Generalized-Monotone Finite Difference Methods with Third-Order Local Truncation Error for Stationary Hamilton-Jacobi Equations

Author: S.M.N.S. Seneviratne
Advisor: Dr. Thomas Lewis

June 16, 2018
Contents

1 Introduction 2

2 A New Finite Difference Method 5
   2.1 Notation and Difference Operators ................. 5
   2.2 The Lax-Friedrich's Method and the Convergence Framework of Crandall and Lions .................. 8
   2.3 A New Third-Order Method .......................... 10

3 Numerical Tests 12
   3.1 Test 1. Smooth Polynomial Solution .................. 13
   3.2 Test 2. Smooth Solution ............................. 13
   3.3 Test 3. Non-smooth Solution .......................... 16

4 Conclusions and Future Directions 18

References 18

Appendix 19
Abstract

A new finite difference (FD) approximation method is proposed for stationary Hamilton-Jacobi problems with Dirichlet boundary conditions. The FD methods have third-order local truncation error and are natural extensions of the second order methods of Feng and Lewis that are proved to converge to the unique viscosity solution of the first order fully nonlinear partial differential equation. The new methods are not monotone. Instead, a stabilization term called a numerical moment is used to ensure the proposed schemes are consistent, admissible, and stable. The new schemes are natural high-order extensions of the convergent Lax-Friedrich’s method. Numerical tests are provided that demonstrate the accuracy and flexibility of the schemes.

1 Introduction

The mathematical theory of partial differential equations (PDE) has a long rich history due to their ability to describe many physical phenomena. Unfortunately only a very small portion of PDEs can be solved exactly. Thus, when the analytical solution is not available, we seek to find an approximate solution to the PDE using numerical methods. The goal of this project is to propose a new third-order finite difference method for approximating the solution to stationary fully nonlinear first order PDE problems with Dirichlet boundary conditions. In particular, we focus on the stationary Hamilton–Jacobi (HJ) problem that seeks a function $u$ such that

\begin{align}
H[u] &= H(\nabla u(x), u(x), x) = 0, \quad x \in \Omega, \\
u(x) &= g(x), \quad x \in \partial \Omega,
\end{align}

where $\nabla$ represents the gradient operator; $\Omega \subset \mathbb{R}^d$ is an open, bounded domain; $H$ is locally Lipschitz in its first two arguments; $H$ satisfies the comparison principle; and $g$ is continuous.

In physics, the HJ problem is a reformulation of classical mechanics. Mathematically the HJ equation is a necessary condition describing extremal geometry in generalizations of problems from the calculus of variations, and
it is a special case of the Hamilton–Jacobi–Bellman equation. It is named for William Rowan Hamilton and Carl Gustav Jacob Jacobi.

The problem (1) is not guaranteed to have a classical solution in $C^1(\Omega) \cap C(\Omega)$. Due to the full nonlinearity of the operator in (1), standard weak solution theory based upon multiplication by a regular test function and integration by parts is not applicable. Furthermore, there may exist multiple functions that satisfy the PDE almost everywhere in $\Omega$. Crandall and Lions introduced in [1] the concept of “viscosity” solutions for HJ problems. The strength of viscosity solution theory is that it provides a framework weak enough to allow solutions of (1) in $C(\Omega)$ while also having enough structure to guarantee the uniqueness of the solution. Since we are assuming both $H$ and $g$ are continuous in (1), we have the following definition of a viscosity solution.

**Definition 1.1.** Let $H$ denote the first order operator in (1).

(i) A function $u \in C(\Omega)$ is called a viscosity subsolution of (1) if $\forall \varphi \in C^1$ such that $u - \varphi$ has a local maximum at $x_0 \in \Omega$, there holds

$$H(\nabla \varphi(x_0), u(x_0), x_0) \leq 0.$$ 

(ii) A function $u \in C(\Omega)$ is called a viscosity supersolution of (1) if $\forall \varphi \in C^1$ such that $u - \varphi$ has a local minimum at $x_0 \in \Omega$, there holds

$$H(\nabla \varphi(x_0), u(x_0), x_0) \geq 0.$$ 

(iii) A function $u \in C(\Omega)$ is called a viscosity solution of (1) if $u$ is both a viscosity subsolution and viscosity supersolution.

The above definition can be informally interpreted as follows. The function $u$ is a viscosity solution of (1) if for all smooth functions $\varphi$ such that $\varphi$ “touches” the graph of $u$ from above at $x_0$, we have

$$H(\nabla \varphi(x_0), \varphi(x_0), x_0) \leq 0$$

and for all smooth functions $\varphi$ such that $\varphi$ “touches” the graph of $u$ from
below at $x_0$, we have
\[ H(\nabla \varphi(x_0), \varphi(x_0), x_0) \geq 0. \]

We can see the illustration of this graphical relationship in Figures 1 and 2. We note that such an approach is sometimes referred to as "differentiation-by-parts" since all derivatives are locally moved from the solution to a test function. This is in contrast to an "integration by parts" approach that globally moves derivatives onto a test function and is the basis of weak solution theory.

![Figure 1: $\varphi$ touches the graph of $u$ from above at $x_0 \in \Omega$.](image)

Using all of the above assumptions, we have problem (1) has a unique viscosity solution $u \in C(\overline{\Omega})$. Our goal is to approximate the viscosity solution using a new finite difference method with third order local truncation
error. The remainder of this paper is organized as follows. In Section 2 we introduce our finite difference notation as well as the Lax-Friedrich’s method for approximating equation (1) and the convergence framework of Crandall and Lions. We also introduce our new third-order method as a high-order generalization of the Lax-Friedrich’s method. We provide several numerical tests in Section 3. The code for the numerical tests is provided in the Appendix.

2 A New Finite Difference Method

In this section we develop a general finite difference (FD) method for approximating viscosity solutions of first order fully nonlinear Dirichlet boundary value problems.

2.1 Notation and Difference Operators

In order to define our new method, we first introduce various difference operators for approximating first order partial derivatives.

Assume $\Omega$ is a $d$-rectangle, i.e. $\Omega = (a_1, b_1) \times (a_2, b_2) \times \cdots \times (a_d, b_d)$. We will only consider grids that are uniform in each coordinate $x_i$, $i = 1, 2, \cdots, d$. Let $J_i$ be a positive integer and $h_i = (b_i - a_i)/(J_i - 1)$ for $i = 1, 2,$
Define \( h = (h_1, h_2, \ldots, h_d) \in \mathbb{R}^d \), \( h = \max_{i=1,2,\ldots,d} h_i \), \( J = \prod_{i=1}^{d} J_i \), and \( N_d^f = \{ \alpha = (\alpha_1, \alpha_2, \ldots, \alpha_d) \mid 1 \leq \alpha_i \leq J_i, i = 1, 2, \ldots, d \} \). Then, \(|N_d^f| = J\). We partition \( \Omega \) into \( \prod_{i=1}^{d} (J_i - 1) \) sub-\( d \)-rectangles with grid points \( x_\alpha = (a_1 + (\alpha_1 - 1)h_1, a_2 + (\alpha_2 - 1)h_2, \ldots, a_d + (\alpha_d - 1)h_d) \) for each multi-index \( \alpha \in N_d^f \).

Let \( e_i \) denote the canonical basis vectors for \( \mathbb{R}^d \) where \( i = 1, \ldots, d \). Then the first order forward and backward difference operators for approximating partial derivatives are defined by

\[
\delta^{+}_{x_i, h_i} v(x) \equiv \frac{v(x + h_i e_i) - v(x)}{h_i}, \quad \delta^{-}_{x_i, h_i} v(x) \equiv \frac{v(x) - v(x - h_i e_i)}{h_i},
\]

respectively.

Our new method will be based on using high-order extensions of \( \delta^{\pm}_{x_i, h_i} \). To this end, we introduce the second order forward and backward difference operators \( \tilde{\delta}^{\pm}_{x_i, h_i} \) defined as follows:

\[
\tilde{\delta}^{+}_{x_i, h_i} v(x) \equiv \frac{-v(x + 2h_i e_i) + 4v(x + h_i e_i) - 3v(x)}{2h_i},
\]

\[
\tilde{\delta}^{-}_{x_i, h_i} v(x) \equiv \frac{3v(x) - 4v(x - h_i e_i) + v(x - 2h_i e_i)}{2h_i}.
\]

We also introduce the central difference operators \( \delta_{x_i, h_i} \) and \( \tilde{\delta}_{x_i, h_i} \) defined by

\[
\delta_{x_i, h_i} \equiv \frac{1}{2} (\delta^{+}_{x_i, h_i} + \delta^{-}_{x_i, h_i}), \quad \tilde{\delta}_{x_i, h_i} \equiv \frac{1}{6} (\delta^{+}_{x_i, h_i} + \tilde{\delta}^{-}_{x_i, h_i}) + \frac{1}{3} (\delta^{+}_{x_i, h_i} + \delta^{-}_{x_i, h_i}).
\]

**Lemma 2.1.**

The operators \( \tilde{\delta}^{+}_{x_i, h_i}, \tilde{\delta}^{-}_{x_i, h_i} \) are second order accurate.
Proof.

Assume \( v \in C^2(\Omega) \). Then,

\[
\tilde{\delta}^+_{x_i,h_i}v(x) = \frac{-v(x + 2h_ie_i) + 4v(x + h_ie_i) - 3v(x)}{2h_i} = \frac{-[v(x) + 2h_i v_{x_i}(x) + 2h_i^2 v_{xx_i}(x) + \mathcal{O}(h_i^3)]}{2h_i} + \frac{[4v(x) + 4h_i v_{x_i}(x) + 2h_i^2 v_{xix_i}(x) + \mathcal{O}(h_i^3)] - [3v(x)]}{2h_i} = \frac{2h_i v_{x_i}(x) + \mathcal{O}(h_i^3)}{2h_i} = v_{x_i}(x) + \mathcal{O}(h_i^2),
\]

and

\[
\tilde{\delta}^-_{x_i,h_i}v(x) = \frac{3v(x) - 4v(x - h_ie_i) + v(x - 2h_ie_i)}{2h_i} = \frac{[3v(x)] - [4v(x) - 4h_i v_{x_i}(x) + 2h_i^2 v_{xix_i}(x) + \mathcal{O}(h_i^3)]}{2h_i} + \frac{[v(x) - 2h_i v_{x_i}(x) + 2h_i^2 v_{xix_i}(x) + \mathcal{O}(h_i^3)]}{2h_i} = \frac{2h_i v_{x_i}(x) + \mathcal{O}(h_i^3)}{2h_i} = v_{x_i}(x) + \mathcal{O}(h_i^2).
\]

Lemma 2.2.

The central difference operators \( \delta_{x_i,h_i}, \tilde{\delta}^-_{x_i,h_i} \) have second and fourth order accuracy, respectively.
Proof.

By the definition of the $\delta_{x_i,h_i}$ operator and Taylor’s expansion, there holds

$$\delta_{x_i,h_i}v(x) = v_{x_i}(x) + O(h_i^2).$$

We also have by the definition of the tilde operator and by Taylor’s expansion,

$$\tilde{\delta}_{x_i,h_i}v(x) = \frac{1}{12h_i} \left( -v(x + 2h_ie_i) + 4v(x + h_ie_i) - 4v(x - h_ie_i) + v(x - 2h_ie_i) \right)$$

$$+ \frac{1}{3h_i} \left( v(x + h_ie_i) - v(x - h_ie_i) \right)$$

$$= \frac{1}{12h_i} \left( -v(x + 2h_ie_i) + 8v(x + h_ie_i) - 8v(x - h_ie_i) + v(x - 2h_ie_i) \right)$$

$$= \frac{1}{12h_i} \left( 12v_{x_i}(x)h_i - \frac{48}{120} h_i^5 v_{x_ix_ix_ix_i}(x) + O(h_i^7) \right)$$

$$= v_{x_i}(x) + O(h_i^4).$$

\( \Box \)

Lastly, we introduce the various discrete gradient operators $\nabla_{h}^{\pm}$, $\nabla_{h}$, $\tilde{\nabla}_{h}^{\pm}$, and $\tilde{\nabla}_{h}$, where the components are all based on the corresponding discrete partial derivative operators, and the standard (second order) central difference operators $\delta_{x_i,h_i}^2$ for approximating second order partial derivatives defined by

$$\delta_{x_i,h_i}^2 v(x) \equiv \frac{v(x + h_ie_i) - 2v(x) + v(x - h_ie_i)}{h_i^2}.$$ 

2.2 The Lax-Friedrich’s Method and the Convergence Framework of Crandall and Lions

In this section we consider finite difference methods of the form

$$\tilde{H}(\nabla_{h}^- U_{\alpha}, \nabla_{h}^+ U_{\alpha}, U_{\alpha}, x_{\alpha}) = 0,$$ (2)
where $\hat{H}$ is called a numerical Hamiltonian operator that can depend upon multiple discrete gradient approximations. Assuming $\hat{H}$ is continuous, we say $\hat{H}$ is consistent if $\hat{H}(q, q, v, x) = H(q, v, x)$ for any $q \in \mathbb{R}^d$, $v \in \mathbb{R}$, and $x \in \Omega$. For the stationary problem, we say $\hat{H}$ is monotone on $[-R, R]$ if it is nondecreasing with respect to the node $U_\alpha$ and nonincreasing with respect to each node $U_{\alpha'}$ such that $x_{\alpha'} \neq x_\alpha$ and $\max_{\alpha} |U_\alpha| \leq R$. We say the scheme (2) is admissible if it has a solution and stable if all solutions are uniformly bounded independent of $h$. The classical result of Crandall and Lions states that if the FD scheme (2) for approximating (1) is consistent, monotone, admissible, and stable, then the FD solution converges locally uniformly to the viscosity solution of (1).

The use of two derivative approximations is useful for enforcing monotonicity and natural when considering how to resolve local extrema of a function that potentially has lower regularity. Suppose $v$ has a local maximum at $x_0$. Observe that $\delta^-_{x_0, h_i} v(x_0) \geq 0$ and $\delta^+_{x_0, h_i} v(x_0) \leq 0$. When combining these observations with the monotonicity properties of $\hat{H}$, we see that we can move the discrete derivative operators from the grid function to a local smooth test function $\varphi$ when $v(x_\alpha) = U_\alpha - \varphi(x_\alpha)$ has a local maximum at $x_\alpha$.

A standard example of a consistent, monotone FD method is the Lax-Friedrich's method defined by

$$\hat{H}[U_\alpha] \equiv H(\nabla_h U_\alpha, U_\alpha, x_\alpha) - \vec{\beta} \cdot (\nabla_h^+ - \nabla_h^-)U_\alpha,$$

where $\vec{\beta} \geq \vec{0}$. Observe that if $H$ is globally Lipschitz and each component of $\vec{\beta}$ is sufficiently large, then the Lax-Friedrich's numerical Hamiltonian is nonincreasing with respect to $\nabla_h^+ U_\alpha$ and nondecreasing with respect to $\nabla_h^- U_\alpha$ using the fact that $\nabla_h U_\alpha = \frac{1}{2} (\nabla_h^+ + \nabla_h^-) U_\alpha$. By writing these operators in terms of their nodal representations, it follows that the method is monotone.

The term $\vec{\beta} \cdot (\nabla_h^+ - \nabla_h^-)U_\alpha$ is called a numerical viscosity due to the fact that
\[ \vec{\beta} \cdot (\nabla_h^+ - \nabla_h^-) v(x) = -\sum_{i=1}^{d} \beta_i (\delta_{x_i,h_i}^+ - \delta_{x_i,h_i}^-) v(x) \]
\[ = -\sum_{i=1}^{d} \beta_i \frac{v(x + h_i e_i) - 2v(x) + v(x - h_i e_i)}{h_i} \]
\[ = \sum_{i=1}^{d} \beta_i h_i \delta_{x_i,h_i}^2 v(x) \]
\[ = \sum_{i=1}^{d} \beta_i h_i (v_{x_i,x_i} + O(h_i^2)) . \]

Thus, the method corresponds to adding a vanishing linear diffusion term consistent with the vanishing viscosity approach discussed in \[1\]. We can also see that the scheme is limited to first order accuracy. It turns out that every convergent monotone FD scheme for (4) implicitly approximates the differential equation
\[ -\beta h \Delta u + H(\nabla u, u, x) = 0 \]
for sufficiently large and possibly nonlinear \( \beta > 0 \), as shown in \[3\]. To achieve a third order scheme we will weaken the monotonicity condition and instead rely upon a stabilization technique.

### 2.3 A New Third-Order Method

In this section we introduce a new third order method for approximating the viscosity solution to (1). The main idea for our new method is to replace the low-order derivative approximations in the Lax-Friedrich’s scheme with high order analogues.

Choose \( \tilde{\gamma} > 0 \) a constant. We propose the FD method defined by finding a grid function \( U_\alpha \) such that
\( \hat{H}_h[U_\alpha] = 0, \quad x_\alpha \in \Omega, \) \hfill (5a)

\( U_\alpha = g(x_\alpha), \quad x_\alpha \in \partial \Omega, \) \hfill (5b)

\( \delta_{x_i,h_i}^2 U_\alpha = \delta_{x_i,h_i}^2 U_{\alpha'}, \quad x_\alpha \in \partial \Omega, \) \hfill (5c)

where \( x_{\alpha'} \) is the closest interior neighbor with \( e_i \) normal to \( \partial \Omega \) and \( \hat{H}_h \) is defined by

\[
\hat{H}_h[U_\alpha] \equiv \beta h^2 \cdot (\nabla^- - \nabla^+) U_\alpha + H(\tilde{\nabla}_h U_\alpha, U_\alpha, x_\alpha) + \tilde{\gamma} \cdot (\tilde{\nabla}_h^- - \tilde{\nabla}_h^+) U_\alpha. \quad (6)
\]

Note that the last boundary condition (5c) is introduced to define the ghost-points needed to enforce (5a) at interior nodes that neighbor the boundary. We call the term \( \tilde{\gamma} \cdot (\tilde{\nabla}_h^- - \tilde{\nabla}_h^+) U_\alpha \) a numerical moment due to the fact that

\[
\left( \tilde{\delta}_h^- - \tilde{\delta}_h^+ \right) v(x) = \frac{3v(x) - 4v(x - h_i e_i) + v(x - 2h_i e_i)}{2h_i} - \frac{-v(x + 2h_i e_i) + 4v(x + h_i e_i) - 3v(x)}{2h_i} = \frac{1}{2h_i} \left( v(x + 2h_i e_i) + v(x - 2h_i e_i) + 6v(x) - 4v(x + h_i e_i) - 4v(x - h_i e_i) \right) = h_i^3 v_{x_i,x_i,x_i} + O(h_i^5).
\]

Thus, \( v_{x_i,x_i,x_i} \) is scaled by \( h_i^3 \), and it follows that the method is a direct realization of the vanishing moment method of [3]. Since the numerical viscosity is now scaled by \( h^2 \) and \( \tilde{\nabla}_h \) is a fourth-order accurate approximation of the first derivative, we have the scheme has a third order local truncation error.

The scheme we have proposed is an extension of the second order method of [2] that has been proved to converge to the unique viscosity solution of (1). Using similar techniques, it can be shown that if \( H \) is globally Lipschitz
and \( \tilde{\gamma} \) is sufficiently large, then the new method is admissible, stable, and consistent. We conjecture that the scheme converges to the viscosity solution based on the stabilizing properties of numerical moment.

### 3 Numerical Tests

In this section we test the third order accuracy of the proposed method as well as its ability to approximate a lower regularity viscosity solution. We will consider several one-dimensional problems with \( \Omega = (a, b) \) and explore the effect of the numerical moment. We will only consider uniformly spaced grids.

In one dimension, the scheme can be written as finding a grid function \( U \) such that

\[
\tilde{H}_h[U] \equiv \beta h^2(\delta^-_{x,h} - \delta^+_{x,h})U_i + H(\tilde{\delta}_{x,h}U_i, U_i, x_i) + \gamma(\tilde{\delta}^-_{x,h} - \tilde{\delta}^+_{x,h})U_i = 0 \tag{7}
\]

for \( i = 1, 2, \ldots, N - 1 \) with

\[
U_0 = g(x_0), \quad U_N = g(x_N),
\]

\[
\delta^2_{x,h}U_0 = \delta^2_{x,h}U_1, \quad \delta^2_{x,h}U_N = \delta^2_{x,h}U_{N-1}.
\]

We will consider two nonlinear solvers for solving (7). The first solver will input an initial guess given by the secant line formed by the boundary data. It will then use an explicit update specified by

\[
\Delta_h U^{(n+1)} = \Delta_h U^{(n)} - \rho \tilde{H}_h[U^n], \tag{8}
\]

where \( \rho > 0 \) is a fixed constant. The update is called \( M \) times as a way to improve the initial guess before using \texttt{fsolve} to try to solve (7). Our tests indicate that \texttt{fsolve} has a hard time solving the nonlinear system with a generic initial guess, and the two phase approach can be used to help \texttt{fsolve} find a solution.
\begin{center}
\begin{tabular}{|c|c|c|}
\hline
$h$  & Error & Order \\ \hline
3.33e-01 & 9.83e-01 & \text{---} \\
1.58e-01 & 7.35e-01 & 0.39 \\
7.69e-02 & 7.37e-03 & 6.40 \\
3.80e-02 & 6.91e-04 & 3.35 \\
1.89e-02 & 7.87e-05 & 3.11 \\
9.40e-03 & 8.99e-06 & 3.12 \\
\hline
\end{tabular}
\end{center}

Table 1: Approximation results for Test 1 using $\gamma = 20$.

### 3.1 Test 1. Smooth Polynomial Solution

Consider the problem
\[ H(u_x, u, x) = u_x^2 + x^2 u - f(x) = 0 \tag{9} \]
with $\Omega = (-2, 1)$ and $f$ chosen such that the exact solution is $u(x) = x^3$. The results for $\gamma = 20$ can be found in Table 1 and Figures 1-2 for various size meshes. We can observe that the error converges with rate three. We note that the error is not exact despite the fact that the solution is cubic due to the error created by the auxiliary boundary condition (5c).

### 3.2 Test 2. Smooth Solution

Consider the problem
\[ H(u_x, x) = \sin^2(u_x) + u_x - f(x) = 0 \tag{10} \]
with $\Omega = (-6, 6)$ and $f$ chosen such that the exact solution is $u(x) = \cos(x)$. The results for $\gamma = 20000$ can be found in Table 2 and the solution’s behavior is pictured in Figures 3-4. Figure 4 indicates that when $h$ is small the approximate solution coincides with the exact solution.
Figure 3: Test 1 plots for $N = 80$ and $N = 160$ with $\gamma = 20$.

Figure 4: Test 1 plot with $N = 640$ and $\gamma = 20$. 
Figure 5: Test 2 plots for $N = 160$ and $N = 320$ with $\gamma = 20000$.

Figure 6: Test 2 plot with $N = 2560$ and $\gamma = 20000$. 

15
Table 2: Approximation results for Test 2 using $\gamma = 20000$.

<table>
<thead>
<tr>
<th>$h$</th>
<th>Error</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.33e+00</td>
<td>1.93e+00</td>
<td></td>
</tr>
<tr>
<td>6.32e-01</td>
<td>1.94e+00</td>
<td>-0.00</td>
</tr>
<tr>
<td>3.08e-01</td>
<td>2.25e+00</td>
<td>-0.20</td>
</tr>
<tr>
<td>1.52e-01</td>
<td>2.45e+00</td>
<td>-0.13</td>
</tr>
<tr>
<td>7.55e-02</td>
<td>5.72e-01</td>
<td>2.08</td>
</tr>
<tr>
<td>3.76e-02</td>
<td>6.31e-02</td>
<td>3.17</td>
</tr>
<tr>
<td>1.88e-02</td>
<td>7.92e-03</td>
<td>2.99</td>
</tr>
<tr>
<td>9.38e-03</td>
<td>9.81e-04</td>
<td>3.01</td>
</tr>
<tr>
<td>4.69e-03</td>
<td>1.22e-04</td>
<td>3.01</td>
</tr>
</tbody>
</table>

Table 3: Approximation results for Test 3 using $\gamma = 1$.

<table>
<thead>
<tr>
<th>$h$</th>
<th>Error</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.22e-01</td>
<td>5.27e-02</td>
<td></td>
</tr>
<tr>
<td>1.05e-01</td>
<td>3.34e-02</td>
<td>0.61</td>
</tr>
<tr>
<td>5.13e-02</td>
<td>1.67e-02</td>
<td>0.97</td>
</tr>
<tr>
<td>2.53e-02</td>
<td>8.28e-03</td>
<td>0.99</td>
</tr>
<tr>
<td>1.26e-02</td>
<td>4.12e-03</td>
<td>1.00</td>
</tr>
<tr>
<td>6.27e-03</td>
<td>2.05e-03</td>
<td>1.00</td>
</tr>
<tr>
<td>3.13e-03</td>
<td>1.03e-03</td>
<td>1.00</td>
</tr>
</tbody>
</table>

3.3 Test 3. Non-smooth Solution

Consider the problem

$$H(u_x) = |u_x| - 1 = 0$$  \hspace{1cm} (11)

with $\Omega = (-1, 1)$ and the boundary data chosen such that the exact solution
is $u(x) = 1 - |x|$. The results for $\gamma = 1$ can be found in Table 2 and
the behavior is pictured in Figures 5-6. Figure 6 indicates that when $h$ is
small the approximate solution coincides with the exact solution. We observe
optimal rates based on the regularity of the solution.
Figure 7: Test 3 plots for $N = 40$ and $N = 160$ with $\gamma = 1$.

Figure 8: Test 3 plot with $N = 640$ and $\gamma = 1$. 
4 Conclusions and Future Directions

We have studied a non-monotone scheme for approximating Hamilton-Jacobi equations that is based on a simple extension of the Lax-Friedrich’s method. The method is a direct realization of the vanishing moment method. The finite difference scheme has third order local truncation error. Numerical tests indicate that the scheme has high-order when the solution is smooth but also converges when the solution has lower regularity. The next step of the project is to try to prove the convergence of the scheme by exploiting properties of numerical moment.

References


Appendix

In this section we have included the Matlab code for Test 1 found in Section 3.1. The code includes the following .m files

- PDE domain.m
- PDE H.m
- PDE F.m
- PDE u.m
- initialize.m
- approximation parameters.m
- FD newton solver.m
- FD sobolev solver.m
- F hat.m
- F norm.m
- trisolve.m
- sobolev.m
- plot.m
- tests.m
(1) **PDE_domain.m**

```matlab
function [xmin, xmax] = PDE_domain()
    xmin = -2.0;
    xmax = 1.0;
```

(2) **PDE_H.m**

```matlab
function out = PDE_H(q,u,x)
    out = q.*q + x.*x.*u - (PDE_u(x,1)).*(PDE_u(x,1)) - x.*x.*PDE_u(x,0);
    return;
```

(3) **PDE_F.m**

```matlab
function out = PDE_F(q,u,x)
    out = q.*q + x.*x.*u - (PDE_u(x,1)).*(PDE_u(x,1)) - x.*x.*PDE_u(x,0);
    return;
```

(4) **PDE_u.m**

```matlab
function out = PDE_u(x,xDeriv)
    if (xDeriv == 0)  % u
        out = x.*x.*x;
    elseif (xDeriv == 1)  % u_x
        out = 3*x.*x;
    else
        fprintf('ERROR -- unknown derivative in PDE_u
        '); 
    end
```

(5) **initialize.m**
function [T_h, U_guess] = initialize(N_x)

[x_min, x_max] = PDE_domain();

T_h.h = (x_max - x_min)/(N_x-1);

T_h.N = N_x;

T_h.mesh = [x_min T_h.h:T_h.h:x_max T_h.h]; % creating N+2 nodes

U_left = PDE_u(x_min,0); % left boundary point
U_right = PDE_u(x_max,0); % right boundary point

u_secant = @(x) (U_right - U_left)/(x_max-x_min) * (x - x_min*ones(1,length(x))) + U_left*ones(1,length(x));
U_guess = u_secant(T_h.mesh); % u_guess is given by using the secant line

%U_guess = -T_h.mesh.*T_h.mesh + ones(1,T_h.N+2);

(6) approximation_parameters.m

function [T_h, Solver_newton,Solver_sobolev, U_guess] = approximation_parameters()

N_x = 500;

Solver_newton.MaxIt = 40; % number of iterations of newtons method
Solver_newton.tolerance = 1.0e-12;
Solver_sobolev.MaxIt = 10000; % number of iterations for the second method
Solver_sobolev.rho = 0.00001;

[T_h, U_guess] = initialize(N_x);

T_h.alpha = 20;
T_h.epsilon = 1.0;

(7) F_norm.m

function value = F_norm(T_h, U)

[x_min, x_max] = PDE_domain(); % Call function PDE_domain
U = [0 PDE_u(x_min,0) U PDE_u(x_max,0) 0];
q_x = 0.5*(U(1,4:T_h.N+1) - U(1,2:T_h.N-1)) / T_h.h;
temp = abs(PDE_H(q_x(1,1:T_h.N-2), U(1,3:T_h.N), T_h.mesh(1,3:T_h.N)));
value = max(temp); % ||H(U)|| infinity norm
return;

(8) FD_newton_solver.m

function [U, value, F_FD_newton] = FD_newton_solver(T_h, U_interior, Solver_newton)

U_interior = U_interior(3:T_h.N);
F_FD_newton = @(V) f(V, T_h

options = optimset('Display','iter','TolFun',Solver_newton.tolerance,...
'TolX',Solver_newton.tolerance,'MaxIter',Solver_newton.MaxIt);

[U_interior, value] = fsolve(F_FD_newton, U_interior,options

U = [0,0,U_interior,0,0];
U(2) = PDE_u(T_h.mesh(2),0);
U(1) = 3.0*U(1,2) - 3.0*U(1,3) + U(1,4);
U(T_h.N+1) = PDE_u(T_h.mesh(T_h.N+1),0);
U(T_h.N+2) = 3.0*U(1,T_h.N+1) - 3.0*U(1,T_h.N) + U(1,T_h.N-1);
return;

function values = f(U, T_h)
% extended mesh which is fixed in every iteration

U_extended = zeros(1,T_h.N+2); % extended from interior
U_extended(3:T_h.N) = U;

U_extended(2) = PDE_u(T_h.mesh(2),0);
U_extended(1) = 3.0*U_extended(2) - 3.0*U_extended(3) + U_extended(4);
U_extended(T_h.N+1) = PDE_u(T_h.mesh(T_h.N+1),0);
U_extended(T_h.N+2) = 3.0*U_extended(1,T_h.N+1) - 3.0*U_extended(1,T_h.N) + U_extended(1,T_h.N-1);

% Enforce PDE on interior and at right endpoint

hx2 = T_h.h*T_h.h; % h^2
q_plus = (U_extended(1,4:T_h.N+1) - U_extended(1,3:T_h.N))/T_h.h;
q_minus = (U_extended(1,3:T_h.N) - U_extended(1,2:T_h.N-1))/T_h.h;
q_right = 0.5*(-1.0*U_extended(1,5:T_h.N+2) + 4.0*U_extended(1,4:T_h.N+1) - 3.0*U_extended(1,3:T_h.N))/T_h.h;
q_left = 0.5*(3.0*U_extended(1,3:T_h.N) - 4.0*U_extended(1,2:T_h.N-1) + 1.0*U_extended(1,1:T_h.N-2))/T_h.h;

q_tilda = (q_right + q_left)/6 + (q_plus + q_minus)/3;
Ph_x = (U_extended(1,2:T_h.N-1) - 2.0*U_extended(1,3:T_h.N) + U_extended(1,4:T_h.N+1))/hx2;

% Enforce high order BC, Dirichlet BC, PDE, PDE_BC, high order BC
values = F_hat(Ph_x, q_left, q_right, q_tilda, U, T_h.mesh(1,3:T_h.N), T_h.alpha, T_h.epsilon);
return;

(9) F_hat.m

function out = F_hat(Ph_x, q_left, q_right, q_tilda, U, mesh, a, epsilon)
out = -epsilon*Ph_x + PDE_H(q_tilda, U, mesh);
out = out + a*(q_left - q_right);
return;

function U = FD_sobolev_solver(T_h, U_guess, Solver_sobolev)
N = T_h.N-2;
U = U_guess;
for n = 1:Solver_sobolev.MaxIt
    hx2 = T_h.h*T_h.h; % h^2
    q_plus = (U(1,4:T_h.N+1) - U(1,3:T_h.N))/T_h.h;
    q_minus = (U(1,3:T_h.N) - U(1,2:T_h.N-1))/T_h.h;
    q_right = 0.5*(-1.0*U(1,5:T_h.N+2) + 4.0*U(1,4:T_h.N+1) - 3.0*U(1,3:T_h.N))/T_h.h;
    q_left = 0.5*(3.0*U(1,3:T_h.N) - 4.0*U(1,2:T_h.N-1) + 1.0*U(1,1:T_h.N-2))/T_h.h;

\[
q_{\tilde{t}} = \frac{(q_{\text{right}} + q_{\text{left}})}{6} + \frac{(q_{\text{plus}} + q_{\text{minus}})}{3};
\]

\[
Ph_x = \frac{(U(1, 2:T_h.N-1) - 2.0*U(1, 3:T_h.N) + U(1, 4:T_h.N+1))}{hx2}
\]

\[
\text{RHS} = -Ph_x - \text{Solver_sobolev.rho*F_hat(Ph_x, q_{\text{left}}, q_{\text{right}}, q_{\tilde{t}})}\text{ilda, U(1, 3:T_h.N), T_h.mesh(1, 3:T_h.N), T_h.alpha, T_h.epsilon)};
\]

\[
d = 2*\text{ones(N, 1)}; \text{% construct diagonal elements}
\]

\[
u = -1*\text{ones(N, 1)}; \text{% construct supper diagonal elements}
\]

\[
u(N) = 0;
\]

\[
l = -1*\text{ones(N, 1)}; \text{% construct sub diagonal elements}
\]

\[
l(1) = 0;
\]

\[
\text{RHS} = hx2*\text{RHS};
\]

\[
\text{RHS}(1) = \text{PDE}_u(T_h.mesh(2), 0);
\]

\[
\text{RHS}(N) = \text{PDE}_u(T_h.mesh(T_h.N+1), 0);
\]

\[
U(3:T_h.N) = \text{trisolve}(l, d, u, \text{RHS});
\]

\[
U(2) = \text{PDE}_u(T_h.mesh(2), 0);
\]

\[
U(1) = 3.0*U(1, 2) - 3.0*U(1, 3) + U(1, 4);
\]

\[
U(T_h.N+1) = \text{PDE}_u(T_h.mesh(T_h.N+1), 0);
\]

\[
U(T_h.N+2) = 3.0*U(1, T_h.N+1) - 3.0*U(1, T_h.N) + U(1, T_h.N-1);
\]

\[
end
\]

\[
end
\]

\[
function \text{ out = F_hat(Ph_x, q_{\text{left}}, q_{\text{right}}, q_{\tilde{t}})}\text{ilda, U, mesh, a, epsilon)}
\]

\[
\text{out} = -\text{epsilon*Ph}_x + \text{PDE}_H(q_{\tilde{t}}\text{ilda, U, mesh)};
\]

\[
\text{out} = \text{out} + a*(q_{\text{left}}-q_{\text{right}});
\]

\[
end
\]

\[
(10) \quad \text{trisolve.m}
\]

\[
function \text{[soln] = trisolve(l, d, u, b)}
\]

\[
n = \text{length(d)};
\]

\[
\text{soln} = \text{zeros(1, n)};
\]

\[
\text{for i=2:n}
\]
\[ d(i) = d(i) - u(i-1) \frac{l(i)}{d(i-1)}; \]
\[ b(i) = b(i) - b(i-1) \frac{l(i)}{d(i-1)}; \]

end

\[ \text{soln}(n) = b(n) / d(n); \]

for \( i = 1 : (n-1) \)
  \[ \text{soln}(n-i) = (b(n-i) - u(n-i) \times \text{soln}(n-i+1)) / d(n-i); \]
end

(11) \textbf{Sobolev.m}

\texttt{function \ sobolev\_out = sobolev(U,P2h\_x,out,R)
  sobolev\_out = -(P2h\_x^2) \times U - R \times out;
end}

(12) \textbf{plot.m}

\texttt{figure;
[x\_min, x\_max] = PDE\_domain();
N = length(U);
U\_extended = U2;

% plot approximation
plot(T\_h.mesh(1,2:T\_h.N+1), U\_extended(1,1:T\_h.N), 'g.', 'MarkerSize', 12); hold on;

% plot exact solution
u\_ex = @(x) PDE\_u(x,0);
plot(T\_h.mesh(1,2:T\_h.N+1),u\_ex(T\_h.mesh(1,2:T\_h.N+1)), 'k-', 'MarkerSize', 6); hold off;
h1 = legend('u\_N', 'u');

title(['Test with N = ', num2str(T\_h.N) ' using gamma = ', num2str(T\_h.alpha)]);
xlim([T\_h.mesh(1,2) T\_h.mesh(1,T\_h.N+1)]);
%ylim([-0.1 6.1]);

xlabel('x');
set(h1,'Location','SouthEast');
set(h1,'Interpreter','none')

(13)  tests.m

clc;
clear;

Nx_tests = [10,20,40,80,160,320,640];

Steps = length(Nx_tests);
[T_h, Solver_newton,Solver_sobolev, U_guess] = approximation_parameters();

alpha = T_h.alpha;
epsilon = T_h.epsilon;

errors = zeros(2,Steps);

for k = 1:Steps
    N_x = Nx_tests(k);
    [T_h, U_guess] = initialize(N_x);
    T_h.alpha = alpha;
    T_h.epsilon = epsilon*T_h.h*T_h.h*T_h.h;

    file_name = ['a_out_Nx_ ' num2str(T_h.N) '_a_ ' num2str(T_h.alpha) '.txt'];
    file_out = fopen(file_name,'w');

    % Display Approximation Parameters
    fprintf('
Solving for : 
');
    fprintf('tN_x = %i, h = %d
', T_h.N, T_h.h);
    fprintf(file_out,'Solving for : 
');
    fprintf(file_out,'tN_x = %i, h = %d
', T_h.N, T_h.h);
    fprintf(file_out,'talpa = %d
', T_h.alpha);
    fprintf(file_out,'tMaxIt.fsolve = %i, tol = %d
', Solver_newton.MaxIt, Solver_newton.tolerance);

    % Solver_sobelev
fprintf(file_out,'Sobolev algorithm:
');

fprintf('Sobolev Algorithm:
');

U = FD_sobolev_solver(T_h, U_guess, Solver_sobolev);
sup_error = max(abs(U(1,3:T_h.N) - PDE_u(T_h.mesh(1,3:T_h.N),0)));

fprintf('Error for Sobolev:  %d.
',sup_error);

U2=U;
plotf;

% Solver using the Newton Algorithm

fprintf(file_out,'Newton Algorithm:
');

fprintf('Newton Algorithm:
');

[U2, value, F_FD_newton] = FD_newton_solver(T_h, U, Solver_newton);
sup_error = max(abs(U2(1,3:T_h.N) - PDE_u(T_h.mesh(1,3:T_h.N),0)));

fprintf('Error for Newton:  %d.
',sup_error);

fsolve_value = norm(value);
    if (fsolve_value > 1.e-12)
        [U2, value, ~] = FD_newton_solver(T_h, U2, Solver_newton);
    end

fsolve_value = norm(value);

fprintf(file_out,'\nApproximation Results : \n\nt|fsolve_value| = %d\n',fsolve_value);
fprintf('\nApproximation Results : \n\nt|fsolve_value| = %d\n',fsolve_value);

pde_value = F_norm(T_h, U2);

fprintf(file_out,'\t|F| = %d\n',pde_value);
fprintf('\t|F| = %d\n',pde_value);

fprintf(file_out,'\ti^infty Error = %d',sup_error);
fprintf('\ti^infty Error = %d',sup_error);

fprintf(file_out,'\nNewton Algorithm Finished.
');

fprintf('Newton Algorithm Finished.
');

fprintf('Plotting...
');
plotf;
fclose(file_out);

errors(1,k) = T_h.h;
errors(2,k) = sup_error;
clear 'T_h';
clear 'U2';
end

file_name = 'a_tex.txt';

fid_tex = fopen(file_name,'w');

fprintf(fid_tex, '\begin{tabular}{| c | c | c |}
\hline
\n');
fprintf(fid_tex, '\t h & Error & Order \\
\hline
');
fprintf(fid_tex, '\n');
for i = 2:Steps
    order = log(errors(2,i-1) / errors(2,i)) / log(errors(1,i-1) / errors(1,i));
    fprintf(fid_tex, '\t %1.2e & %1.2e & %1.2f \\
');
end
fprintf(fid_tex, '\end{tabular}
');
fclose(fid_tex);

fclose(file_out);