that for each consecutive order of blob, another degree of freedom is added to our function.
For a 6th order blob 3 degrees of freedom are needed

$$\phi_6 = (\Gamma_0 + \Gamma_1 x^2 + \Gamma_2 x^4) \frac{e^{-x^2}}{\sqrt{\pi}}.$$ 

Since this is a 6th order blob:
1.) The 0th moment is equal to one, \( M_0 = 1 \).
2.) The 2nd moment is equal to zero, \( M_2 = 0 \)
3.) The 4th moment is equal to zero, \( M_4 = 0 \).

As we closely analyze the Taylor’s Theorem, because the first 3 moments are controlled, we notice that the convolution is

$$ (f \ast \phi_6) = f(x) + O(\epsilon^6) $$

for the 6th order blob

$$ \phi_6 = \left( \frac{15}{8} - \frac{5}{2} x^2 + \frac{1}{2} x^4 \right) \frac{e^{-x^2}}{\sqrt{\pi}}.$$ 

For an \( n \)th order blob the convolution is

$$ (f \ast \phi_n) = f(x) + O(\epsilon^n). $$

Since \( \epsilon \) is small, as the order of the blob increases in the convolution, the error decreases, and the approximation is closer to the exact solution. In theory the order of the blob can be increased without bound leading to an infinite order blob. In the following investigation, it is observed that in practice the reduction of the error ceases after a certain point.

Figure (2) depicts the many blobs that we have constructed above.

4 Results Using the Dispersion-Velocity Method

4.1 Data Tables

We now consider the dispersion-velocity method for the nonlinear dispersive equation

$$ u_t + (u^2)_x + (uu_{xx})_x = 0 \quad u(x,0) = 2 \cos^2(x/2). \quad (13) $$

We know that this equation has the exact solution

$$ u(x,t) = 2 \left[ \cos \left( \frac{x - t}{2} \right) \right]^2, \quad |x - t| \leq \pi.$$
In order to analyze the quality of each method we introduce the following definition, which will apply to all of our error analysis below.

**Definition 1** The convergence rate of a numerical approximation using $N_1$ and $N_2 > N_1$ particles and generating errors $E_1$ and $E_2$, respectively is

$$\log_2\left(\frac{E_1}{E_2}\right)/\log_2\left(\frac{N_2}{N_1}\right)$$

Our primary motivation for this work is to improve upon the converge rates found in [1], where both the infinity norm and the two norm yielded convergence rates approximately equal to one. That is to say, as the number of particles used was doubled, the errors decreased by a factor of 2. These rates were obtained using the 4th order blob

$$\phi_4(x) = \frac{1}{\sqrt{\pi}} \left(\frac{3}{2} - x^2\right) e^{-x^2}.$$  \hspace{1cm} (14)

The following chart is included [1].

| Grid | $||u - u_N||_\infty$ | $L^\infty$ Rate | $||u - u_N||_2$ | $L^2$ Rate |
|------|----------------|-----------------|----------------|------------|
| $N = 40$ | 0.039 | - | 0.068 | - |
| $N = 80$ | 0.019 | 0.986 | 0.0345 | 0.986 |
| $N = 160$ | $9.77e - 3$ | 0.989 | 0.017 | 0.989 |
| $N = 320$ | $4.93e - 3$ | 0.988 | $8.72e - 3$ | 0.989 |
Figure 3: Plot of Numerical Solutions to the Nonlinear Equation with $\epsilon = 1.5\sqrt{h}$ and a $4^{th}$ order blob. Notice that the positions of the particles above are dynamic, but their weights are fixed in time.

Where $\epsilon = 0.5\sqrt{h}$ and $T = 2$. A forth-order blob is expected to have errors $O(\epsilon^4) = O(h^2)$, but the reported errors are $O(h)$. There are some notable differences between the way Chertock and Levy evaluate the solution and our method. They use:

$$u(x,0) = \begin{cases} 
0 & \text{for } x < -\pi \\
2\cos^2(x/2) & \text{for } x \in [-\pi, \pi] \\
0 & \text{for } x > \pi 
\end{cases}$$

This function $u(x,t)$ is continuous for it’s $1^{st}$ derivative $u(x,t)_x$ but it is discontinuous for the $2^{nd}$ derivative $u(x,t)_{xx}$. This introduces mathematical difficulties, since the dispersive-velocity method’s velocity, $a(u)$ is directly dependent on the function and it’s $2^{nd}$ derivative.

We circumvent this problem altogether by implementing periodic boundary conditions, thus the function, $u(x,t)$ is smooth for all derivatives of the solution. We can thus use the Taylor’s approximation without difficulties.

Using our Matlab approximation functions for the $4^{th}$ order blob, we obtain convergence rates approximately equal to two. This is expected for the choice $\epsilon = C\sqrt{h}$.

Due to their nature, if higher-order blobs are used, it is possible to obtain lower errors and higher convergence rates. In theory one could increase the order of a blob to infinity, and obtain increasingly better results, although in practice this process has limits. For the finite data set that we explored, $N=40$ to $N=640$ we were only
able to go to the 12th order blob while witnessing improvements in convergence rates and errors.

Table 4.2: Convergence Rates for the Solution to Equation(13)

| Grid | $||u - u_N^*||_{\infty}$ | $L^\infty$ Rate | $||u - u_N^*||_2$ | $L^2$ Rate |
|------|--------------------------|-----------------|-----------------|------------|
| $N = 40$ | 0.0518 | - | 0.094 | - |
| $N = 80$ | 0.014 | 1.864 | 0.026 | 1.882 |
| $N = 160$ | 0.004 | 1.933 | 0.007 | 1.94 |
| $N = 320$ | 9.53e-4 | 1.967 | 0.002 | 1.966 |

Where $\epsilon = 3\sqrt{n}$ and $T=2$. Our goal was to investigate the reduction of the error as a function of the blob.

For the 6th ordered blob

$$\phi_6 = \left( \frac{15}{8} - \frac{5}{2} x^2 + \frac{1}{2} x^4 \right) e^{-x^2}$$

(15)

In table 4.3 we obtain convergence rates that are approximately equal to 3. The initial errors for $N=40$ are also extremely low, about an order of magnitude less than the 4th ordered blob.

Table 4.3: Convergence Rates for the Solution to Equation(13)

| Grid | $||u - u_N^*||_{\infty}$ | $L^\infty$ Rate | $||u - u_N^*||_2$ | $L^2$ Rate |
|------|--------------------------|-----------------|-----------------|------------|
| $N = 40$ | 0.028 | - | 0.05 | - |
| $N = 80$ | 4.24E-3 | 2.711 | 7.60E-3 | 2.73 |
| $N = 160$ | 5.85E-4 | 2.856 | 1.04E-3 | 2.865 |
| $N = 320$ | 7.68E-5 | 2.928 | 1.37E-4 | 2.93 |

Where $\epsilon = 4\sqrt{n}$, $T = 2$.

The 8th ordered blob

$$\phi_8 = \left( \frac{35}{16} - \frac{25}{8} x^2 + \frac{7}{4} x^4 - \frac{1}{6} x^6 \right) e^{-x^2}$$

(16)

yields convergence rates that are approximately equal to 4, as seen in table 4.4. As we reach an optimal $\epsilon$, the errors become very small, although they are still comparable to the 6th ordered blob, for $N=40$. Since the convergence rates are so high, when increasing the number of particles evaluated, by 2 say, the error drops by a factor of $2^4$. 

10
Table 4.4: Convergence Rates for the Solution to Equation (13)

| Grid | $||u - u_N||_\infty$ | $L^\infty$ Rate | $||u - u_N||_2$ | $L^2$ Rate |
|------|-----------------------|----------------|----------------|-------------|
| $N = 40$ | 0.019 | - | 0.035 | - |
| $N = 80$ | $1.72e - 3$ | 3.503 | $3.08e - 3$ | 3.518 |
| $N = 160$ | $1.27e - 4$ | 3.751 | $2.27e - 4$ | 3.761 |
| $N = 320$ | $8.67e - 6$ | 3.876 | $1.54e - 5$ | 3.881 |

Where $\epsilon = 5\sqrt{n}$, $T = 2$.

The 10th ordered blob

$$\phi_{10} = \frac{(\frac{315}{128} - \frac{105}{16}x^2 + \frac{63}{16}x^4 - \frac{3}{4}x^6 + \frac{1}{24}x^8)}{\sqrt{\pi}} e^{-x^2}$$  (17)

results in a convergence rate approximately equal to 5, as the number of particles are increased, as demonstrated in table 4.5.

Table 4.5: Convergence Rates for the Solution to Equation (13)

| Grid | $||u - u_N||_\infty$ | $L^\infty$ Rate | $||u - u_N||_2$ | $L^2$ Rate |
|------|-----------------------|----------------|----------------|-------------|
| $N = 40$ | $9.50e - 3$ | - | $1.73e - 2$ | - |
| $N = 80$ | $4.69e - 4$ | 4.339 | $8.43e - 4$ | 4.355 |
| $N = 160$ | $1.84e - 5$ | 4.673 | $3.28e - 5$ | 4.684 |
| $N = 320$ | $6.44e - 7$ | 4.837 | $1.15e - 6$ | 4.839 |

Where $\epsilon = 5.6\sqrt{n}$, $T = 2$.

The 12th ordered blob

$$\phi_{12} = \frac{(\frac{693}{256} - \frac{155}{128}x^2 + \frac{231}{32}x^4 - \frac{33}{16}x^6 + \frac{11}{48}x^8 - \frac{1}{120}x^{10})}{\sqrt{\pi}} e^{-x^2}$$  (18)

yields convergence rates approximately equal to 6, which can be seen in table 4.6. Unfortunately, we witness low convergence rates for small values of $N$. This leads to the conjecture, that very high order blobs are best suited for large numbers of particles. Nevertheless, these are extremely satisfactory, and in principle we can evaluate extremely large number of particles and obtain very minuscule error.
Table 4.6: Convergence Rates for the Solution to Equation(13)

<table>
<thead>
<tr>
<th>12th Order Blob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid</td>
</tr>
<tr>
<td>( N = 40 )</td>
</tr>
<tr>
<td>( N = 80 )</td>
</tr>
<tr>
<td>( N = 160 )</td>
</tr>
<tr>
<td>( N = 320 )</td>
</tr>
</tbody>
</table>

Where \( \epsilon = 7\sqrt{h}, \quad T = 2 \).

The results for the 14th ordered blob

\[
\phi_{14} = \frac{(\frac{3003}{1024} - \frac{3003}{256} x^2 + \frac{3003}{256} x^4 - \frac{143}{32} x^6 + \frac{143}{128} x^8 - \frac{13}{230} x^{10} + \frac{1}{720} x^{12}) e^{-x^2}}{\sqrt{\pi}}
\]

are inconclusive for the number of particles that we are able to evaluate. Our convergence rates do not reach the expected constant 7 (see table 4.7). They oscillate between 2 and 5, which is very inconsistent, and difficult to analyze. We conjecture that this is also attributed to the small number of particles that we are able to evaluate. Since the convergence rate is an asymptotic value as \( h \) approaches zero, with larger \( N \), convergence rates nearing 7 are expected.

Table 4.7: Convergence Rates for the Solution to Equation(13)

<table>
<thead>
<tr>
<th>14th Order Blob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid</td>
</tr>
<tr>
<td>( N = 40 )</td>
</tr>
<tr>
<td>( N = 80 )</td>
</tr>
<tr>
<td>( N = 160 )</td>
</tr>
<tr>
<td>( N = 320 )</td>
</tr>
</tbody>
</table>

Where \( \epsilon = 10\sqrt{h}, \quad T = 2 \).

4.2 Relating Epsilon to Error

Using a particular blob, the weights of the given particles, and the distance between consecutive particles, we approximate continuous functions. In order to normalize the blob, we introduce an \( \epsilon \), a small parameter strategically inserted into the blob to avoid singularity. This \( \epsilon \) maintains the regularity of the blob, but different values of
\(\epsilon\) result in different errors when comparing the particle approximation to the exact solution. We relate \(\epsilon\) to the distance between the particles, \(h\), such that, \(\epsilon = C \sqrt{h}\) where \(h = 2\pi/N\) and \(N\) is the number of particles.

One aspect of our present investigation involves understanding the mathematical relationship between \(\epsilon\) and the error in the approximation given by the dispersive velocity method. For a given blob, we hold constant the number of particles, \(N\), while we vary \(\epsilon\) in order to find the smallest error. As we vary the constant \(C\), we expect to find an optimal \(\epsilon\) that would return the smallest error. Our investigation confirms the assumption, as the errors plotted against \(\epsilon\) are almost parabolic, with the minimum error corresponding to our optimal \(\epsilon\). We further expect that as we increase \(N\), the optimal \(\epsilon\) would decrease. Yet, we have not seen this relationship. We attribute this deviation to the inherent properties of the blob. For higher order blobs, there are more oscillations within the blob, and depending on the solution function that we expect to obtain, this may or may not be desired. Our investigation into this finding continues.

### 4.3 Optimal Delta Graph Analysis

![Fourth Order Blob: Optimal Delta for Various N](image)

Figure 4: The Optimal Epsilon at Different N Using a 4\(^{th}\) Order Blob in Matlab

For any number of points, the above graphs are exactly as would be expected according to the nature of blobs. The minimum point of each graph is interpreted as the optimal value of the parameter \(\epsilon\), noted \(\epsilon_{\text{best}}\) for using that number of points to approximate the solution to (2). The catastrophic rise in the error for values of \(\epsilon < \epsilon_{\text{best}}\) represents the point at which the width of the blobs being used is not enough
An unexpected behavior demonstrated in this graph is that the values of $\epsilon_{\text{best}}$ increase proportionally to the number of points being used to estimate. Due to the inverse proportionality between the number of points used in the approximation and the distance between the approximating blobs, one would expect that as the blobs get closer together the epsilon parameter would be allowed to decrease in accordance. This is clearly not the case, however, in either of the above graphs depicting the value of $\epsilon_{\text{best}}$ for the $4^{th}$ and $8^{th}$ order blob approximations.

One possible explanation that is consistent with our results is that the number of particles necessary to asymptotically approach the exact solution for a particular order of blob directly corresponds to the order of the method. We conjecture that if we extend these graphs to include the lines for 1280, 2560, and 5120 particles, we would see a reversal, and these graphs will discontinue their present trend. We leave a full explanation of the behavior demonstrated above open to further research.
5 The Dispersion-Velocity Method and the Linear Problem

Following the application of the dispersion-velocity method to the nonlinear equation (2) we turned to analysis of the linear equation below with periodic boundary conditions

\[ u_t(x, t) - u_{xxx}(x, t) = 0 \quad u(x, t = 0) = \cos(x) \]  

(20)

using the same method. Putting (1) into the form of (3) we see that the particle velocities must satisfy the following:

\[ a(u) = -\frac{u_{xx}}{u} \]  

(21)

Since we know that the solution to (20) is

\[ u_{\text{exact}} = \cos(x - t) \]  

(22)

we know that \( u_{xx} = -\cos(x - t) \) and so we want \( a(u) = 1 \). In [1] the following blob is used to approximate the solution to this equation

\[ \phi_\epsilon(x) = \frac{e^{\frac{-x^2}{2\epsilon}}}{{\epsilon}^{\sqrt{2\pi}}} \]  

(23)

This is troublesome, however, given that the solution in (22), attains the value of 0 at certain points, in the denominator of (21). In [1] this fact is recognized, and the recommendation is made that near such locations, \( \frac{1}{u} \) be approximated with \( \frac{u}{u^2 + \delta^2} \) for some small \( \delta \). In theory the approximation can be carried out exactly as it was done for the nonlinear problem above, using the final adjustment

\[ a(x, t) = -\frac{u_{xx}u}{u^2 + \delta^2} \]  

(24)

Using this method with the blob in (23) and \( \epsilon = .5\sqrt{\frac{2\pi}{N-1}} \) we were not able to duplicate the results generated in [1]. This is likely because \( \lim_{u\to0}(\frac{u_{xx}u}{u^2 + \delta^2}) = 0 \neq 1 \), though we know that \( a(x, t) \equiv 1 \).

As a second attempt, we tried using another smooth approximation

\[ a(x, t) = \sqrt{\frac{u_{xx}^2 + \delta^2}{u^2 + \delta^2}} \]  

(25)

after observing that \( \lim_{u\to0}\sqrt{\frac{u_{xx}^2 + \delta^2}{u^2 + \delta^2}} = 1 \), (since \( u = -u_{xx} \), so \( u \to 0 \) implies \( u_{xx} \to 0 \)). While this smooth approximation allowed us to generate similar errors to those
in [1], (25) was dependent on our knowledge of the exact solution and its derivatives. We found this fact rather unsatisfying as a step in a method of approximation. While this was the only linear problem we attempted to approximate using the dispersion-velocity method it is clear that for all linear equations a denominator of \( u \) will arise in the process of putting the equation into the form of (3), and will thus lead to a similar issue if the range of the function includes 0. As an alternative to the dispersion-velocity method for linear problems, we implemented the following method described in [5].

6 The Particle Strength Exchange Method

6.1 Theory and Derivation

We now look at another particle method, used to help us approximate solutions to the linear partial differential equation. Recall our linear dispersive model problem

\[
  u_t - u_{xxx} = 0, \quad u(x, 0) = \cos(x)
\]

where the solution to the above equation takes the form \( u(x, t) = \cos(x - t) \). We use an extension of the Particle Strength Exchange (PSE) method presented in [5], that approximates spatial derivatives in particle methods using integral operators, meaning that we can approximate derivatives of any order. The PSE method allows us to approximate \( u_{xxx} \), directly, instead of converting the PDE to the linear transport equation. However, to approximate \( u_{xxx} \), we construct high order blob functions. The method is known as PSE because of the conservation properties that are inherent when two particles exchange strength with one another.

Eldredge, Leonard, and Colonious introduce the general integral PSE operator for approximating the differential operator \( D^\beta \) in [5]. Since we are dealing in one dimension, our differential operator becomes

\[
  D^k = \frac{d^k}{dx^k} = f^{(k)}, \quad k = 1, 2, 3, \ldots
\]

which give us the \( k \)-th derivative of a function. Thus, the PSE operator takes the form

\[
  L^k u(x) = \frac{1}{\varepsilon^k} \int_{-\infty}^{\infty} (u(y) \mp u(x)) \Lambda^k y(x - y) dy, \tag{26}
\]

where \( \Lambda \) is a scaled kernel function. Since we are looking to obtain an approximation of \( u_{xxx} \), Eq.(26) takes the form

\[
  L^3 u(x) = \frac{1}{\varepsilon^3} \int_{-\infty}^{\infty} (u(y) + u(x)) \Lambda^3 y(x - y) dy, \tag{27}
\]
As mentioned earlier, this derivative approximation will depend on what kernels we use. Specifically, since the function we wish to approximate the derivative of is odd, we choose to construct our kernel to be odd. In order to obtain an approximation for any differentiable operator, we want to approximate the integrand in Eq.(26) using the convolution function defined earlier.

Consider a Taylor expansion of a smooth function \( u(y) \) about a point \( x \) which satisfies

\[
  u(y) = u(x) + \sum_{n=1}^{\infty} \frac{1}{n!} u^{(n)}(x)(y-x)^n
\]  

(28)

Subtracting \( u(x) \) from both sides of Eq.(28) and applying a convolution to each term with the unknown kernel \( \Lambda_c \), we yield

\[
  L^k u(x) = \frac{1}{\epsilon^k} \sum_{n=1}^{\infty} \frac{1}{n!} u^{(n)}(x) \int_{-\infty}^{\infty} (y-x)^n \Lambda^k_c(x-y) dy,
\]

(29)

where the differential operator is defined in Eq.(26). This holds true, only if \( u(x) \) has infinitely many derivatives and our kernel \( \Lambda_c \) has infinitely many bounded moments. Thus, Eq.(29) can be expressed as

\[
  L^k u(x) = \sum_{n=1}^{\infty} \frac{(-1)^n \epsilon^{n-k}}{n!} u^{(n)}(x) M_n(\Lambda),
\]  

(30)

where \( M_n(\Lambda) \) is known as the \( n \)-th moment of \( \Lambda \). Since derivative we wish to approximate is odd, due to our exact solution being cosine, we will choose to construct an odd kernel so that all even moments be zero. We will discuss the construction of our kernel further in our discussion. In a similar manner, if we wanted to approximate the derivative of an even function then all the odd moments would be zero. Hence, we can write Eq.(30) as

\[
  L^k u(x) = \sum_{n=2}^{\infty} \frac{(-1)^{(2n+1)} \epsilon^{2n-k+1}}{(2n + 1)} u^{(2n+1)}(x) M_{(2n+1)}.
\]  

(31)

To construct a useful derivative approximation, we must satisfy the essential moment conditions that will make some of the moments in the summation equal zero and the \( k \)-th moment be nonzero. The moment conditions we must satisfy are \( M_k = (-1)^k k! \) if \( n = k \), and \( M_k = 0 \) if \( n \neq k \), for \( k \) up to some maximum value. If we denote \( k \) to be 3, then what we are approximating is \( u_{xxx} \), which is what we want. In order to obtain an approximation for \( u_{xxx} \), we wish to satisfy the moment conditions where \( M_0 = 0, M_1 = 0, M_3 = -6, \) and \( M_{2n} = 0 \) such that

\[
  u_{xxx} \approx u^{(m)}(x) + \sum_{n=2}^{\infty} \frac{(-1)^{(2n+1)} \epsilon^{2n-2}}{(2n + 1)} u^{(2n+1)}(x) M_{(2n+1)}.
\]

(32)

Thus, behold our approximation for \( u_{xxx} \).
Now that we have an approximation for $u_{xxx}$, we construct the kernel function that is essential for our approximation to work. Since the derivative we wish to approximate is odd, we construct our kernel $\Lambda(z)$ to be odd, so that consequently, all even moments become zero. To build these kernels we use the general form

$$\Lambda(x) = \frac{x}{\pi} \left( \sum_{j=0}^{m} a_j x^{2j} \right) e^{-x^2}$$

introduced in [5], for which all even moments vanish. For example, to construct a kernel of order 4, the kernel we attempt to construct takes the form

$$\Lambda(x) = \frac{x}{\pi} (a_0 + a_1 x^2 + a_2 x^4) e^{-x^2}$$

satisfying the moment conditions $M_0 = 0, M_1 = 0, M_3 = -6, M_5 = 0$, and all even moments equal 0. Finding our coefficients, we derive the 4th-order kernel to be

$$\Lambda(x) = \frac{x}{\sqrt{\pi}} (42 - 48 x^2 + 8 x^4) e^{-x^2}. \quad (35)$$

In a similar manner, we derive the 6th-order and 8th-order kernels satisfying the moment conditions $M_0 = 0, M_1 = 0, M_3 = -6, M_5 = 0, M_7 = 0, M_9 = 0$, respectively. These kernels take the form of

$$\Lambda(x) = \frac{-x}{\sqrt{\pi}} \left( (-189/2) + 153 x^2 - 50 x^4 + 4 x^6 \right) e^{-x^2} \quad \text{and} \quad (36)$$

$$\Lambda(x) = \frac{-x}{\sqrt{\pi}} \left( (-693/4) + 363 x^2 - 176 x^4 + 28 x^6 - (4/3)x^8 \right) e^{-x^2} \quad (37)$$

6.2 Numerical Simulations for Linear Equation

In this section we present our numerical results obtained from the PSE method. Now that we have obtained an approximation for $u_{xxx}$, we can approximate a solution to the linear equation

$$u_t - u_{xxx} = 0$$

with initial condition $u(x, 0) = \cos(x)$, exact solution $u(x, t) = \cos(x - t)$, and periodic boundary conditions $x \in [-\pi, \pi]$. The method is as follows.

Let $u_{xxx} = F(u)$, where $F(u)$ is the approximation defined by Eq.(27). If we let

$$u_t = F(u), \quad (38)$$

then we can think of Eq.(38) as

$$u_t + 0 \cdot u_x = F(u). \quad (39)$$
The method goes back to the method of characteristics, where parametrize our solution by letting $Z(s) = u(x(s), t(s))$. Then,

$$Z'(s) = x'(s)u_x + t'(s)u_t.$$ 

Thus, we form a system of Ordinary Differential Equations, where

$$
egin{align*}
t'(s) &= 1, & t(0) &= 0 \\
x'(s) &= 0, & x(0) &= r \\
Z'(s) &= F(Z), & Z(0) &= u(r, 0)
\end{align*}
$$

and solve these using our ODE 45 in our MatLab program.

To approximate solutions to our linear PDE, we will first we use the 4th order kernel

$$
\Lambda(x) = \frac{x}{\sqrt{\pi}}(42 - 48x^2 + 8x^4)e^{-x^2}.
$$

We choose such a blob in hopes of obtaining small error and improve the order of accuracy. We select our width of our kernel to be $\epsilon = 0.8\sqrt{h}$, with $2\pi/(N - 1)$, and the number of particles we take are 40, 80, 160, and 320. We show the convergence rate of our linear problem at time equal to one in Table 9.1. The table shows that the convergence rate is approximately two. We display the exact and approximate solutions of the linear problem in 6.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$\infty$-Norm</th>
<th>log$_2$ ratio</th>
<th>2-Norm</th>
<th>log$_2$ ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>3.266e-4</td>
<td>–</td>
<td>5.893e-4</td>
<td>–</td>
</tr>
<tr>
<td>80</td>
<td>8.028e-5</td>
<td>2.024</td>
<td>1.436e-4</td>
<td>2.037</td>
</tr>
<tr>
<td>160</td>
<td>1.99e-5</td>
<td>2.012</td>
<td>3.544e-5</td>
<td>2.019</td>
</tr>
<tr>
<td>320</td>
<td>4.956e-6</td>
<td>2.006</td>
<td>8.803e-6</td>
<td>2.01</td>
</tr>
</tbody>
</table>

Using a higher order kernel, say of 6th order in Eq.(36), we approximate the same linear PDE with the same initial condition, same boundary conditions, and same exact solution. For this example we let $\epsilon = \sqrt{h}$, where $h = 2\pi/(N - 1)$ and we take $N$ to be 40, 80, 160, 320. We show the convergence rate at time equal to 1 for this example in Table 9.2. Notice that for a higher order kernel we obtain a convergence rate of three, a higher order of accuracy than the previous example. A similar approach is taken when we consider an 8th-order kernel defined in Eq.(37). The convergence rate at time equals one is displayed in Table 9.3. We choose $\epsilon = \sqrt{h}$. 

19
Figure 6: Plot of Numerical Solutions to the Nonlinear Equation with $\epsilon = 0.8\sqrt{h}$ and a $4^{th}$ order kernel. Notice that the positions of the particles above are fixed in time, but their weights are dynamic.

### Table 9.2: Convergence Rate for the Linear Problem $u_t - u_{xxx} = 0$
with Initial Data $u(x, 0) = \cos(x)$

<table>
<thead>
<tr>
<th>$N$</th>
<th>$\infty$-Norm</th>
<th>$\log_2$ ratio</th>
<th>2-Norm</th>
<th>$\log_2$ ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>$1.057e-5$</td>
<td>$-$</td>
<td>$1.9065e-5$</td>
<td>$-$</td>
</tr>
<tr>
<td>80</td>
<td>$1.29e-6$</td>
<td>3.033</td>
<td>$2.308e-6$</td>
<td>3.046</td>
</tr>
<tr>
<td>160</td>
<td>$1.595e-7$</td>
<td>3.016</td>
<td>$2.84e-7$</td>
<td>3.023</td>
</tr>
<tr>
<td>320</td>
<td>$1.985e-8$</td>
<td>3.007</td>
<td>$3.522e-8$</td>
<td>3.011</td>
</tr>
</tbody>
</table>

### Table 9.3: Convergence Rate for the Linear Problem $u_t - u_{xxx} = 0$
with Initial Data $u(x, 0) = \cos(x)$

<table>
<thead>
<tr>
<th>$N$</th>
<th>$\infty$-Norm</th>
<th>$\log_2$ ratio</th>
<th>2-Norm</th>
<th>$\log_2$ ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>$1.062e-7$</td>
<td>$-$</td>
<td>$1.916e-7$</td>
<td>$-$</td>
</tr>
<tr>
<td>80</td>
<td>$6.41e-9$</td>
<td>4.05</td>
<td>$1.146e-8$</td>
<td>4.063</td>
</tr>
<tr>
<td>160</td>
<td>$3.94e-10$</td>
<td>4.025</td>
<td>$7.01e-10$</td>
<td>4.031</td>
</tr>
<tr>
<td>320</td>
<td>$2.47e-11$</td>
<td>3.996</td>
<td>$4.28e-11$</td>
<td>4.035</td>
</tr>
</tbody>
</table>

Tables 9.1-9.3 show that the PSE method employed for the linear problem consistently achieves convergence rates within 0.001 of the theoretically expected value.
The success of this method, in contrast with the difficulties in using the dispersion-velocity method, can be attributed to the elimination of the division by \( u(x, t) \), which equals zero in portions of the evaluated domain. To successfully navigate past this issue requires too much prior knowledge of certain fixes, for example:

\[
\frac{u_{xx}}{u} = \frac{u}{u^2 + \delta^2} u_{xx}
\]

and

\[
\frac{u_{xx}}{u} = \sqrt{\frac{u_{xx}^2 + \delta^2}{u^2 + \delta^2}}.
\]

Although these fixes aim to eliminate any singularity, the parameter \( \delta \) is too sensitive. It proves very difficult to find the correct \( \delta \), thus making this method unsatisfactory. Therefore, the precision and ease of the more robust PSE method are preferable to the dispersion-velocity method for the linear equation case.

We note that although the convergence rates in tables 12.1 - 12.3 closely approach the theoretical convergence rates, this accuracy does not come without cost. The computation time for each data point of the PSE tables 12.1-12.3 is approximately twice that of single data points for the tables 4.2-4.7, and the evaluated time is only \( T=1 \). In one example, although the error using the 8th order kernel in table 12.3 with \( N=320 \) was extremely low, \( O(10^{-11}) \), such accuracy requires the use of a tolerance with the Matlab Runge-Kutta ODE solver (ode23) of \( \sim 10^{-14} \). In this setting, the calculation of the approximate solution requires \( \sim 5 \) hours using Matlab on a PC with a 1.33 GHz AMD processor and 256 MB of RAM. Such time expenditures lead to ideas about future research, where one can make the program more time effective.

7 Conclusions and Extensions

Our work demonstrates the success, up to practical limits, of incorporating blobs of increasing orders in the dispersion-velocity method. We use these blobs to achieve high convergence rates for approximating nonlinear dispersive PDEs. This paper also provides an alternative to this method for approximating linear dispersive PDEs, and demonstrates a similar success. We feel that the compilations of our data is extremely significant in the improvement of both the dispersion-velocity method and the Particle Strength Exchange Method. We are able look at the data and determine the appropriate order of blob, to get the lowest error, given the parameter \( N \), the number of particles and \( \epsilon \) the scaling parameter. We can also choose a specific blob to use for a given the number of particles, \( N \). We then introduce a new question of an optimal \( \epsilon \), and future research may answer the question of the relative error for different numbers of particles.
Acknowledgements

We would like to thank our project director Dr. Ricardo Cortez of Tulane University for his guidance, expertise and advice throughout the project. We would also like to thank our seminar associate, Alexander Villacorta, the SIMU directors, Dr. Herbert Medina of Loyola Marymount University and Dr. Ivelisse Rubio of the University of Puerto Rico – Humacao for providing us with the opportunity to conduct this research. Finally we thank the National Science Foundation and the National Security Agency for their funding of SIMU.

References