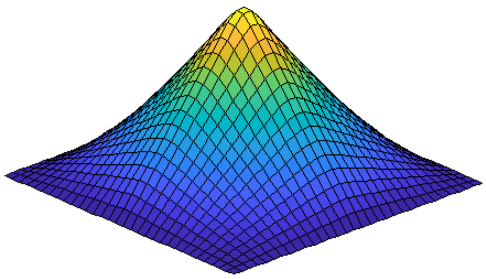
MATH 714: COMPUTATIONAL MATH I
FINAL PROJECT.

SOME FINITE DIFFERENCE METHODS

FOR STATIC HAMILTON–JACOBI EQUATIONS

SON TU

December 20, 2017



Contents

1 Hamilton-Jacobi equations and viscosity solutions

We are interested in solving

$$\begin{cases} H(x, \nabla u(x)) = f(x) & \text{for } x \in \Omega, \\ u(x) = g(x) & \text{for } x \in \partial\Omega \end{cases} \tag{PDE}$$

where Ω is an open set the Hamiltonian H is a nonlinear Lipschitz continuous function. We introduce the function $F : \Omega \times \mathbb{R} \times \mathbb{R}^d \longrightarrow \mathbb{R}$ which we define as

$$F(x,r,p) = \begin{cases} H(x,p) - f(x) & x \in \Omega, \\ r - g(x) & x \in \partial\Omega. \end{cases}$$

We say $u \in C^1(\Omega)$ is a solution of (PDE) if $F[u](x) = F(x,u(x),\nabla u(x)) = 0$ for all $x \in \Omega$. However we won't always have classical solutions which motives the definition of viscosity solutions. They were introduced in [1], via the upper and lower semicontinuous envelopes of a function

$$u^*(x) = \limsup_{y \longrightarrow x} u(y) \qquad \text{and} \qquad u_*(x) = \liminf_{y \longrightarrow x} u(y)$$

It's easy to see that for F^* and F_* we have

$$\begin{cases} F*(x,r,p) = F^*(x,r,p) = H(x,p) - f(x) & x \in \Omega, \\ F_*(x,r,p) = \min\{H(x,p), r - g(x)\} & x \in \partial\Omega, \\ F^*(x,r,p) = \max\{H(x,p), r - g(x)\} & x \in \partial\Omega. \end{cases}$$

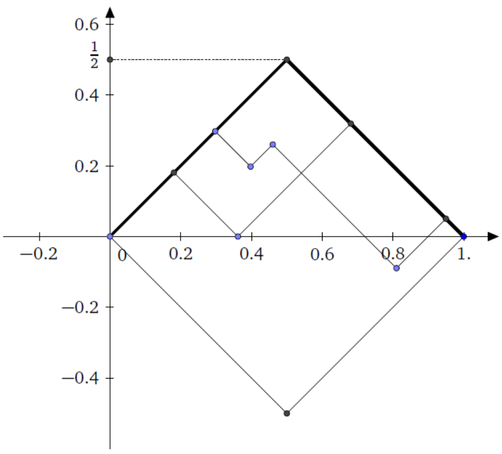
Definition 1 (Viscosity solution). *An upper (lower) semicontinuous function u is a viscosity subsolution (supersolution) of* (PDE) *if for every $\phi \in C^1(\overline{\Omega})$ such that $u - \phi$ has a local maximum (minimum) at $x \in \overline{\Omega}$ then $F_*(x,u(x),\nabla\phi(x)) \leq 0$ ($F^*(x,u(x),\nabla\phi(x)) \geq 0$). A function u is a viscosity solution if it is both a subsolution and supersolution.*

We assume that (PDE) satisfies a comparison principle: if $u \in \text{USC}(\overline{\Omega})$ is a subsolution and $v \in \text{LSC}(\overline{\Omega})$ is a supersolution of (PDE), then $u \leq v$ on $\overline{\Omega}$. The proof for this principle based on the main technical argument `doubling variables` in the viscosity solutions theory [1].

A typical example is the following Eikonal's equation

$$\begin{cases} |u'(x)| & = 1 \qquad \text{on } (-1,1), \\ u(1) = u(-1) & = 0. \end{cases} \tag{1}$$

This equation has no C^1 solution, but it has infinitely many a.e solution. The unique viscosity solution is the biggest one in the following picture. The theory of viscosity solution ensures that the solution has u' cannot change from negative to positive at any point (corner from below is not allowed) and u' changes its sign from positive to negative at only one point.



An approximation scheme is a family of functions parametrized by $h \in \mathbb{R}^+$

$$F^h : \overline{\Omega} \times \mathbb{R} \times L^\infty(\overline{\Omega}) \longrightarrow \mathbb{R}$$

which we write as $F^\delta(x, r, u(\cdot))$. Given a function $u \in L^\infty(\overline{\Omega})$ we write

$$F^h[u](x) = F^h(x, u(x), u(\cdot)). \tag{PDE^h}$$

The function u^h is a solution of the scheme F^h if $F^h[u^h](x) = 0$ for all $x \in \overline{\Omega}$. The idea of solving (PDE) numerically is

1. Construct a scheme F^h such that there exists a stable solution $F^h[u^h] = 0$ to (PDE^h). The existence is usually guaranteed by Banach fixed point theorem.
2. Under certain mild requirements, one can show that $u^h \longrightarrow u$ where u is the unique viscosity solution of (PDE).

2 Monotone schemes for static equations

Let's consider the time dependent equation

$$\begin{cases} v_t(x,t) + H(x, v(x), v'(x)) = f(x) & \text{for } x \in (a,b), t \in (0,\infty), \\ v(x,0) = v_0(x) & \text{for } x \in (a,b), \\ v(x,t) = g(x) & \text{for } x \in \{a,b\}, t \in [0,\infty). \end{cases}$$

with Lipschitz Hamiltonian $H : \mathrm{R} \times \mathbb{R} \longrightarrow \mathbb{R}$. Assume that as $t \longrightarrow \infty$, the solution of this problem approaches a stable or equilibrium state $v(x,t) \longrightarrow u(x)$, then we recover the static problem

$$\begin{cases} H(x, u(x), u'(x)) = f(x) & \text{for } x \in (a,b), \\ u(x) = g(x) & \text{for } x \in \{a,b\}. \end{cases}$$

We use the uniform mesh for discretization $x_i = i\Delta x$ where $i = 0, 1, 2, \dots, m+1$, i.e., $\Delta x = \frac{b-a}{m+1}$. The corresponding grid function is a vector defined as grid points $u_i = u(x_i)$ for $i = 0, \dots, m+1$ with m unknowns are $u_1, \dots, u_m$. Discretizing the derivative $v'(x)$ by

$$u'(x_i) \approx \frac{Du_i^+ + Du_i^-}{2} \qquad \text{where} \qquad Du_i^+ = \frac{u_{i+1} - u_i}{\Delta x}, \qquad Du_i^- = \frac{u_i - u_{i-1}}{\Delta x}$$

are forward and backward Euler approximation of the first derivative. The first order numerical Hamiltonian is of the form $\widehat{H}\left(x_i, u_i, Du_i^+, Du_i^-\right)$ which is a Lipschitz continuous function with respect to all of its arguments and is consistent with the original Hamiltonian in the sense that $\widehat{H}(x, u, p, p) = H(x, u, p)$. From the numerical Hamiltonian, we define the following scheme

$$F^i[u] \equiv F(u_i, u_i - u_{i+1}, u_i - u_{i-1}) = H\left(u_i, -\frac{u_i - u_{i+1}}{\Delta x}, \frac{u_i - u_{i-1}}{\Delta x}\right) - f_i \qquad (i = 1, 2, \dots, m)$$

is a function of $p_{+1} = u_i - u_{i+1}$ and $p_{-1} = u_i - u_{i-1}$ (we suppress the explicit dependence on Δx and x_i). Such a scheme is monotone if each component F^i is nondecreasing with respect to each variable u_i, $u_i - u_{i+1}$ and $u_i - u_{i-1}$. In particular, within our notations, the scheme $F(u, p_{+1}, p_{-1})$ is monotone if $H(u, \alpha, \beta)$ is non-increasing in α and non-decreasing in β and u, denoted by $F(\uparrow, \uparrow, \uparrow) \sim H(\uparrow, \downarrow, \uparrow)$.

2.1 Existence, uniqueness and comparison principle for schemes

When talking about a grid function, we always mean $u = (u_1, \ldots, u_m)$, where we suppressed the indexes u_0, u_{m+1} which are fixed by the boundary conditions.

Definition 2.

- *The finite different scheme F is Lipschitz continuous if there exists a constant K such that for all $i = 1, 2, \ldots, m$ and $\overline{p} = (p_1, p_2, p_3)$ and $\overline{q} = (q_1, q_2, q_3)$ we have*

$$\left|F^i(\overline{p}) - F^i(\overline{q})\right| \leq K\|\overline{p} - \overline{q}\|_\infty.$$

- *The explicit Euler map with time step ρ of the differential equation $\frac{du}{dt} + F[u] = 0$ is $S_\rho : \mathbb{R}^m \longrightarrow \mathbb{R}^m$ maps $u \longmapsto u - \rho F[u]$.*
- *(Nonlinear CFL condition) For F be a Lipschitz continuous, monotone scheme with Lipschitz constant K, the nonlinear CFL condition for the Euler map S_ρ is*

$$\rho K \leq 1. \tag{CFL}$$

- *Given $u, v \in \mathbb{R}^m$, we define*

$$u \vee v = \max\{u, v\}, \qquad u^+ = \max\{u, 0\}, \qquad u^- = \min\{u, 0\}$$

  component-wise and $u \leq v$ means $u_i \leq v_i$ for $i = 0, 1, 2, \ldots, m, m+1$.
- *A finite difference scheme is proper if there exists $\delta > 0$ such that for $i = 1, 2, \ldots, m$ and for all $x \in \mathbb{R}^2$ and $x_0, y_0 \in \mathbb{R}$ then*

$$x_0 \leq y_0 \qquad \Longrightarrow \qquad F^i(x_0, x) - F^i(y_0, x) \leq \delta(x_0 - y_0).$$

Theorem 3 (Comparison principle for schemes). *Let F be a proper, monotone scheme. If $F[u] \leq F[v]$ then $v \leq v$. In particular solutions to the scheme $F[\cdot] = 0$ are unique.*

Proof. Suppose that $u \leq v$ is not true, there exists an index $i \in \{1, 2, \ldots, m\}$ such that

$$u_i - v_i = \max_{j=1,\ldots,m} \{u_j - v_j\} > 0 \qquad \Longrightarrow \qquad u_i - u_j \geq v_i - v_j \quad \text{for all} \quad j = 1, 2, \ldots, m$$

here we understood $u_0 = v_0, u_{m+1} = v_{m+1}$. Since $F(\uparrow, \downarrow, \uparrow)$, we obtain

$$\begin{aligned} F^i[u] = F\big(u_i, u_{i+1} - u_i, u_i - u_{i-1}\big) &\geq F\big(u_i, v_{i+1} - v_i, v_i - v_{i-1}\big) && \text{(by monotonicity)} \\ &> F\big(v_i, v_{i+1} - v_i, v_i - v_{i-1}\big) = F^i[v] && \text{(since } F \text{ is proper)} \end{aligned}$$

which is a contradiction to $F[u] \leq F[v]$. Uniqueness follows obviously. □

Theorem 4 (Ordered Lipschitz continuity property). *Let F be a Lipschitz continuous, monotone with Lipschitz constant K. Then for $i = 1, 2, \ldots, m$ and $x, y \in \mathbb{R}^3$ we have*

$$-K\left\|(x - y)^-\right\|_\infty \leq F^i(x) - F^i(y) \leq K\left\|(x - y)^+\right\|_\infty.$$

Proof. By monotonicity, we have

$$F^i(x) - F^i(y) \le F^i(x \vee y) - F^i(y) \le K\|x \vee y - y\|_\infty = K\|(x-y)^+\|_\infty$$

and the other side can be obtained similarly. ☐

Theorem 5 (Monotonicity of the Euler map). *Let F be a Lipschitz continuous, monotone with Lipschitz constant K, Then the Euler map S_ρ is monotone provided (CFL) holds.*

Proof. Suppose $u \le v$, for an index $i \in \{1, \dots, m\}$ we have

$$\begin{aligned} S^i_\rho[u] - S^i_\rho[v] &= u_i - v_i + \rho\left[F^i(v_i, v_i - v_{i+1}, v_i - v_{i-1}) - F^i(u_i, u_i - u_{i+1}, u_i - u_{i-1})\right] \\ &\le u_i - v_i + \rho K \left\| \underbrace{(v_i - u_i, v_i - u_i + u_{i+1} - v_{i+1}, v_i - u_i + u_{i-1} - v_{i-1})^+}_{\vec{\alpha}} \right\|_\infty . \end{aligned}$$

Observe that $u \le v$ implies $v_i - u_i \ge 0, v_{i+1} - u_{i+1} \ge 0, v_{i-1} - u_{i-1} \ge 0$ and

$$\vec{\alpha} = \left(v_i - u_i, \max\left\{v_i - u_i - (v_{i+1} - u_{i+1}), 0\right\}, \max\left\{v_i - u_i - (v_{i-1} - u_{i-1}), 0\right\}\right)$$

which implies that

$$\|\vec{\alpha}\|_\infty \le v_i - u_i \qquad \Longrightarrow \qquad S^i_\rho[u] - S^i_\rho[v] \le (1 - \rho K)(u_i - v_i) \le 0$$

by the (CFL) condition. ☐

Theorem 6 (The Euler map is a contraction in l^∞). *Let F be a Lipschitz continuous, proper monotone scheme. Then the Euler map is a strict contraction in $\mathbb{R}^m$ equipped with the max norm, provided (CFL) holds.*

Proof. Assume $u_i \ge v_i$, we first show the lower bound

$$\begin{aligned} S^i_\rho[u] - S^i_\rho[v] &= u_i - v_i - \rho\left[F^i\big(u_i, u_i - u_{i+1}, u_i - u_{i-1}\big) - F^i\big(v_i, v_i - v_{i+1}, v_i - v_{i-1}\big)\right] \\ &\ge u_i - v_i - \rho K \left\| (v_i - u_i, v_i - u_i + u_{i+1} - v_{i+1}, v_i - u_i + u_{i-1} - v_{i-1})^+ \right\|_\infty \\ &= u_i - v_i - \rho K \left\| \underbrace{\big(u_i - v_i, u_i - v_i - (u_{i+1} - v_{i+1}), u_i - v_i - (u_{i-1} - v_{i-1})\big)^-}_{\vec{\beta}} \right\|_\infty \end{aligned}$$

where we use the fact that for $w \in \mathbb{R}^m$ then

$$-(-w)^- = w^+ \qquad \Longrightarrow \qquad \|(-w)^-\|_\infty = \|w^+\|_\infty .$$

Observe that $u_i \ge v_i$ implies

$$\vec{\beta} = \left(0, \min\left\{u_i - v_i - (u_{i+1} - v_{i+1}), 0\right\}, \min\left\{u_i - v_i - (u_{i-1} - v_{i-1}), 0\right\}\right).$$

which implies that for $j = i+1$ or $j = i-1$ then

$$\|\vec{\beta}\|_\infty \le \left|u_i - v_i - (u_j - v_j)\right| \le |u_i - v_i| + \|u - v\|_\infty = u_i - v_i + \|u - v\|_\infty .$$

Thus

$$\begin{aligned} S^i_\rho[u] - S^i_\rho[v] &\ge (u_i - v_i) - \rho K\Big((u_i - v_i) + \|u - v\|_\infty\Big) \\ &= (1 - \rho K)(u_i - v_i) - \rho K\|u - v\|_\infty \ge -\rho K\|u - v\|_\infty \end{aligned}$$

since $u_i \geq v_i$. For the upper bound we proceed as following

$$\begin{aligned}
S^i_\rho[u] - S^i_\rho[v] &= u_i - v_i - \rho\Big[F^i(u_i, u_i - u_{i+1}, u_i - u_{i-1}) - F^i(v_i, u_i - u_{i+1}, u_i - u_{i-1})\Big] \\
&\quad + \rho\Big[F^i(v_i, v_i - v_{i+1}, v_i - v_{i-1}) - F^i(v_i, u_i - u_{i+1}, u_i - u_{i-1})\Big] \\
&\leq (1 - \rho\delta)(u_i - v_i) + \rho \underbrace{\Big[F^i(v_i, v_i - v_{i+1}, v_i - v_{i-1}) - F^i(v_i, u_i - u_{i+1}, u_i - u_{i-1})\Big]}_{\gamma}
\end{aligned}$$

by monotonicity. Now by Lipschitz property of F^i we have

$$\begin{aligned}
\gamma &\leq K \left\| \Big((u_{i+1} - v_{i+1}) - (u_i - v_i), (u_{i-1} - v_{i-1}) - (u_i - v_i)\Big)^+ \right\|_\infty \\
&\leq K \left|(u_j - v_j) - (u_i - v_i)\right| \leq K\Big((u_j - v_j) - (u_i - v_i)\Big) \leq K\Big(\|u - v\|_\infty - (u_i - v_i)\Big)
\end{aligned}$$

for $j = i - 1$ or $j = i + 1$. Thus we obtain the upper bound

$$\begin{aligned}
S^i_\rho[u] - S^i_\rho[v] &\leq (1 - \rho\delta)(u_i - v_i) + \rho K\Big(\|u - v\|_\infty - (u_i - v_i)\Big) \\
&\leq (1 - \rho\delta - \rho K)(u_i - v_i) + \rho K\|u - v\|_\infty \leq (1 - \rho\delta)\|u - v\|_\infty
\end{aligned}$$

if we choose ρ small such that $\rho\delta, \rho K \leq \frac{1}{2}$. This prove our estimate

$$\left\|S_\rho[u] - S_\rho[v]\right\|_\infty \leq r\|u - v\|_\infty$$

where $r = \min\{\rho K, 1 - \rho\delta\}$. □

Combine all of these fact above, we obtain the following theorem about the existence of solution to the scheme.

Theorem 7 (Existence of solution to the scheme). *A proper, Lipschitz continuous monotone scheme has a unique solution.*

Proof. Since S_ρ is a strict contraction on $\mathbb{R}^m$ provided (CFL) holds, the iterates of the Euler map converge to the unique fixed point, which is a solution for arbitrary initial data, by the Banach's fixed point theorem. □

An important remark: if a scheme is not proper, we can consider instead $F[u] + \varepsilon u$. By taking ε small enough, we can assume the scheme is proper without loss of generality.

Theorem 8 (Crandall-Lions, [2]). *Monotone schemes are stable and convergent (to the viscosity solution) in the l^∞ norm, with the error estimate is at least half order $\mathcal{O}(\sqrt{\Delta x})$.*

Here we mean by stability in l^∞ norm that the Euler map $u \longmapsto S_\rho[u]$ is non-expansive in l^∞ norm, i.e., $\|S_\rho[u]\|_\infty \leq \|u\|_\infty$.

2.2 The Lax-Friedrich scheme

We consider the problem

$$\begin{cases} H(x, u'(x)) = f(x) & \text{for } x \in (a, b), \\ u(x) = g(x) & \text{for } x \in \{a, b\}. \end{cases}$$

with $H : \mathbb{R}^2 \longrightarrow \mathbb{R}$ is Lipshitz continuous with constant K. The Lax-Friedrich scheme is the one associated with the Lax-Friedrich numerical Hamiltonian

$$H^h_{LF}[u](x) = \widehat{H}(p^+, p^-) = H\left(x, \frac{p^+ + p^-}{2}\right) - \frac{1}{2}\sigma_x\left(p^+ - p^-\right)$$

where σ_x to be chosen such that this scheme is monotone, $p = u_x$ and $p^\pm$ are the corresponding forward and backward differences approximations of u_x, and $h = \Delta x$ is the step size. Within our notation, the Lax-Friedrich scheme is monotone if H is non-increasing in p^+ and non-decreasing in p^-. Indeed, if $p_1^+ \geq p_2^+$ then

$$\begin{aligned}\widehat{H}(p_1^+, p^-) - \widehat{H}(p_2^+, p^-) &= H\left(x, \frac{p_1^+ + p^-}{2}\right) - H\left(x, \frac{p_2^+ + p^-}{2}\right) - \frac{1}{2}\sigma_x\left(p_1^+ - p_2^+\right) \\ &\leq \frac{K - \sigma_x}{2}\left(p_1^+ - p_2^+\right) \leq 0\end{aligned}$$

provided $\sigma_x \geq K$. The other direction in p^- can be done similarly. Define $\Delta^+ u_i = u_i - u_{i+1}$ and $\Delta^- u_i = u_i - u_{i-1}$ then

$$\begin{aligned}\left|F\left(\Delta^- u_i, \Delta^+ u_i\right) - F\left(\Delta^- v_i, \Delta^+ v_i\right)\right| &= \left|H\left(x_i, \frac{\Delta^- u_i - \Delta^+ u_i}{2\Delta x}\right) - H\left(x_i, \frac{\Delta^- v_i - \Delta^+ v_i}{2\Delta x}\right)\right| \\ &\leq \frac{K}{\Delta x}\left(\left|\frac{\Delta^- u_i - \Delta^+ u_i}{2}\right| + \left|\frac{\Delta^- v_i - \Delta^+ v_i}{2}\right|\right) \\ &\leq \frac{K}{\Delta x}\left\|\left(\Delta^- u_i, \Delta^+ u_i\right) - \left(\Delta^- v_i, \Delta^+ v_i\right)\right\|_\infty.\end{aligned}$$

Thus the nonlinear (CFL) condition concludes that the Euler map $S_\rho[u] = u - \rho F[u]$ is monotone if

$$\rho \leq \frac{\Delta x}{K}.$$

Theorem 8 concludes that the Lax-Friedrich schemes work well. Let's illustrate it by writing down explicitly the update formula

$$H\left(x_i, \frac{u_{i+1} - u_i}{2\Delta x}\right) - \frac{\sigma_x}{2}\left(\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x}\right) = f(x_i)$$

which results in the iteration formula (we don't follow the Euler iteration, which is more complicated)

$$u_i^{\text{new}} = \frac{\Delta_x}{\sigma_x}\left(f(x_i) - H\left(x_i, \frac{u_{i+1}^{\text{old}} - u_{i-1}^{\text{old}}}{2\Delta x}\right)\right) + \frac{u_{i+1}^{\text{old}} + u_{i-1}^{\text{old}}}{2}. \tag{2}$$

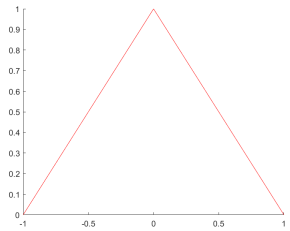
2.3 Implementation in one dimension

Example 2.1 (Classical Eikonal's equation).

$$\begin{cases} |u'(x)| &= 1 \quad \text{on } (-1,1), \\ u(1) = u(-1) &= 0. \end{cases} \tag{Ex.2.1}$$

has a unique viscosity solution $u(x) = 1 - |x|$.

Here we choose $\sigma_x = 1.001$ and initial guess $u_0 = 0$. Note that in this example, the solution converges for any $\sigma_x \geq 1$, but in order to obtain the 1-order accuracy in l^∞, we need $\sigma_x > 1$.



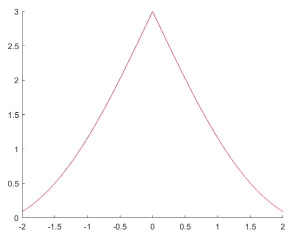
Grid points	l^∞ **Errors**	l^∞ **Accuracy**
64	0.1587×10^{-4}	$----$
128	0.0787×10^{-4}	0.99986
256	0.0392×10^{-4}	0.99999
512	0.0196×10^{-4}	1.00000
1024	0.0098×10^{-4}	0.99980

Example 2.2 (Eikonal's equation).

$$\begin{cases} |u'(x)| & = 1 + \cos x \qquad on\ (-2,2), \\ u(-2) = u(2) & = 3 - |2 + \sin 2|. \end{cases} \tag{Ex.2.2}$$

has a unique viscosity solution $u(x) = 3 - |x + \sin(x)|$.

Here we choose $\sigma_x = 1$ and initial guess $u_0 = 0$.



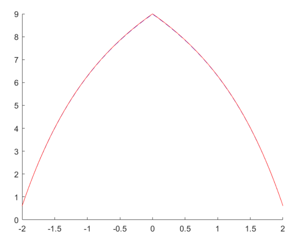
Grid points	l^∞ **Errors**	l^∞ **Accuracy**
64	0.0446	$----$
128	0.0222	0.99374
256	0.0111	0.99531
512	0.0055	0.99645
1024	0.0028	0.99726

Example 2.3 (Eikonal's equation).

$$\begin{cases} |u'(x)| & = 1 + e^{|x|} \qquad on\ (-2,2), \\ u(-2) = u(2) & = 8 - e^2. \end{cases} \tag{Ex.2.3}$$

has a unique viscosity solution $u(x) = 10 - |x| - e^{|x|}$.

Here we choose $\sigma_x = 1$ and initial guess $u_0 = 0$.



Grid points	l^∞ **Errors**	l^∞ **Accuracy**
64	0.1997	$----$
128	0.0998	0.98866
256	0.0499	0.99152
512	0.0250	0.99351
1024	0.0125	0.99492

Example 2.4 (Hamilton-Jacobi equation with convex Hamiltonian).

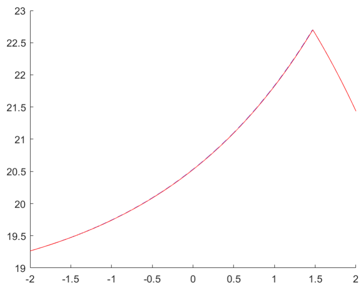
$$\begin{cases} |u'(x)|^2 & = e^x \qquad on\ (-2,2), \\ u(-2) & = -2e + 20, \\ u(2) & = \ \ 2e + 16. \end{cases} \tag{Ex.2.4}$$

This equation has a unique viscosity solution

$$u(x) = \begin{cases} 2e^{\frac{x}{2}} - 4e^{-1} + 20 & x \in [-2, x_0], \\ -2e^{\frac{x}{2}} + 4e + 16 & x \in [x_0, 2]. \end{cases} \qquad \text{where} \qquad x_0 = 2\ln\big(e + e^{-1} - 1\big).$$

Here we choose $\sigma_x = 4.5$ and the initial guess to be the linear function connecting two initial data at -2 and 2. An easier choice is the sub-solution $u_0 \equiv 18$, but we need the artifical viscosity to be very huge to have convergence. Note that if we start from 0 the solution won't converge (requires bigger σ_x). This is a very sensitive case, we choose the tolerant 10^{-9} in this case (not 10^{-12} as usual).

It is easy to see that the solution cannot have corner from below, and will have exactly one corner from above, i.e., there exists $x_0 \in (-2, 2)$ such that u' changes its sign from positive to negative at x_0. We obtain the exact solution from that argument.



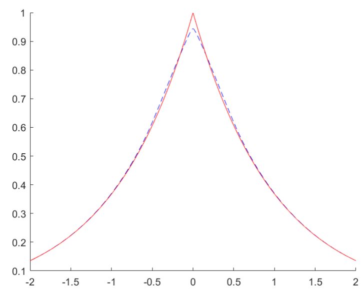
Grid points	l^∞ Errors	l^∞ Accuracy
64	0.1143	———
128	0.0589	0.94597
256	0.0300	0.95748
512	0.0151	0.96672
1024	0.0076	0.97364

Example 2.5 (Non-convex Hamiltonian). $H(x, p) = \cos(p)^2 + |p|$

$$\begin{cases} \cos(u'(x))^2 + |u'(x)| & = \cos(e^{-|x|})^2 + e^{-|x|} \qquad on\ (-2, 2), \\ u(-2) = u(2) & = e^{-2}. \end{cases} \tag{Ex.2.5}$$

has a unique viscosity solution $u(x) = e^{-|x|}$.

Here we choose $\sigma_x = 2$ and initial guess $u_0 = 0$.



Grid points	l^∞ Errors	l^∞ Accuracy
64	0.1328	———
128	0.1095	0.274169
256	0.0886	0.28959
512	0.0704	0.30300
1024	0.0540	0.32147

2.4 Implementation in two dimensions

We consider the problem

$$\begin{cases} H(x, \nabla u(x)) = f(x) & \text{for } (x, y) \in \Omega = [0,1] \times [0,1], \\ u(x) = g(x) & \text{for } (x, y) \in \partial\Omega. \end{cases}$$

with $H : \mathbb{R}^4 \longrightarrow \mathbb{R}$ is Lipshitz continuous with constant K. The Lax-Friedrich scheme is the one associated with the Lax-Friedrich numerical Hamiltonian

$$\begin{aligned} H^h_{LF}[u](x,y) &= \widehat{H}(p^+,p^-,q^+,q^-) \\ &= H\left(x,y,\frac{p^+ + p^-}{2},\frac{q^+ + q^-}{2}\right) - \frac{1}{2}\sigma_x\left(p^+ - p^-\right) - \frac{1}{2}\sigma_y\left(q^+ - q^-\right) \end{aligned}$$

where $\sigma_x \geq \max\left|\frac{\partial H}{\partial p}\right|$ and $\sigma_y \geq \max\left|\frac{\partial H}{\partial q}\right|$ such that this scheme is monotone, $p = u_x$ and $p^\pm$ are the corresponding forward and backward differences approximations of u_x, $q = u_y$ and $q^\pm$ are the corresponding forward and backward differences approximations of u_y, with the uniform mesh size. On each side we discretize $x_j = j\Delta x$, where $j = 0, 1, 2, \dots, m+1$ with $\Delta x = \frac{1}{m+1}$. The m^2 unknowns are $u_{ij} = u(x_i, y_j)$ where $1 \leq i, j \leq m$. We look at the update formula

$$\begin{aligned} H\left(x_i, y_j, \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x}, \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y}\right) \\ - \frac{\sigma_x}{2}\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta y} - \frac{\sigma_x}{2}\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y} = f\left(x_{i,j}\right) \end{aligned}$$

which results in the update formula:

$$\begin{aligned} u_{i,j} = \left(\frac{\sigma_x}{\Delta, x} + \frac{\sigma_y}{\Delta y}\right)^{-1}\Bigg[f_{i,j} - H\left(x_i, y_j, \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x}, \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y}\right) \\ + \sigma_x \frac{u_{i+1,j} + u_{i-1,j}}{2\Delta x} + \sigma_y \frac{u_{i,j+1} + u_{i,j-1}}{2\Delta y}\Bigg]. \end{aligned}$$

In case $\Delta x = \Delta y = h$, we have

$$\begin{aligned} u^{\text{new}}_{i,j} = & \frac{h}{\sigma_x + \sigma_y}\left[f_{i,j} - H\left(x_i, y_j, \frac{u^{\text{old}}_{i+1,j} - u^{\text{old}}_{i-1,j}}{2h}, \frac{u^{\text{old}}_{i,j+1} - u^{\text{old}}_{i,j-1}}{2h}\right)\right] \\ & + \frac{\sigma_x}{\sigma_x + \sigma_y}\frac{u^{\text{old}}_{i+1,j} + u^{\text{old}}_{i-1,j}}{2} + \frac{\sigma_y}{\sigma_x + \sigma_y}\frac{u^{\text{old}}_{i,j+1} + u^{\text{old}}_{i,j-1}}{2}. \end{aligned}$$

Remark 9. *The method runs very slow in two dimensions. Smaller artificial viscosity makes convergence faster, but if the solution is very singular then small artificial viscosity may break the convergence.*

Example 2.6 (Eikonal's equation in two dimensions). $H(x,p) = |p| = \sqrt{p_1^2 + p_2^2}$.

$$\begin{cases} |\nabla u| &= 1 \qquad \text{in } \Omega = [-2,2] \times [-2,2], \\ \quad u &= (2 - |(x,y)|)\big|_{\partial\Omega} \qquad \text{on } \partial\Omega. \end{cases} \tag{Ex.2.6}$$

has a unique viscosity solution $u(x,y) = 2 - \|(x,y)\|_2$, *starting with* $u_0 = 0$.

Here we choose $\sigma_x = \sigma_y = 1 + 10^{-12}$ with initial guess $u_0 = 0$.

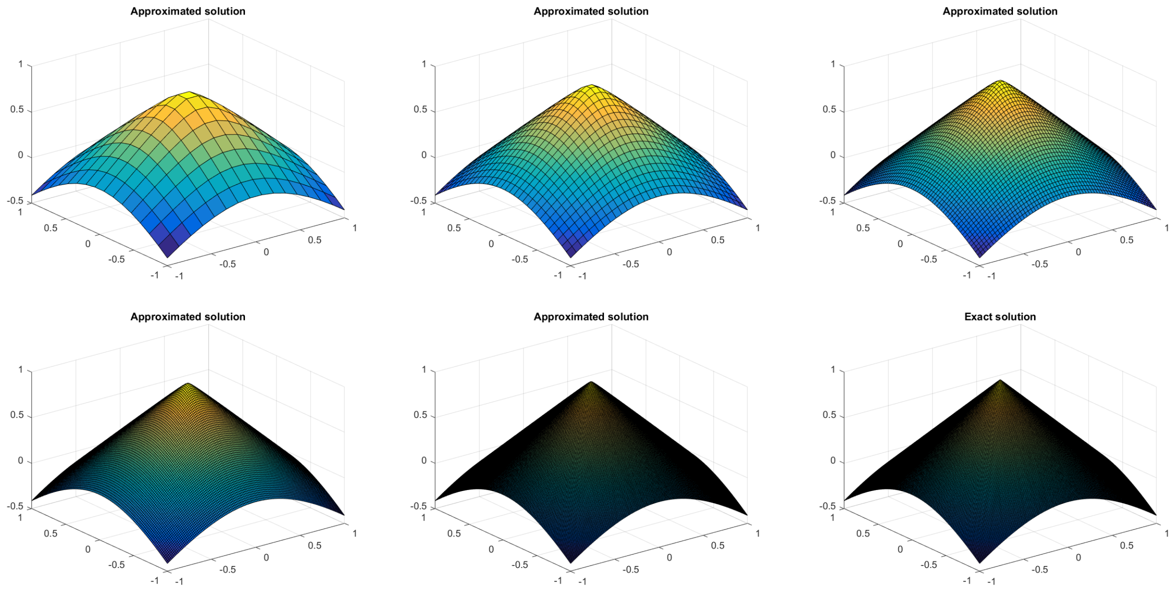
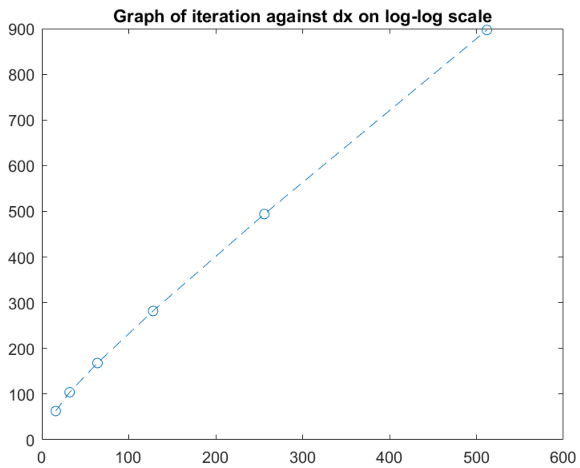
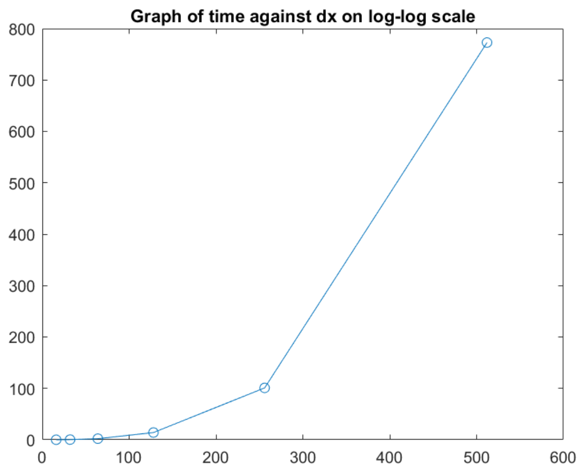


Figure 1: Approximated solutions with $m = 16, 32, 64, 128, 256$.





Grid points	l^∞ Errors	l^∞ Accuracy	Iterations	Elapsed time (s)
16	0.1393	$----$	61	0.0547
32	0.0906	0.5930	100	0.3018
64	0.0558	0.6377	163	2.0774
128	0.0332	0.6726	276	14.4202
256	0.0193	0.7009	485	104.2492
512	0.0110	0.7244	885	794.0576

Example 2.7 (Eikonal's equation in two dimensions). $H(x,p) = |p| = \sqrt{p_1^2 + p_2^2}$.

$$\begin{cases} |\nabla u(x,y)| & = \sqrt{(1-|x|)^2 + (1-|y|)^2} \qquad in\ \Omega = [-1,1]\times[-1,1], \\ u & = 0 \qquad on\ \partial\Omega. \end{cases} \tag{Ex.2.7}$$

has a unique viscosity solution $u(x,y) = (1-|x|)(1-|y|)$.

Here we choose $\sigma_x = \sigma_y = 1 + 10^{-12}$ with initial guess $u_0 = 0$.

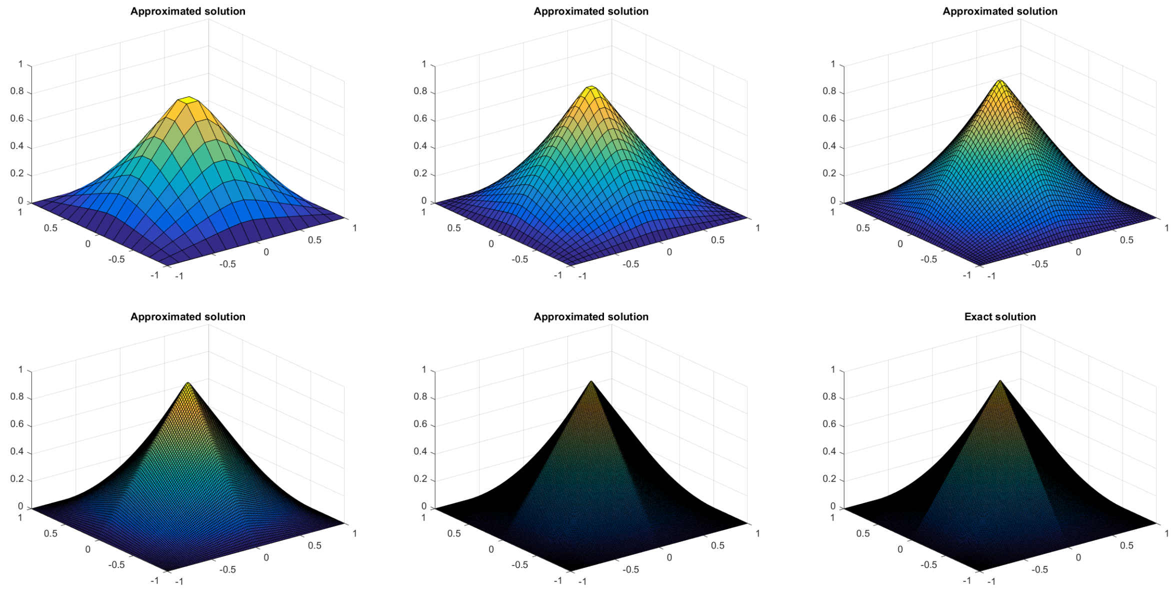
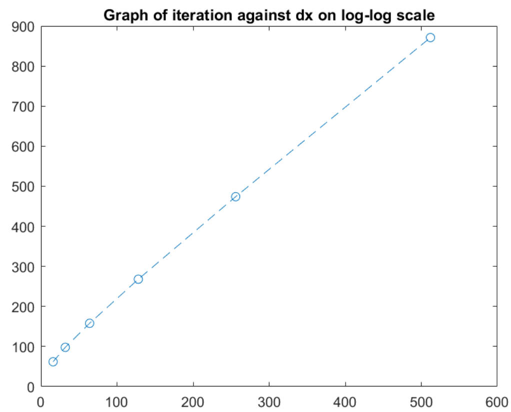
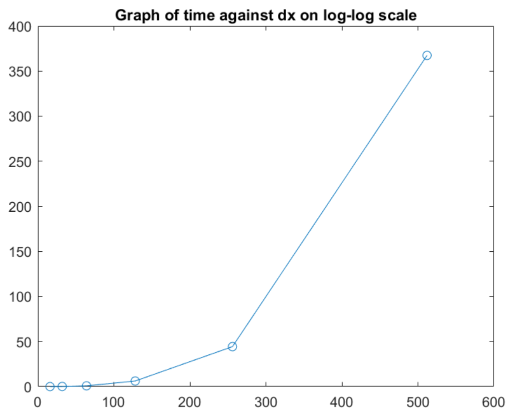


Figure 2: Approximated solutions with $m = 16, 32, 64, 128, 256$.





Grid points	l^∞ Errors	l^∞ Accuracy	Iterations	Elapsed time (s)
16	0.0635	————	60	0.0247
32	0.0371	0.7423	95	0.1221
64	0.0186	0.8550	154	0.8501
128	0.0090	0.9194	261	5.9737
256	0.0044	0.9512	465	44.0847
512	0.0022	0.9647	857	351.2299

3 Lax-Friedrich sweeping: A faster scheme

The idea of sweeping method is istead of updating at each step the new value by entire old value, we keep updateing the new value at u_{i+1} by the new value of previous step u_i. It is similar to Gauss-Seidel iteration method.

3.1 Implementation in one dimension

We consider the problem

$$\begin{cases} H(x, u'(x)) = f(x) & \text{for } x \in (a, b), \\ u(x) = g(x) & \text{for } x \in \{a, b\}. \end{cases}$$

with $H : \mathbb{R}^2 \longrightarrow \mathbb{R}$ is Lipshitz continuous with constant K, $\sigma_x \geq \frac{\partial H}{\partial p}$ as usual. The sweeping formula is slightly different to the monotone one (2)

- Sweeping from the left to the right with $i = 1, 2, \ldots, m$:

$$u_i^{\text{temp}} = \frac{\Delta_x}{\sigma_x}\left(f(x_i) - H\left(x_i, \frac{u_{i+1}^{\text{old}} - u_{i-1}^{\text{temp}}}{2\Delta x}\right)\right) + \frac{u_{i+1}^{\text{old}} + u_{i-1}^{\text{temp}}}{2}.$$

- Sweeping from the right to the left with $i = m, m-1, \ldots, 1$:

$$u_i^{\text{new}} = \frac{\Delta_x}{\sigma_x}\left(f(x_i) - H\left(x_i, \frac{u_{i+1}^{\text{new}} - u_{i-1}^{\text{temp}}}{2\Delta x}\right)\right) + \frac{u_{new}^{\text{old}} + u_{i-1}^{\text{temp}}}{2}.$$

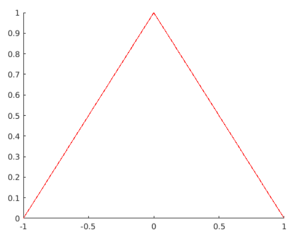
Remark 10. *The method is stable and convergent in l^1 (around 2nd order with nice solution, as the following Eikonal's equation).*

Example 3.1 (Classical Eikonal's equation).

$$\begin{cases} |u'(x)| & = 1 \quad \text{on } (-1, 1), \\ u(1) = u(-1) & = 0. \end{cases} \tag{Ex.3.1}$$

has a unique viscosity solution $u(x) = 1 - |x|$.

Here we choose $\sigma_x = 1 + e^{-12}$ and initial guess $u_0 = 0$. Note that in this example, the solution converges for any $\sigma_x \geq 1$, but in order to obtain the 1-order accuracy in l^1, we need $\sigma_x > 1$.



Grid points	l^1 Errors	l^1 Accuracy
64	0.6192×10^{-8}	————
128	0.1524×10^{-8}	1.99999
256	0.0378×10^{-8}	1.99999
512	0.0094×10^{-8}	1.99999
1024	0.0023×10^{-8}	1.99999

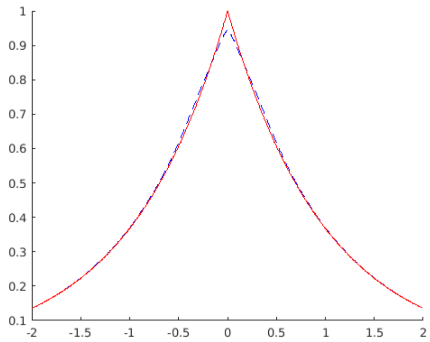
Remark 11. *The method is stable and convergent in l^1 only with 1st order for other examples Ex.2.2, Ex.2.3 and Ex.2.4. But with Ex.2.5 it converges in l^1 with order 0.65, twice faster than the monotone scheme.*

Example 3.2 (Non-convex Hamiltonian). $H(x, p) = \cos(p)^2 + |p|$

$$\begin{cases} \cos(u'(x))^2 + |u'(x)| & = \cos(e^{-|x|})^2 + e^{-|x|} \quad \text{on } (-2, 2), \\ u(-2) = u(2) & = e^{-2}. \end{cases} \tag{Ex.3.5}$$

has a unique viscosity solution $u(x) = e^{-|x|}$.

Here we choose $\sigma_x = 2$ and initial guess $u_0 = 0$.



Grid points	l^1 Errors	l^1 Accuracy
64	0.1339	————
128	0.0853	0.64355
256	0.0535	0.65548
512	0.0336	0.66148
1024	0.0211	0.66445

3.2 Implementation in two dimensions

We consider the problem

$$\begin{cases} H(x, \nabla u(x)) = f(x) & \text{for } (x, y) \in \Omega = [0,1] \times [0,1], \\ u(x) = g(x) & \text{for } (x, y) \in \partial\Omega. \end{cases}$$

with $H : \mathbb{R}^4 \longrightarrow \mathbb{R}$ is Lipshitz with constant K. In case $\Delta x = \Delta y = h$, we sweep 4 times:

- Sweeping $i = 1, \ldots, m$, $j = 1, \ldots, m$.
- Sweeping $i = m, \ldots, 1$, $j = 1, \ldots, m$.
- Sweeping $i = 1, \ldots, m$, $j = m, \ldots, 1$.
- Sweeping $i = 1, \ldots, m$, $j = m, \ldots, 1$.

Remark 12. *The method runs faster comparing to the monotone scheme slow in two dimensions with 2nd order of convergence in l^1 norm.*

Example 3.3 (Eikonal's equation in two dimensions). $H(x, p) = |p| = \sqrt{p_1^2 + p_2^2}$.

$$\begin{cases} |\nabla u| & = 1 \quad \textit{in } \Omega = [-2,2] \times [-2,2], \\ u & = (2 - |(x, y)|)\big|_{\partial\Omega} \quad \textit{on } \partial\Omega. \end{cases} \tag{Ex.3.6}$$

has a unique viscosity solution $u(x, y) = 2 - \|(x, y)\|_2$, starting with $u_0 = 0$.

Here we choose $\sigma_x = \sigma_y = 1 + 10^{-12}$ with initial guess $u_0 = 0$.

Grid points	l^1 Errors	l^1 Accuracy	Iterations	Elapsed time (s)
16	0.0154	————	10	0.0295
32	0.0039	1.8811	14	0.0517
64	0.0010	1.9146	21	0.2938
128	0.0002	1.9371	32	1.7249
256	0.0001	1.9526	59	12.3893
512	0.0000	1.9634	112	100.2518

Example 3.4 (Eikonal's equation in two dimensions). $H(x,p) = |p| = \sqrt{p_1^2 + p_2^2}$.

$$\begin{cases} |\nabla u(x,y)| & = \sqrt{(1-|x|)^2 + (1-|y|)^2} \quad \textit{in } \Omega = [-1,1] \times [-1,1], \\ u & = 0 \quad \textit{on } \partial\Omega. \end{cases} \tag{Ex.3.7}$$

has a unique viscosity solution $u(x,y) = (1-|x|)(1-|y|)$.

Here we choose $\sigma_x = \sigma_y = 1 + 10^{-12}$ with initial guess $u_0 = 0$.

Grid points	l^1 Errors	l^1 Accuracy	Iterations	Elapsed time (s)
16	0.0076	————	10	0.0206
32	0.0023	1.6707	14	0.0455
64	0.0006	1.7140	21	0.2714
128	0.0002	1.7536	33	1.7850
256	0.0000	1.7872	54	11.9170
512	0.0000	1.8150	95	96.0690

4 MatLab code

4.1 MatLab code for 1D case

```
function [XX sol Max_norm_error u_exact] = monotone1D(H,R,L,nx,A,B,sigma,exact_sol,u_init)
u_old = B*ones(1,nx);
x_length = 2*L;
dx = x_length/(nx-1);
XX = -L:dx:L;
u_new = u_init(XX);
error = norm(u_old - u_new,Inf);
u_old(1) = exact_sol(-L); u_old(nx) = exact_sol(L);
u_new(1) = exact_sol(-L); u_new(nx) = exact_sol(L);
 while error > 1.0000e-9 % 1.0000e-12
        u_old = u_new;
        for i = 2:(nx-1)
            u_new(i)  = dx/sigma*(R(XX(i)) - H(XX(i),(u_old(i+1) - u_old(i-1))/(2*dx))) + 1/2*(u_old(i+1) + u_old(i-1));
        end
        error = norm(u_old - u_new,Inf);
 end
u_exact = exact_sol(XX);
sol = u_new;
Max_norm_error = norm(sol - u_exact,Inf);
end
```

```
function [XX sol Max_norm_error u_exact] = sweeping1D(H,R,L,nx,A,B,sigma,exact_sol,u_init)
u_old = B*ones(1,nx);
```

```
x_length = 2*L;
dx = x_length/(nx-1);
XX = -L:dx:L;
u_new = u_init(XX);
error = dx*norm(u_old - u_new,1);
u_old(1) = exact_sol(-L); u_old(nx) = exact_sol(L);
u_new(1) = exact_sol(-L); u_new(nx) = exact_sol(L);
 while error > 1.0000e-12 % 1.0000e-12
        u_old = u_new; u_temp = u_new;
        for i = 2:(nx-1)
            u_temp(i) = dx/sigma*(R(XX(i)) - H(XX(i),(u_old(i+1) - u_temp(i-1))/(2*dx))) + 1/2*(u_old(i+1) + u_temp(i-1));
        end
        for i = (nx-1):-1:2
            u_new(i) =  dx/sigma*(R(XX(i)) - H(XX(i),(u_new(i+1) - u_temp(i-1))/(2*dx))) + 1/2*(u_new(i+1) + u_temp(i-1));
        end
        error = dx*norm(u_old - u_new,1);
 end
u_exact = exact_sol(XX);
sol = u_new;
Max_norm_error = dx*norm(sol - u_exact,1);
end
```

```
clear all
close all
clc % Domain is [-L,L]
% % 1st example %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% L = 1;
% H = @(x,p) abs(p); % R = @(x) 1;
% exact_sol = @(x) 1 - abs(x);
% Upper_Bound = 1;  Lower_Bound = 0;
% u_init = @(x) 0;
% sigma = 1+exp(-12);
% 2nd example %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% L = 2;
% H = @(x,p) abs(p); % R = @(x) 1+cos(x);
% exact_sol = @(x) 3-abs(x+sin(x));
% Upper_Bound = 3;  Lower_Bound = 0;
% u_init = @(x) 0;
% sigma = 1+exp(-12);
% 3rd example %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% L = 2;
% H = @(x,p) abs(p); % R = @(x) 1+exp(abs(x));
% exact_sol = @(x) 10-abs(x)-exp(abs(x));
% Upper_Bound = 10;  Lower_Bound = 0;
% u_init = @(x) 0; % sigma = 1;
% % 4th example %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% L = 2;
```

```
% H = @(x,p) p.^2; % R = @(x) exp(x);
% x0 = 2*log(exp(1)+exp(-1)-1);
% exact_sol = @(x) (x>=-2 & x<=x0).*(-2*exp(-1)+20 + 2*(exp(x/2)-exp(-1)))
     + (x>x0 & x<=2).*(2*exp(1)+16 + 2*(exp(1)-exp(x/2)));
% lambda = @(x) 1/2*(1-x./L);
% u_init = @(x) lambda(x).*exact_sol(-L) + (1-lambda(x)).*exact_sol(L);
% Upper_Bound = 23; Lower_Bound = 18;
% sigma = 4.5;
% 5th example %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L = 2;
H = @(x,p) cos(p).^2 + abs(p);
R = @(x) cos(exp(-abs(x))).^2 + exp(-abs(x));
exact_sol = @(x) exp(-abs(x));
Upper_Bound = 30; Lower_Bound = 0;
u_init = @(x) 0; sigma = 2+exp(-12);
% % Number of grid points
nx_vec = [64 128 256 512 1024];
for mesh = 1:length(nx_vec)
    nx = nx_vec(mesh);
    dx = 2*L/(nx-1);
    %[XX sol norm_error(mesh) u_exact] = monotone1D(H,R,L,nx,Lower_Bound,
       Upper_Bound,sigma,exact_sol,u_init);
    [XX sol norm_error(mesh) u_exact] = sweeping1D(H,R,L,nx,Lower_Bound,
       Upper_Bound,sigma,exact_sol,u_init);
        figure(mesh)
        hold on
        plot(XX,sol,'b--');
        plot(XX,u_exact,'r');
        hold off
    ddx(mesh) = dx;
    new_nx_vec(mesh) = nx_vec(mesh);
    norm_error
    newfigure = figure(mesh+5); % compute the order of accuracy
    set(newfigure,'color','white');
    loglog(ddx,abs(norm_error));
    title('Graph of norm error against dx on log-log scale')
    first_col = ones(length(new_nx_vec),1); second_col = log(ddx)';
    AA = [first_col,second_col];
    FF_accuracy = log(norm_error);
    solution_accuracy = (AA'*AA)^(-1)*AA'*FF_accuracy';
    K = solution_accuracy(1); p = solution_accuracy(2);
    vpa([K,p])
end
```

4.2 MATLAB code for 2D case

```
function [time_need iteration XX YY sol Max_norm_error u_exact] = monotone2D(H,R,L,nx,A,B,sigma_x,sigma_y,exact_sol)
u_old = B*ones(nx,nx);
u_init = A*ones(nx,nx);
x_length = 2*L;
y_length = 2*L;
dx = x_length/(nx-1); dy = dx;
[XX YY] = meshgrid(-L:dx:L,-L:dy:L);
u_exact = exact_sol(XX,YY);
u_new = u_init;
error = max(max(abs(u_old - u_new)));
u_old(1,:) = u_exact(1,:); u_old(nx,:) = u_exact(nx,:);
u_old(:,1) = u_exact(:,1); u_old(:,nx) = u_exact(:,nx);
u_new(1,:) = u_exact(1,:); u_new(nx,:) = u_exact(nx,:);
u_new(:,1) = u_exact(:,1); u_new(:,nx) = u_exact(:,nx);
count = 0;
tic
while error > 1.0000e-12
        count = count + 1;
        u_old = u_new;
        for i = 2:(nx-1)
            for j = 2:(nx-1)
                u_new(i,j)  = dx/(sigma_x+sigma_y)*(R(XX(i,j),YY(i,j)) - H(XX(i,j),YY(i,j),(u_old(i+1,j) - u_old(i-1,j))/(2*dx),(u_old(i,j+1) - u_old(i,j-1))/(2*dy))) + sigma_x/(sigma_x+sigma_y)*1/2*(u_old(i+1,j) + u_old(i-1,j)) + sigma_y/(sigma_x+sigma_y)*1/2*(u_old(i,j+1) + u_old(i,j-1));
            end
        end
        error = max(max(abs(u_old - u_new)));
    end
time_need = toc;
u_exact = exact_sol(XX,YY);
sol = u_new;
Max_norm_error = max(max(abs(u_exact-u_new)));
iteration = count;
end
```

```
function [time_need iteration XX YY sol norm_error u_exact] = sweeping2D(H,R,L,nx,A,B,sigma_x,sigma_y,exact_sol)
u_old = B*ones(nx,nx);
u_init = A*ones(nx,nx);
x_length = 2*L;
y_length = 2*L;
dx = x_length/(nx-1); dy = dx;
[XX YY] = meshgrid(-L:dx:L,-L:dy:L);
u_exact = exact_sol(XX,YY);
u_new = u_init;
```

```
error = dx^2*norm(abs(u_old-u_new),1);
u_old(1,:) = u_exact(1,:); u_old(nx,:) = u_exact(nx,:);
u_old(:,1) = u_exact(:,1); u_old(:,nx) = u_exact(:,nx);
u_new(1,:) = u_exact(1,:); u_new(nx,:) = u_exact(nx,:);
u_new(:,1) = u_exact(:,1); u_new(:,nx) = u_exact(:,nx);
count = 0;
tic
while error > 1.0000e-12
        count = count + 1;
        u_old = u_new;
        u_1 = u_new;
        for i = 2:(nx-1)
            for j = 2:(nx-1)
                u_1(i,j)  = dx/(sigma_x+sigma_y)*(R(XX(i,j),YY(i,j)) - H(XX(i,j),YY(i,j),(u_old(i+1,j) - u_1(i-1,j))/(2*dx),(u_old(i,j+1) - u_1(i,j-1))/(2*dy))) + sigma_x/(sigma_x+sigma_y)*1/2*(u_old(i+1,j) + u_1(i-1,j)) + sigma_y/(sigma_x+sigma_y)*1/2*(u_old(i,j+1) + u_1(i,j-1));
            end
        end
        u_2 = u_1;
        for i = (nx-1):-1:2
            for j = 2:(nx-1)
                u_2(i,j)  = dx/(sigma_x+sigma_y)*(R(XX(i,j),YY(i,j)) - H(XX(i,j),YY(i,j),(u_2(i+1,j) - u_1(i-1,j))/(2*dx),(u_1(i,j+1) - u_2(i,j-1))/(2*dy))) + sigma_x/(sigma_x+sigma_y)*1/2*(u_2(i+1,j) + u_1(i-1,j)) + sigma_y/(sigma_x+sigma_y)*1/2*(u_1(i,j+1) + u_2(i,j-1));
            end
        end
        u_3 = u_2;
        for i = 2:(nx-1)
            for j = (nx-1):-1:2
                u_3(i,j)  = dx/(sigma_x+sigma_y)*(R(XX(i,j),YY(i,j)) - H(XX(i,j),YY(i,j),(u_2(i+1,j) - u_3(i-1,j))/(2*dx),(u_3(i,j+1) - u_2(i,j-1))/(2*dy))) + sigma_x/(sigma_x+sigma_y)*1/2*(u_2(i+1,j) + u_3(i-1,j)) + sigma_y/(sigma_x+sigma_y)*1/2*(u_3(i,j+1) + u_2(i,j-1));
            end
        end
        u_4 = u_3;
        for i = (nx-1):-1:2
            for j = (nx-1):-1:2
                u_4(i,j)  = dx/(sigma_x+sigma_y)*(R(XX(i,j),YY(i,j)) - H(XX(i,j),YY(i,j),(u_4(i+1,j) - u_3(i-1,j))/(2*dx),(u_4(i,j+1) - u_3(i,j-1))/(2*dy))) + sigma_x/(sigma_x+sigma_y)*1/2*(u_4(i+1,j) + u_3(i-1,j)) + sigma_y/(sigma_x+sigma_y)*1/2*(u_4(i,j+1) + u_3(i,j-1));
            end
```

```
        end
        u_new = u_4;
        error = dx^2*norm(abs(u_old-u_new),1);
    end
time_need = toc;
u_exact = exact_sol(XX,YY);
sol = u_new;
norm_error = dx^2*norm(abs(u_exact-u_new),1);
iteration = count;
end
```

```
clear all
close all
clc % Domain is [-L,L]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % 1nd example
% L = 1;
% H = @(x,y,p,q) sqrt(p.^2 + q.^2);
% R = @(x,y) 1;
% exact_sol = @(x,y) 1 - sqrt(x.^2 + y.^2);
% Upper_Bound = 1;
% Lower_Bound = 0;
% u_init = @(x) 0;
% sigma_x = 1.000000000001;
% sigma_y = 1.000000000001;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2st example
L = 1;
H = @(x,y,p,q) sqrt(p.^2 + q.^2);
R = @(x,y) sqrt((1-abs(x)).^2 + (1-abs(y)).^2);
exact_sol = @(x,y) (1-abs(x)).*(1-abs(y));
Upper_Bound = 1;
Lower_Bound = 0;
u_init = @(x) 0;
sigma_x = 1.000000000001;
sigma_y = 1.000000000001;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Number of grid points
nx_vec = [16 32 64 128 256 512];% 1024];
for mesh = 1:length(nx_vec)
    nx = nx_vec(mesh);
    dx = 2*L/(nx-1);
    %[time iteration XX YY sol error u_exact] = monotone2D(H,R,L,nx,
       Lower_Bound,Upper_Bound,sigma_x,sigma_y,exact_sol);
    [time iteration XX YY sol error u_exact] = sweeping2D(H,R,L,nx,
       Lower_Bound,Upper_Bound,sigma_x,sigma_y,exact_sol);
    new_nx_vec(mesh) = nx;
    norm_error_vec(mesh) = error;
    time_need_vec(mesh) = time;
```

```
    ddx_vec(mesh) = dx;
    iteration_vec(mesh) = iteration;
    figure(mesh);
    surf(XX,YY,sol); title('Approximated solution');
    figure(mesh+1)
    surf(XX,YY,u_exact); title('Exact solution');
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %newfigure = figure(mesh+10); % compute the order of accuracy
    %set(newfigure,'color','white');
    %loglog(ddx_vec,abs(norm_error_vec));
    %title('Graph of norm error against dx on log-log scale')
    first_col = ones(length(new_nx_vec),1); second_col = log(ddx_vec)';
    AA = [first_col,second_col];
    FF_accuracy = log(norm_error_vec);
    solution_accuracy = (AA'*AA)^(-1)*AA'*FF_accuracy';
    %K = solution_accuracy(1); p = solution_accuracy(2); vpa([K,p])
    accuracy_vec(mesh) = solution_accuracy(2);
end
accuracy_vec
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
newfigure2 = figure;
plot(nx_vec,iteration_vec,'--o');
set(newfigure2,'color','white');
title('Graph of iteration against dx on log-log scale');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
newfigure3 = figure;
plot(nx_vec,time_need_vec,'-o');
set(newfigure3,'color','white');
title('Graph of time against dx on log-log scale');
```

References

[1] Michael G. Crandall, Hitoshi Ishii and Pierre-Louis Lions, User's guide to viscosity solutions of second order partial differential equation, Bull. Amer. Math. Soc. 27 (1992), pp 1-67.

[2] M.G Crandall, P.L Lions, Two approximations of solutions of Hamilton-Jacobi equations, Math. Comp., 43 (1984), pp. 1-19.

[3] Adam M.Oberman, Tiago Salvador, Filtered schemes for Hamilton - Jacobi equations: A simple construction of convergent accurate difference schemes, Journal of Computational Physics, 284 (2015), pp 367-388.

[4] Chiu YenKao, Stanley Osher, Jianliang Qian, Lax–Friedrichs sweeping scheme for static Hamilton–Jacobi equations, Journal of Computational Physics, 196 (2004), pp 367-391.