Numerical Methods for Monge-Ampere Equation

APMA 4301 - Final Project

Ivan Mitevski and Zirui Xu

5/15/2019

Abstract

The Monge-Ampere (MA) equation is a fully nonlinear degenerate elliptic partial differential equation that arises in optimal mass transportation, beam shaping, image registration, seismology, etc. In the classical form this equation is given by $\det(D^2\phi(x)) = f(x)$ where ϕ is constrained to be convex. Previous work has produced solvers that are fast but can fail on realistic (non-smooth) data or robust but relatively slow. The purpose of this work is to implement a more robust and time-efficient scheme for solving the MA equation and do convergence studies for different discretization and full multigrid schemes. We express the MA operator as the product of the eigenvalues of the Hessian matrix. This allows for a globally elliptic discretization that is provably convergent. The method combines a nonlinear Gauss-Seidel iterative method with different discretizations which is stable because the underlying scheme preserves monotonicity. In order to solve these systems efficiently, the V-cycle full approximation scheme multigrid method is exploited with error correction within the recursive algorithm; this scheme is used to leverage the low cost of computation on the coarse grids to build up the finer grids. This work shows computational results that demonstrate the speed and robustness of the algorithm.

1 Introduction

The Monge-Ampere (MA) equation is a fully nonlinear degenerate elliptic partial differential equation (PDE) with applications in classical problems of prescribed Gauss curvature, optimal mass transport, and other applications [2]. The right hand side of the equation can be strictly positive (and bounded) and in this case the operator is uniformly elliptic and the solutions turn out to be regular as long as the domain is strictly convex [2]. In the case when the right-hand side touches zero, the operator is degenerate elliptic and it is possible the solutions to be elliptic [2]. It is more difficult to solve the degenerate elliptic case.

1.1 Equation Setting

The equation is in the form

$$\det(D^2u(x)) = f(x), \qquad \text{for } x \text{ in } \Omega$$

where the operator is the determinant of the Hessian. In order for the equation to be elliptic, it has to have u be convex. The convexity constraint is necessary for the uniqueness of solutions and it is essential for numerical stability. In our case, we consider boundary conditions of the true solution.

1.2 Applications

The MA PDE is a geometric equation dating back to Monge and Ampere. The equation naturally appears is many geometric problems such as the Minkowski problem for prescribing the

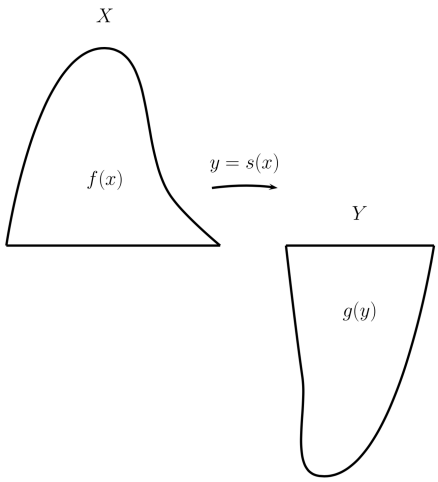

Figure 1: The mass transport problem [by Brittany Hamfeldt]

Gaussian curvature of a surface [6] and early applications such as dynamic meteorology, elasticity, and geometric optics [10]. The MA equation is also related to certain formulations of the Monge-Kantorovich optimal mass transportation problem. Other areas where the MA equations have been used are in image registration, mesh generations (Fig. 2), and astrophysics [6].
The MA equation appears as the optimal condition in the case of the optimal mass transport with a quadratic cost. This problem looks for a map $\mathbf{g}(x)$ which is used for moving the measure $\mu_1(x)$ to $\mu_2(x)$ and this minimizes the transportation cost functional [6]:

$$\int_{\mathbb{R}}^{d} |x - \mathbf{g}(x)|^2 d\mu_1.$$

Mass transport problem is shown in Fig. 1. The optimal mapping for this transport is given by $\mathbf{g} = \nabla u$ where u is convex and satisfies the MA equation

$$\det(D^2 u(x)) = \mu_1(x)/\mu_2(\nabla u(x))$$

and in this case, the Dirichlet boundary conditions are replaced by the second boundary value problem

$$\mathbf{g}(\cdot) : \Omega_1 \to \Omega_2$$

where we have Ω_1 to be the support measure of μ_1 and Ω_2 to μ_2 [6]. There are difficulties with the boundary conditions in this case as they are implicit and it is not easy to implement them in our numerical schemes. In many applications the domains are squares and there is a simplifying assumption of Neumann boundary conditions which is allowed via having the edges mapped to edges. In some other cases, we see periodic boundary conditions as well. Dr. Hamfeldt has developed some method to implement the boundary condition $\mathbf{g}(\cdot) : \Omega_1 \to \Omega_2$ in [9].
In some other problems, we have the MA operator appear in an inequality constraint in a variational problem for optimal mapping where the cost is some sort of restricting the local area change on the set of admissible mappings [8].

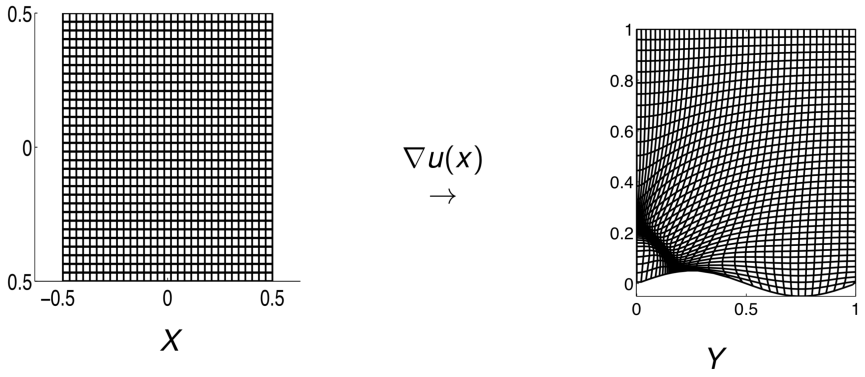

Figure 2: Mesh Generation [by Brittany Hamfeldt]

1.3 Numerical Difficulties

Most of the work done on solving MA equation numerically was done recently. The equation is very difficult to solve numerically due to the following reasons as presented in [2]:

- Non-linearity: The equation is fully nonlinear and therefore the weak solutions are either geometric solutions or viscosity solutions [2]. Even if the two-dimensional equation can be written in the divergence form we would still have the Hessian of the solution. This places restrictions on using the Finite Element Method [2].
- Singularity: The weak solutions can be very singular especially if the source f is not strictly positive then solutions may not be in $\boldsymbol{H}^2(\Omega)$.
- The convexity constraint: We need to have convex solution u in order for the MA equation to be elliptic. In case we do not have this convexity constraint, the equation will not have a unique solution [2].
- Fast solvers: It is challenging to do fast solvers with greater accuracy. The works of Brittany D. Hamfeldt and Adam M. Oberman show fast solvers for the MA equation [6]. They have developed some solvers with explicit methods independent of the solution time and for regular solutions a faster (1 order of magnitude) semi-implicit solution method which is slower (1 order of magnitude) when it comes to singular solutions [6].

Here is the divergence form in two dimensions:

$$\det(D^2u) = \frac{1}{2}\text{div}\left(\begin{pmatrix} u_{yy} & -u_{xy} \\ -u_{xy} & u_{xx} \end{pmatrix}(u_x, u_y)^T\right).$$

2 The Methods

In this section we are explaining the problem setup, some of the existing methods and its advantages and disadvantages, and we lay out the advantages of the alternative form of the MA equation.

2.1 Problem Setup

We are trying to solve the MA equation on a 2D convex region Ω with Dirichlet boundary condition

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2}\dfrac{\partial^2 u}{\partial y^2} - \left(\dfrac{\partial^2 u}{\partial x \partial y}\right)^2 = f \geqslant 0, & \text{in } \Omega, \\ u = g, & \text{on } \partial\Omega, \\ \text{constraint} : u \text{ is convex.} \end{cases} \tag{1}$$

2.2 Regularity Results

According to [4, Theorem 1.1], there exists a unique strictly convex solution $u \in C^\infty(\bar{\Omega})$ to Equation (1), if the following assumptions are satisfied:

- smoothness assumption: $f, g \in C^\infty(\bar{\Omega})$;
- strict positiveness assumption: $f > 0$;
- shape requirement of the domain: Ω is strictly convex.

If the above smoothness condition $f,\ g \in C^\infty(\bar{\Omega})$ is weakened to Hölder continuity $f \in C^\alpha(\Omega)$ and $g \in C^{2,\alpha}(\partial\Omega)$, then there exists a unique solution $u \in C^{2,\alpha}$. [7]

2.3 Viscosity Solution

In some cases, f is not a smooth function, and there does not exist a classical solution to MA equation. Therefore it is necessary to consider the weak solution to our problem.

Let's define the viscosity solution of the above MA equation. The definition we are using here is adapted from [5, Definition 2.2].

We say u is a viscosity subsolution if u is convex and continuous, and for all $\phi \in C^2(\Omega)$ and $x_0 \in \Omega$, the following inequality holds

$$\frac{\partial^2 \phi(x_0)}{\partial x^2}\frac{\partial^2 \phi(x_0)}{\partial y^2} - \left(\frac{\partial^2 \phi(x_0)}{\partial x \partial y}\right)^2 \geqslant f(x_0),$$

if ϕ is convex and x_0 is a local maximum of $u - \phi$.

Similarly, we say u is a viscosity supersolution if u is convex and continuous, and for all $\phi \in C^2(\Omega)$ and $x_0 \in \Omega$, the following inequality holds

$$\frac{\partial^2 \phi(x_0)}{\partial x^2}\frac{\partial^2 \phi(x_0)}{\partial y^2} - \left(\frac{\partial^2 \phi(x_0)}{\partial x \partial y}\right)^2 \leqslant f(x_0),$$

if ϕ is convex and x_0 is a local minimum of $u - \phi$.

If u is both a viscosity subsolution and viscosity supersolution, then we say u is a viscosity solution.

If u is a classical solution, i.e. $u \in C^2(\Omega)$ and satisfies (1), then u is also a viscosity solution. To prove this, consider convex and continuously twice differentiable ϕ, if x_0 is a local maximum of $u - \phi$, then $\nabla u(x_0) = \nabla \phi(x_0)$ and $A \preceq B$, where $A = D^2 u(x_0)$, $B = D^2 \phi(x_0)$. To prove u

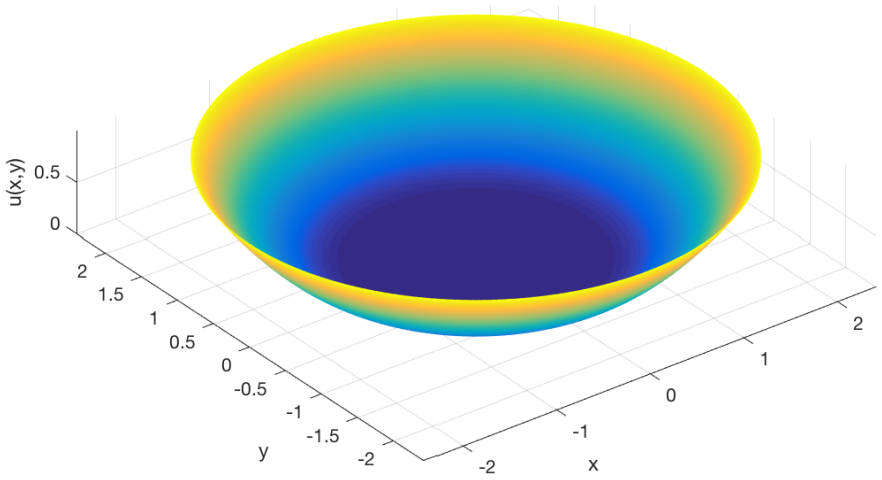

Figure 3: A viscosity solution in $\mathbb{R}^2$

is a viscosity subsolution, we only need to prove $\det(A) \leqslant \det(B)$ (and we only need to consider the case when A is non-degenerate). Denote $C = B - A$, then C is non-negative definite. Note that $B = A + C = A^{\frac{1}{2}}(I + A^{-\frac{1}{2}}CA^{-\frac{1}{2}})A^{\frac{1}{2}}$, where $A^{-\frac{1}{2}}CA^{-\frac{1}{2}} \succeq 0$, so $\det(I + A^{-\frac{1}{2}}CA^{-\frac{1}{2}}) \geqslant 1$, also $\det(A^{\frac{1}{2}}) = \det(A)^{\frac{1}{2}}$, therefore we have $\det(B) \geqslant \det(A)$ and thus u is indeed a viscosity subsolution. Similarly, u is also a viscosity supersolution. Thus we proved that classical solution is viscosity solution.

Here is an example of viscosity solution from [6, Example 1]:

$$u(x, y) = \frac{1}{2} \max\left\{\sqrt{x^2 + y^2} - 1,\ 0\right\}^2,$$

and the corresponding f is given by

$$f(x, y) = \max\left\{1 - \frac{1}{\sqrt{x^2 + y^2}},\ 0\right\}.$$

The solution u is similar to a pan, as is shown in Figure 3. It is not twice differentiable near the unit circle, but it satisfies the conditions for viscosity solutions.

2.4 Monotonicity of Numerical Schemes

Now let us take a break and discuss the importance of the monotonicity of numerical schemes. We say an update scheme is monotone if for all the discrete solutions U and V, $\forall\, n \geqslant 1$,

$$\forall\, j,\ U_j^n \geqslant V_j^n \quad \Longrightarrow \quad \forall\, j,\ U_j^{n+1} \geqslant V_j^{n+1}.$$

Example 1: forward Euler scheme for the heat equation $u_t = \kappa u_{xx}$ is

$$U_j^{n+1} = U_j^n + \frac{\kappa \Delta t}{\Delta x^2}(U_{j+1}^n - 2U_j^n + U_{j-1}^n),$$

this is monotone if and only if the CFL condition is satisfied: $2\kappa\Delta t \leqslant \Delta x^2$.

Example 2: upwind scheme for the advection equation $u_t + au_x = 0$ is

$$U_j^{n+1} = U_j^n - \frac{a\Delta t}{\Delta x}(U_j^n - U_{j-1}^n),$$

this is monotone if and only if the CFL condition is satisfied: $0 \leqslant a\Delta t \leqslant \Delta x$.

Example 3: Lax-Friedrichs scheme for the advection equation $u_t + au_x = 0$ is

$$U_j^{n+1} = \frac{1}{2}(U_{j-1}^n + U_{j+1}^n) - \frac{a\Delta t}{2\Delta x}(U_{j+1}^n - U_{j-1}^n),$$

this is monotone if and only if the CFL condition is satisfied: $|a|\Delta t \leqslant \Delta x$.

From the above examples we can see that monotonicity is important to the stability of numerical schemes (although they are not always concurrent, there are some counterexamples).

As a remark, the heat equation and the advection equation are monotone equations in the sense that for all the true solutions u and v, $\forall\ \Delta t > 0$,

$$\forall\, x,\ u(x,t) \geqslant v(x,t) \quad \Longrightarrow \quad \forall\, x,\ u(x,t+\Delta t) \geqslant v(x,t+\Delta t),$$

here we didn't consider the boundary condition.

Also, the MA operator is monotone in the sense that for all $u,\ v \in C^2(\Omega)$, if $u(x_0) = v(x_0)$ and $\forall\, x,\ u(x) \geqslant v(x)$, then $\det\left(D^2u(x_0)\right) \geqslant \det\left(D^2v(x_0)\right)$. To prove this, denote $A = D^2u(x_0)$ and $B = D^2v(x_0)$, then $A - B \succeq 0$ because x_0 is the minimum of $u - v$. If B is non-degenerate, we have $\det(A) = \det(B + A - B) = \det\left(B^{\frac{1}{2}}\left(I + B^{-\frac{1}{2}}(A-B)B^{-\frac{1}{2}}\right)B^{\frac{1}{2}}\right) = \det(B)\det\left(I + B^{-\frac{1}{2}}(A - B)B^{-\frac{1}{2}}\right) \geqslant \det(B)$. Based on this, it is necessary for us to examine the monotonicity of the discrete MA operator.

2.5 Disadvantages of the Simple Discretization

A simple discretization of the MA operator $u_{xx}u_{yy} - u_{xy}^2$ on 9-points stencil is

$$(\mathrm{MA}_h u)_{ij} = f_{ij},$$

where

$$(\mathrm{MA}_h u)_{ij} = \frac{u_{i+1\,j} + u_{i-1\,j} - 2u_{i\,j}}{h^2} \times \frac{u_{i\,j+1} + u_{i\,j-1} - 2u_{i\,j}}{h^2} - \left(\frac{u_{i+1\,j+1} + u_{i-1\,j-1} - u_{i+1\,j-1} - u_{i-1\,j+1}}{4h^2}\right)^2,$$

this discretization scheme MA_h is not monotone, due to the discretization for u_{xy}:

$$\frac{u_{i+1\,j+1} + u_{i-1\,j-1} - u_{i+1\,j-1} - u_{i-1\,j+1}}{4h^2},$$

that is, if we use coordinate descent method (which is similar to Gauss-Seidel method), the update scheme will be solving the following quadratic equation with respect to $U_{i\,j}^{n+1}$ (we choose the smaller root for the sake of the convexity),

$$\frac{U_{i+1\,j}^n + U_{i-1\,j}^n - 2U_{i\,j}^{n+1}}{h^2} \times \frac{U_{i\,j+1}^n + U_{i\,j-1}^n - 2U_{i\,j}^{n+1}}{h^2} - \left(\frac{U_{i+1\,j+1}^n + U_{i-1\,j-1}^n - U_{i+1\,j-1}^n - U_{i-1\,j+1}^n}{4h^2}\right)^2 = f_{ij},$$

denote the root-finding algorithm by T, i.e.,

$$U^{n+1}_{i\,j} = T(U^n_{i+1\,j+1},\, U^n_{i+1\,j},\, U^n_{i+1\,j-1},\, U^n_{i\,j+1},\, U^n_{i\,j-1},\, U^n_{i-1\,j+1},\, U^n_{i-1\,j},\, U^n_{i-1\,j-1},\, f_{i\,j}).$$

The bad news is that, if T increases as $U^n_{i+1\,j+1}$, $U^n_{i-1\,j-1}$ increases, then T will decrease as $U^n_{i-1\,j+1}$, $U^n_{i+1\,j-1}$ increases, and vice versa. So it is impossible for T to be monotonically increasing with respect to the four arguments $U^n_{i+1\,j+1}$, $U^n_{i-1\,j-1}$, $U^n_{i-1\,j+1}$, $U^n_{i+1\,j-1}$ at the same time. As a result, this discretization scheme will never be monotone.

Here is another way to understand that MA_h is not monotone. Assume

$$\begin{aligned}
u_{i\,j} &= v_{i\,j},\\
u_{i+1\,j} &\geqslant v_{i+1\,j},\\
u_{i-1\,j} &\geqslant v_{i-1\,j},\\
u_{i\,j+1} &\geqslant v_{i\,j+1},\\
u_{i\,j-1} &\geqslant v_{i\,j-1},\\
u_{i+1\,j+1} &\geqslant v_{i+1\,j+1},\\
u_{i-1\,j-1} &\geqslant v_{i-1\,j-1},\\
u_{i+1\,j-1} &\geqslant v_{i+1\,j-1},\\
u_{i-1\,j+1} &\geqslant v_{i-1\,j+1},
\end{aligned}$$

we still cannot guarantee that $(\mathrm{MA}_h u)_{ij} \geqslant (\mathrm{MA}_h v)_{ij}$. In this regard, the discrete operator MA_h hasn't inherited the monotonicity of the continuum MA operator $\det(D^2)$.

2.6 Alternative Form of MA Operator

Under a new orthonormal basis q_1, q_2, the hessian matrix of u is $Q^{\mathrm{T}}AQ$, where $Q = [q_1\ q_2]$. Since $Q^{\mathrm{T}}AQ$ is non-negative definite, by Hadamard's inequality, we know $\det(A) = \det(Q^{\mathrm{T}}AQ) \leqslant \prod \mathrm{diag}(Q^{\mathrm{T}}AQ)$, where $\prod \mathrm{diag}()$ means the product of the diagonal entries, and the equality is achieved if and only if $Q^{\mathrm{T}}AQ$ is a diagonal matrix. Therefore we can reformulate $\det\left(D^2u(x_0)\right) = f(x_0)$ to be

$$\min_{\substack{\text{Orthonormal}\\ \text{basis } \{q_1,\,q_2\}}} u_{q_1q_1}(x_0)u_{q_2q_2}(x_0) = f(x_0),$$

the left hand side is essentially the convexified MA operator $(\mathrm{MA})^+$ proposed in [6].

The constraint for u to be convex at x_0 is equivalent to requiring $D^2u(x_0)$ be non-negative definite, and thus is equivalent to requiring $u_{q_1q_1} \geqslant 0$, $u_{q_2q_2} \geqslant 0$ for every orthonormal basis $\{q_1,\, q_2\}$.

The advantage of this new reformulation is that we have monotone discretization scheme for $u_{q_1q_1}$ and $u_{q_2q_2}$.

2.7 Advantages of the New Discretization

For $(\mathrm{MA})^+$, we can adopt the following discretization on $(2L+1)\times(2L+1)$ $(L \in \mathbb{N})$ stencil:

$$\det\left(D^2u_{ij}\right) \approx \max\left\{0,\ (\mathrm{MA}^+_h u)_{ij}\right\},$$

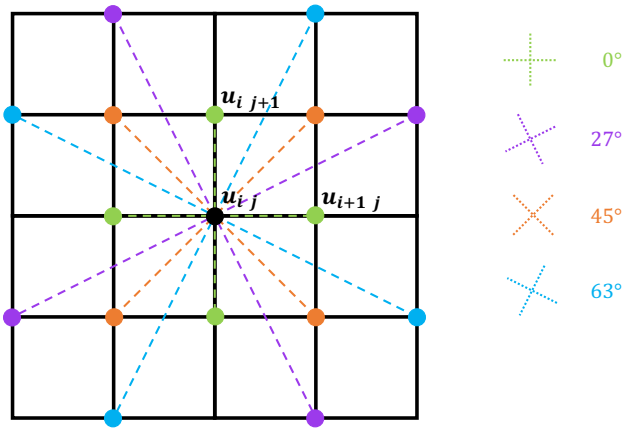

Figure 4: $\mathcal{D}_V^{\mathrm{WS}}$ on 5×5 stencil

where

$$(\mathrm{MA}_h^+ u)_{ij} = \min_{\substack{0<m\leqslant L \\ 0\leqslant l\leqslant L \\ \gcd(m,l)=1}} \frac{u_{i+m\ j+l} - 2u_{i\ j} + u_{i-m\ j-l}}{m^2h^2 + l^2h^2} \times \frac{u_{i+l\ j-m} - 2u_{i\ j} + u_{i-l\ j+m}}{l^2h^2 + m^2h^2}.$$

Since u is convex, we know $\det\left(D^2 u\right) \geqslant 0$, so we take the maximum value of 0 and the minimization result. In [1, Equation (1.6)], this discretization is called Wide Stencil scheme $\mathcal{D}_V^{\mathrm{WS}}$. For 5×5 stencil (i.e. $L = 2$), $\mathcal{D}_V^{\mathrm{WS}}$ is shown in Figure 4, where four different colors represent four orthogonal bases of different orientations, 0°, 27°, 45°, 63° respectively. If we instead use 3×3 stencil, then only the greenish and orange orthogonal bases are available.

The main advantage of the new scheme is that $u_{i+m\ j+l} - 2u_{i\ j} + u_{i-m\ j-l}$ is just the standard second-order central difference, which has better monotonicity than $u_{i+1\ j+1} + u_{i-1\ j-1} - u_{i+1\ j-1} - u_{i-1\ j+1}$. To make it more clear, after finding the minimizer pair m^*, l^*, we then solve the following quadratic equation with respect to $U_{i\ j}^{n+1}$ (again, we choose the smaller root for the sake of the convexity)

$$\frac{U_{i+m^*\ j+l^*}^n - 2U_{i\ j}^{n+1} + U_{i-m^*\ j-l^*}^n}{(m^*h)^2 + (l^*h)^2} \times \frac{U_{i+l^*\ j-m^*}^n - 2U_{i\ j}^{n+1} + U_{i-l^*\ j+m^*}^n}{(l^*h)^2 + (m^*h)^2} = f_{ij},$$

denote the root-finding algorithm by T, i.e.,

$$U_{i\ j}^{n+1} = T(U_{i+m^*\ j+l^*}^n,\ U_{i-m^*\ j-l^*}^n,\ U_{i+l^*\ j-m^*}^n,\ U_{i-l^*\ j+m^*}^n,\ f_{i\ j},\ m^*,\ l^*).$$

it can be shown that T increases as $U_{i+m^*\ j+l^*}^n$, $U_{i-m^*\ j-l^*}^n$, $U_{i+l^*\ j-m^*}^n$, $U_{i-l^*\ j+m^*}^n$ increase, so this new scheme is monotone. And it has inherited the monotonicity of the continuum MA operator

$\det(D^2)$, that is, under the assumption

$$\begin{aligned} u_{i\,j} &= v_{i\,j}, \\ u_{i+1\,j} &\geqslant v_{i+1\,j}, \\ u_{i-1\,j} &\geqslant v_{i-1\,j}, \\ u_{i\,j+1} &\geqslant v_{i\,j+1}, \\ u_{i\,j-1} &\geqslant v_{i\,j-1}, \\ u_{i+1\,j+1} &\geqslant v_{i+1\,j+1}, \\ u_{i-1\,j-1} &\geqslant v_{i-1\,j-1}, \\ u_{i+1\,j-1} &\geqslant v_{i+1\,j-1}, \\ u_{i-1\,j+1} &\geqslant v_{i-1\,j+1}, \end{aligned}$$

we can guarantee that $(\mathrm{MA}_h^+ u)_{ij} \geqslant (\mathrm{MA}_h^+ v)_{ij}$.

As one of the drawbacks of this new discretization, in order to improve accuracy, we have to use very wide stencil because we have to consider angular resolution, from which the error arises can dominate in the case when the solutions are singular. However in practice, narrow stencil is already competent for most regular solutions.

Another similar reformulation of the MA operator is proposed in [1, Definition 1.4].

$$\min_{\substack{\text{Unit vectors} \\ \{q_1, q_2, q_3\}}} h\big(u_{q_1 q_1}(x_0),\, u_{q_2 q_2}(x_0),\, u_{q_3 q_3}(x_0)\big) = f(x_0),$$

where

$$h(a,b,c) = \begin{cases} bc, & \text{if } a \geqslant b + c, \\ ac, & \text{if } b \geqslant a + c, \\ ab, & \text{if } c \geqslant a + b, \\ \dfrac{bc + ac + ab}{2} - \dfrac{a^2 + b^2 + c^2}{4}, & \text{otherwise.} \end{cases}$$

The discretization of this new formulation is called $\mathcal{D}_V^{\mathrm{LBR}}$. According to [1, Theorem 1.9], $\mathcal{D}_V^{\mathrm{LBR}}$ is consistent under the M-obtuse assumption.

3 Multigrid

After discretization, the MA equation is reduced to a system of nonlinear equations. We use full approximation scheme (FAS, see [3, Chapter 6]) to solve those nonlinear equations efficiently on multigrid.

Generally speaking, the FAS of depth two can be described as follows:

- aim: to solve discrete nonlinear equations on a fine grid with grid size h: $A^h(u^h) = f^h$;
- given a current approximation solution on the fine grid: v^h;
- compute the residual on the fine grid: $r^h = f^h - A^h(v^h)$;
- restrict r^h to the coarse grid: $r^{2h} = I_h^{2h} r^h$;

- restrict the approximation solution v^h to the coarse grid: $v^{2h} = I_h^{2h} v^h$;
- solve the restricted problem on the coarse grid: $A^{2h}(u^{2h}) = A^{2h}(v^{2h}) + r^{2h}$;
- compute the error on the coarse grid: $e^{2h} = u^{2h} - v^{2h}$;
- interpolate e^{2h} to the fine grid: $e^h = I_{2h}^h e^{2h}$;
- use e^h to correct the current approximation on the fine grid: $v^h \leftarrow v^h + e^h$

where I_h^{2h} denotes the restriction operator, and I_{2h}^h denotes the interpolation operator.

Note that if v^h is the exact solution on the fine grid, i.e., $r^h = 0$, then $r^{2h} = 0$ and thus $A^{2h}(u^{2h}) = A^{2h}(v^{2h})$, by the non-degeneracy of A^{2h} we know $u^{2h} = v^{2h}$, so $e^{2h} = 0$ and $e^h = 0$, therefore v^h is a fixed point of our FAS iteration, as we expect.

We should be aware that the nonlinear equations we try to solve on the coarse grid is equivalent to

$$A^{2h}(u^{2h}) = I_h^{2h} f^h + A^{2h}(I_h^{2h} v^h) - I_h^{2h} A^h(v^h), \tag{2}$$

instead of

$$A^{2h}(u^{2h}) = I_h^{2h} f^h,$$

because A and I_h^{2h} do not commute in general.

In other words, the equations on the coarse grid depend on the current solution on the fine grid. According to [3], it is the tau correction term $A^{2h}(I_h^{2h} v^h) - I_h^{2h} A^h(v^h)$ in Equation (2) that alters the equations on the coarse grid and reduces the resolution error arising from the relatively large grid size $2h$.

We have only talked about two layer version of FAS. In fact, we can apply this FAS method recursively to solving the restricted problem on the coarser grid, and therefore reduce the size of problem, i.e. A^h, A^{2h}, A^{4h}, A^{8h}, $\cdots$. In practice, we can implement FAS to be a V-cycle or W-cycle scheme.

4 Results

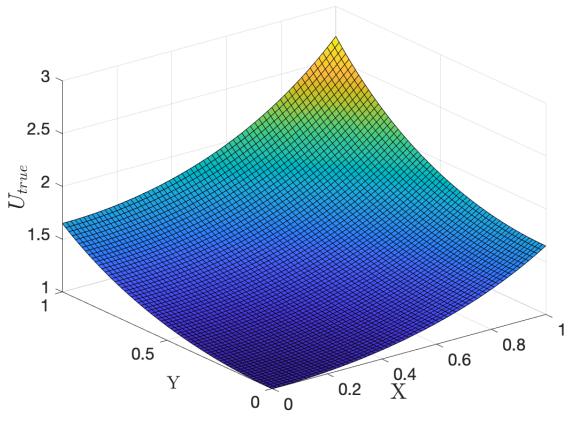

Figure 5: True Solution with $N = 65$

We are plotting the true solution in Fig. 5 and the error at different iterations in Fig. 6 for $N = 65$. The solver works with two basis of 0^0 and 45^0 as described in Sec. 2.7. The number of iterations represents one V-cycle in the multigrid which can be from 1-4 levels in depth. The error goes down and we control the error by setting the residual tolerance in the `res` variable which is set to $h^2/10$ where h is the grid-size. We have the error to be zero at the boundary and that is a good check with our problem setup. It is interesting to see in Fig. 6 is how the error oscillates in different iterations between positive and negative at the beginning iterations. Iteration two is where the error gets positive and then after that iteration, it stays negative. We used few iterations of Gauss-Seidel to damp the high-frequency components of the error and since the error is smoother then we went to represent it on a coarser resolution grid. Then we did Gauss-Seidel again and continued the multigrid cycle where the error was less oscillatory.

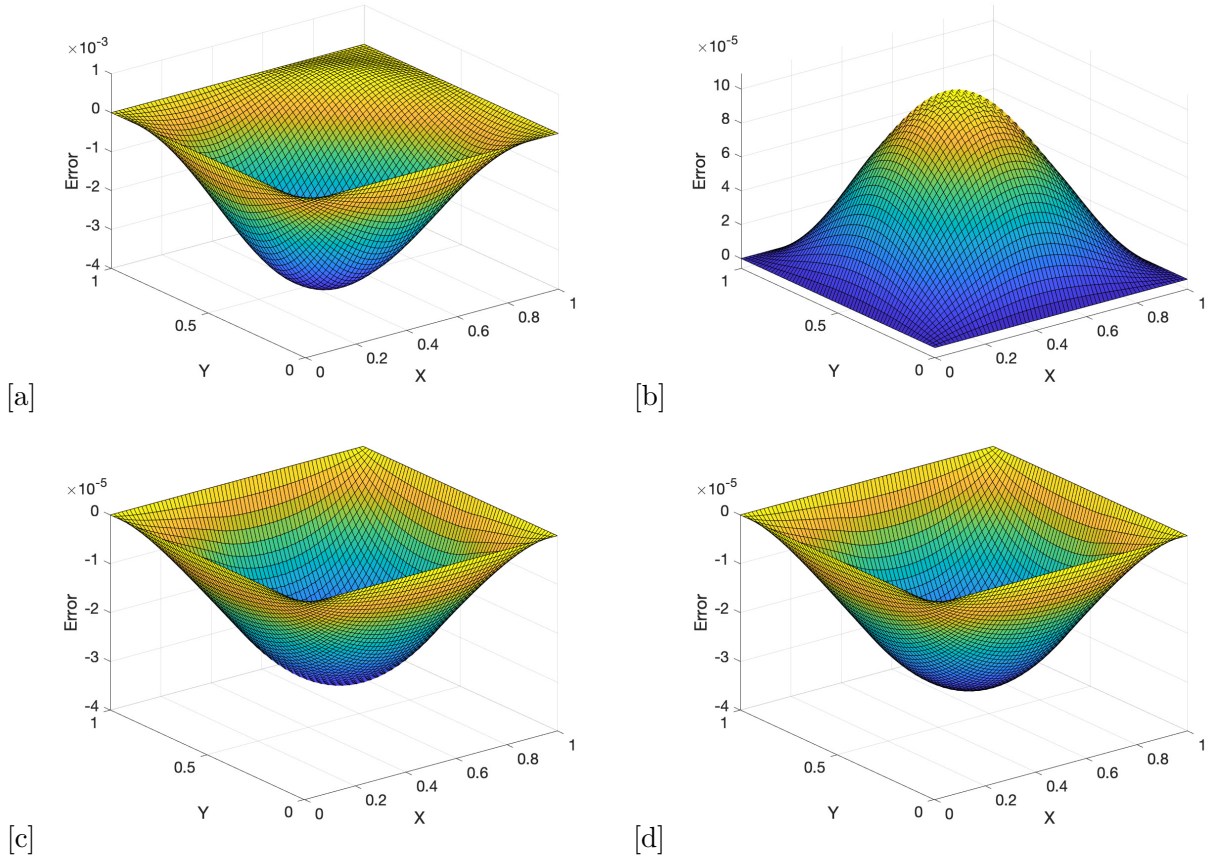

Figure 6: These figures show the error behavior on the domain when using mesh of $N = 65$ and 2 level multigrid at iteration number a) one b) two c) three and d) four.

Fig. 7.b shows how the error behaves which is pretty linear as we have set it to be and therefore we do not do more in-depth studies on this matter. However, we use the "prescribed" error to study the number of iterations it took us to get there which is important to see the efficiency of using multiple levels in multigrid. As we can see in Fig. 7.a, the more levels we use as we go on a finer grid, the fewer iterations it takes us to get to the prescribed error. It appears that this method is pretty efficient when scaling to finer grids and can be further developed for better solvers for Monge-Ampere.

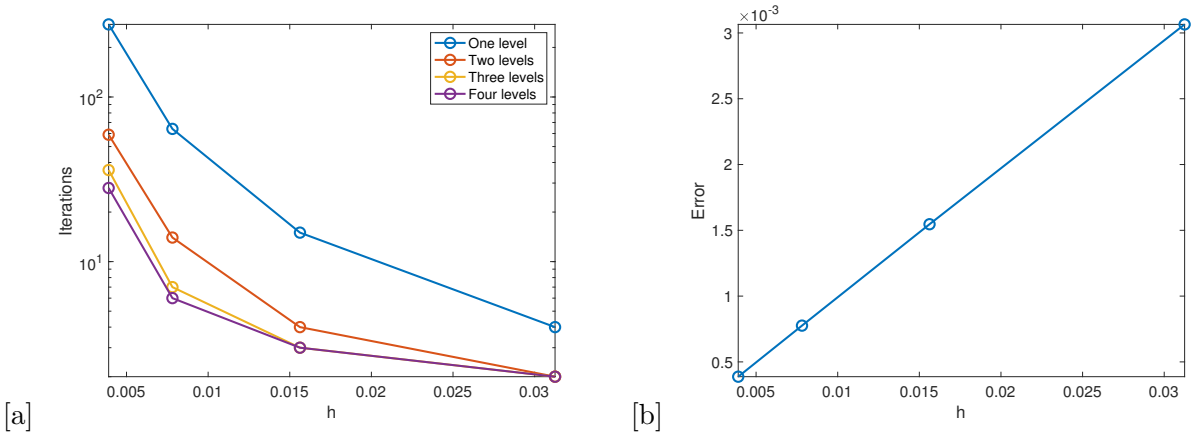

Figure 7: a) Iteration counts for different mesh sizes; b) Error behavior at different mesh sizes

For illustrative purposes, we are showing the residual at different iterations in Fig. 8 and we can see the decreasing and smoothing trend as we are more iterations in the process. These trends are similar with the ones we say in Fig. 6 for the error.

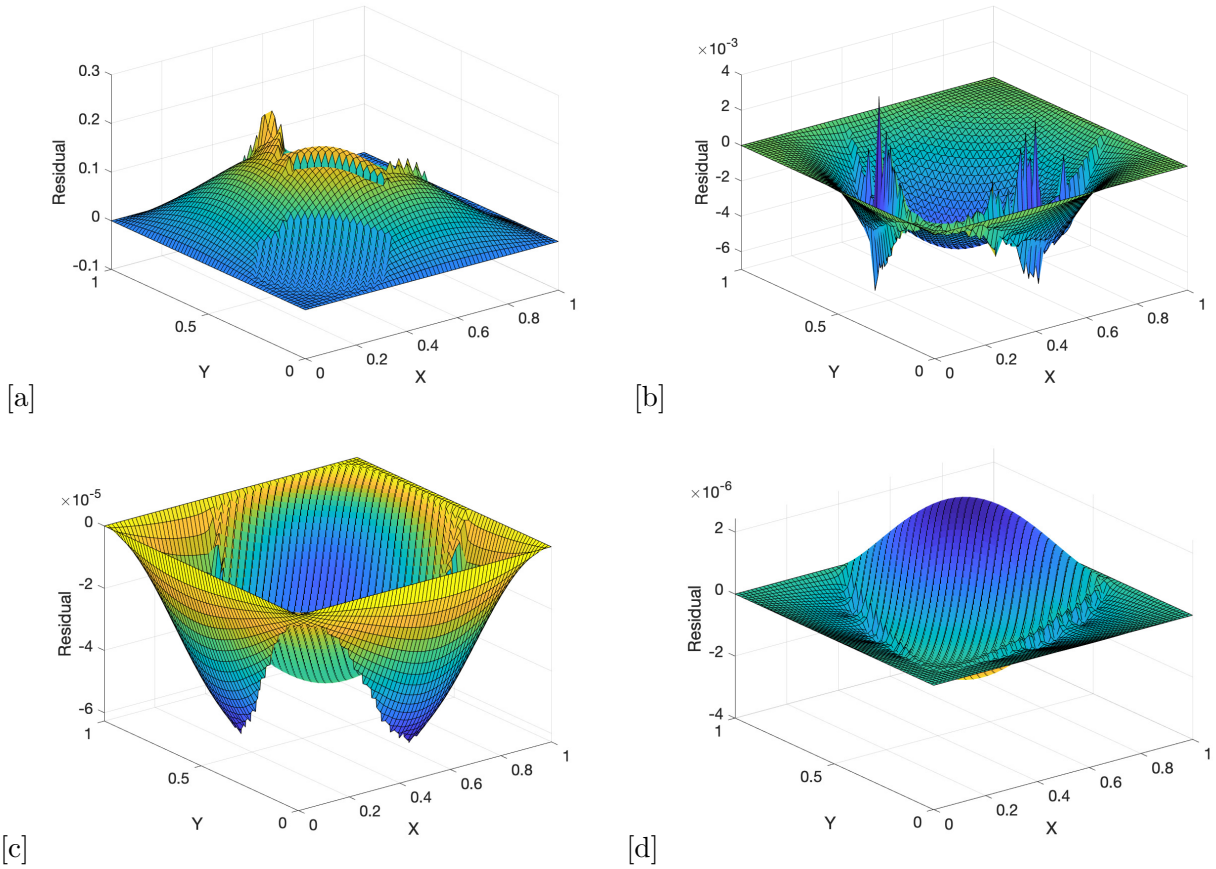

Figure 8: These figures show the residual behavior on the domain when using mesh of $N = 65$ and 2 level multigrid at iteration number a) one b) two c) three and d) four.

5 Summary

We analyzed the Monge-Ampere equation which is fully nonlinear elliptic PDE and we mentioned its various applications in different areas. There are substantial numerical difficulties when it comes to the non-linearity, singularity, and convexity constraints which are a major burden to making solvers which are robust but also fast. It was only recently when substantial work was done on this equation and we go over in shallow details of these methods and then describe our multigrid approach of the alternate form of the equation operator that we work with. We use the monotonicity properties to approximate the operator and then use multigrid with Gauss-Seidel operations on it. The results show that the solver is pretty robust and the more levels we use in multigrid the fewer iterations it takes us to get to the prescribed error.

References

[1] Jean-David Benamou, Francis Collino, and Jean-Marie Mirebeau. "Monotone and consistent discretization of the Monge-Ampere operator". In: *Mathematics of computation* 85.302 (2016), pp. 2743–2775.

[2] Jean-David Benamou, Brittany D Froese, and Adam M Oberman. "Two numerical methods for the elliptic Monge-Ampere equation". In: *ESAIM: Mathematical Modelling and Numerical Analysis* 44.4 (2010), pp. 737–758.

[3] William L Briggs, Steve F McCormick, et al. *A multigrid tutorial*. Vol. 72. Siam, 2000.

[4] Luis Caffarelli, Louis Nirenberg, and Joel Spruck. "The dirichlet problem for nonlinear second-order elliptic equations I. Monge-ampégre equation". In: *Communications on pure and applied mathematics* 37.3 (1984), pp. 369–402.

[5] Michael G Crandall, Hitoshi Ishii, and Pierre-Louis Lions. "User's guide to viscosity solutions of second order partial differential equations". In: *Bulletin of the American mathematical society* 27.1 (1992), pp. 1–67.

[6] Brittany D Froese and Adam M Oberman. "Convergent finite difference solvers for viscosity solutions of the elliptic Monge–Ampère equation in dimensions two and higher". In: *SIAM Journal on Numerical Analysis* 49.4 (2011), pp. 1692–1714.

[7] Cristian E Gutiérrez and Haim Brezis. *The Monge-Ampere equation*. Vol. 44. Springer, 2001.

[8] Eldad Haber, Raya Horesh, and Jan Modersitzki. "Numerical optimization for constrained image registration". In: *Numerical Linear Algebra with Applications* 17.2fffdfffdfffd3 (2010), pp. 343–359. DOI: 10.1002/nla.715. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nla.715. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.715.

[9] Brittany Hamfeldt. "A Numerical Method for the Elliptic Monge–Ampfffdfffdre Equation with Transport Boundary Conditions". In: *CoRR* abs/1101.4981 (Jan. 2011). DOI: 10.1137/110822372.

[10] Vladimir I Oliker and Laird D Prussner. "On the numerical solution of the equation and its discretizations, i". In: *Numerische Mathematik* 54.3 (1989), pp. 271–293.

6 Appendix: MATLAB Code

The code and figures can be found on the GitHub repository `https://github.com/imitevski/ma_4301` and it is also attached in this appendix. The original code for this work was developed by Ivan Mitevski's undergraduate project (joint work with Matthew Illingworth and David Yousuf) on Monge-Ampere equation with Brittny Froese Hamfeldt and the same was edited for this work.

```
close all
clear all

% The solver is executed by running this file

% f1 and f2 are different depending on which basis we use
f1 = @(x,y) (1 + x.^2).*(1 + y.^2).*exp(x.^2 + y.^2);
f2 = @(x,y) exp(x.^2 + y.^2).*(1+.5*(x+y).^2).*(1+.5*(y-x).^2);
% f = @(x,y) (1 + x.^2 + y.^2).*exp(x.^2 + y.^2);
g = @(x,y) exp((x.^2 + y.^2)/2);

%Set the minimum and maximum values of n for the solution matrix.
minN = 4;
maxN = 5;
nValNum = length(minN:maxN);

stats = zeros(nValNum,maxN,2);
nVec = 2.^(minN+1:maxN+1)+1;
hVec = 2.^(-(minN+1:maxN+1));

% errCell is for storing the error matrices at specific "frames"
% (iterations). frameCell is for remembering which "frames" those were.

errCell = cell(nValNum,maxN);
resCell = cell(nValNum,maxN);
frameCell = cell(nValNum,maxN);

% exactSolCell stores g (the exact solution) evaluated on each mesh.

exactSolCell = cell(nValNum,1);
basesNum = 2;
iterVec = [5 50];

for i = minN:maxN
    i
    n = 2^(i+1) + 1;
    h = 1/(n-1);
    xa = 0; xb = 1; ya = 0; yb = 1; tol = h^2/10;
    [X,Y] = meshgrid(xa:h:xb,ya:h:yb);

    if basesNum == 1
        F = f1(X,Y);
    else
        F1 = f1(X,Y);
        F2 = f2(X,Y);
        F = min(F1,F2);
    end

    G = g(X,Y);
```

```
50      exactSolCell{i-minN+1} = G;
51      A = zeros(n);
52      u0 = init(F,g,n,h,X,Y);
53
54      u0(:,1) = g(X(:,1),Y(:,1));
55      u0(:,n) = g(X(:,n),Y(:,n));
56      u0(1,:) = g(X(1,:),Y(1,:));
57      u0(n,:) = g(X(n,:),Y(n,:));
58
59      N = n;
60
61      for j = 1:4
62
63          % The last argument is 0 because we don't (read: can't) use the mex
64          % file, and that argument is what turns it on or off.
65          [u,resMat,errMat,time,count] = ...
                looper2(F,g,n,N,j,2*iterVec,h,u0,xa,xb,ya,yb,tol,0);
66          stats(i-minN+1,j,1) = norm(errMat(:,:,end),inf);
67          stats(i-minN+1,j,2) = count;
68
69          % Choose up to 4 almost evenly spaced iteration numbers and store them.
70          errFrames = unique(ceil(linspace(1,count,4)));
71          frameCell{i-minN+1,j} = errFrames;
72
73          % Store the error matrices at the chosen iterations.
74          errCell{i-minN+1,j} = errMat(:,:,errFrames);
75          resCell{i-minN+1,j} = resMat(:,:,errFrames);
76
77      end
78
79  end
80
81  % Save everything, just for safety.
82
83  save('stats.mat','stats');
84  save('errCell.mat','errCell');
85  save('exactSolCell.mat','exactSolCell');
86
87  %% Error and iteration number plots
88
89  legendStrs = {'One level','Two levels','Three levels','Four levels',...
90      'Five levels','Six levels','Seven levels','Eight levels',...
91      'Nine levels','Ten levels'};
92
93  % Error figure
94  errFig = figure;
95  plot(hVec,stats(:,1,1), 'o-')
96  xlabel('h')
97  ylabel('Error')
98  title('Error vs. h for all depths of recursion')
99  axis tight
100 % set(gca, 'XDir','reverse') %reverses the order of h
101 saveas(errFig,'errFig.fig')
102
103 % Iteration figure
104 countFig = figure;
105 semilogy(hVec,stats(:,:,2),'o-');
106 legend(legendStrs(1:4));
107 xlabel('h')
```

```
ylabel('Iterations')
title(sprintf('Number of iterations vs. h for %d depths of recursion',4))
axis tight
% set(gca, 'XDir','reverse')
saveas(countFig,'countFig.fig')

%% Error surface plots

errorDir = 'error_surfs';
mkdir(errorDir);
resDir = 'res_surfs';
mkdir(resDir);
exactSolDir = 'exact_sol_surfs';
mkdir(exactSolDir);

for i = 1:nValNum

    nValDir1 = sprintf('%s/N_%d',errorDir,1/hVec(i)+1);
    mkdir(nValDir1)

    nValDir2 = sprintf('%s/N_%d',resDir,1/hVec(i)+1);
    mkdir(nValDir2)

    for j = 1:maxN
        if ¬isempty(errCell{i,j})
            depthDir1 = sprintf('%s/depth_%d',nValDir1,j);
            mkdir(depthDir1)

            depthDir2 = sprintf('%s/depth_%d',nValDir2,j);
            mkdir(depthDir2)

            subplotNum = size(errCell{i,j},3);
            err = errCell{i,j};
            res = resCell{i,j};

            for k = 1:subplotNum

                fig1 = figure;
                surf(linspace(0,1,nVec(i)),linspace(0,1,nVec(i)),abs(err(:,:,k)),...
                    'linestyle','none');
                title(sprintf('h = %f, depth = %d levels, iteration = %d',...
                    hVec(i),j,frameCell{i,j}(k)));
%                 zlim([0 norm(err(:),inf)]);
                saveas(fig1,sprintf('%s/count_%d.fig',depthDir1,frameCell{i,j}(k)));

                fig2 = figure;
                surf(linspace(0,1,nVec(i)),linspace(0,1,nVec(i)),res(:,:,k),...
                    'linestyle','none');
                title(sprintf('h = %f, depth = %d levels, iteration = %d',...
                    hVec(i),j,frameCell{i,j}(k)));
%                 zlim([0 norm(res(:),inf)]);
                saveas(fig2,sprintf('%s/count_%d.fig',depthDir2,frameCell{i,j}(k)));

            end
        end
    end

    fig = figure;
    surf(exactSolCell{i},'linestyle','none');
```

```
    title(sprintf('Exact solution evaluated for h = %f',hVec(i)));
    saveas(fig,sprintf('%s/N_%d.fig',exactSolDir,1/hVec(i)+1));
end

%% Error Surface Plots

% error at N = 65, depth 2
err = errCell{2,2}

fig1 = figure;
surf(linspace(0,1,65),linspace(0,1,65),err(:,:,1));
title('error evaluated at N=65, depth_2, count 1')
saveas(fig1,sprintf('error_65_d2_c1.fig'))

fig = figure;
surf(linspace(0,1,65),linspace(0,1,65),err(:,:,2));
title('error evaluated at N=65, depth_2, count 2')
saveas(fig,sprintf('error_65_d2_c2.fig'))

fig3 = figure;
surf(linspace(0,1,65),linspace(0,1,65),err(:,:,3));
title('error evaluated at N=65, depth_2, count 3')
saveas(fig3,sprintf('error_65_d2_c3.fig'))

fig4 = figure;
surf(linspace(0,1,65),linspace(0,1,65),err(:,:,4));
title('error evaluated at N=65, depth_2, count 4')
saveas(fig4,sprintf('error_65_d2_c4.fig'))
```

```
function [U] = A_solver(u,h,basesNum)
% A_SOLVER calculates the LHS of the equation using 2 basis and brings back
% the non-negative terms

%Calculate u_xx*u_yy, and leave only the nonnegative elements.
u_xy = ...
    max(1./h.^2.*(u(1:end-2,2:end-1)+u(3:end,2:end-1)-2.*u(2:end-1,2:end-1)),0)...
        .*max(1./h.^2.*(u(2:end-1,1:end-2)+u(2:end-1,3:end)-2.*u(2:end-1,2:end-1)),0);

if basesNum == 2

    %Calculate u_vv*u_v(perp)v(perp), and leave only the nonnegative elements.
    u_vw = ...
        max(1./(2.*h.^2).*(u(3:end,1:end-2)+u(1:end-2,3:end)-2.*u(2:end-1,2:end-1)),0)...
            .*max(1./(2.*h.^2).*(u(3:end,3:end)+u(1:end-2,1:end-2)-2.*u(2:end-1,2:end-1)),0);

    %Take the minimum of the two matrices.
    U = min(u_xy,u_vw);

else

    U = u_xy;

end

end
```

```
function [u] = init(F,g,n,h,X,Y)
%init.m calculates a reasonable initial guess to plug into the FAS
%function.

m = n-2;

F = sqrt(2*F(2:m+1,2:m+1));

G = g(X,Y);

F(:,1) = F(:,1) - G(2:m+1,1)/h^2;
F(:,m) = F(:,m) - G(2:m+1,m+2)/h^2;
F(1,:) = F(1,:) - G(1,2:m+1)/h^2;
F(m,:) = F(m,:) - G(m+2,2:m+1)/h^2;

F = reshape(F,m^2,1);

I = speye(m);
e = ones(m,1);
T = spdiags([e -4*e e],[-1 0 1],m,m);
S = spdiags([e e],[-1 1],m,m);
A = (kron(I,T) + kron(S,I))/h^2;

uVec = A\F;
G(2:m+1,2:m+1) = reshape(uVec,[m,m]);

u = G;

end
```

```
function [u,resRec,errMat,time,count] = ...
    looper2(F,g,n,N,levels,iterVec,h,u0,xa,xb,ya,yb,tol,mex)

%LOOPER2 is used to put stopping critera on the FAS_V2 iterations

tic

count = 0;

u = u0;
res = 1;

while res > tol

    count = count + 1;

    [u,resMat,err] = FAS_V2(F,g,n,N,levels,iterVec,h,u,xa,xb,ya,yb,count,mex);
    res = norm(resMat(:),inf);

    if count == 1
        errMat = err;
        resRec = resMat;
    else
        errMat = cat(3,errMat,err);
        resRec = cat(3,resRec,resMat);
    end
```

```
26
27 end
28
29 time = toc;
30
31 end
```

```
1  function [u,resMat,err] = FAS_V2(F,g,n,N,levels,iterVec,h,u0,xa,xb,ya,yb,count,mex)
2  %FAS_V2.m implements the Full Approximation Scheme.
3
4  direction = 'down';
5  plotFigs = 0;
6
7  %Do the initial Gauss-Seidel iteration.
8  [u,resMat] = GaussSeidel(F,g,iterVec,h,u0,xa,xb,ya,yb,0,mex);
9  res = norm(resMat(:),inf);
10
11 if plotFigs == 1
12     fig1 = figure(18);
13     surf(resMat,'linestyle','none')
14     title(sprintf('n = %d, res = %f, count = %d',n,res,count))
15     drawnow
16 end
17
18 %Go down one level of granularity, setting a new n and h.
19 n = (n+1)/2;
20 h = 2*h;
21
22 %Set the coarse versions of u, F, and the residual matrix.
23 v = restrict(u);
24 resCoarse = restrict(resMat);
25
26 %Evaluate the MA equation on the coarse grid, pad it with zeros so that
27 %it's the same size as resCoarse, and add them together.
28 A = padarray(A_solver(v,h,2),[1,1],0) + resCoarse;
29
30 n
31
32 %If we're on the lowest level, run the Gauss-Seidel calculation and set
33 %vNew as the output. Otherwise, set vNew as the recursive output of FAS_V.
34 if floor(log2(N)) - floor(log2(n)) ~= levels
35
36     vNew = FAS_V2(A,g,n,N,levels,iterVec,h,v,xa,xb,ya,yb,count,mex);
37
38 else
39
40     vNew = GaussSeidel(A,g,iterVec,h,v,xa,xb,ya,yb,1,mex);
41
42 end
43
44 direction = 'up';
45
46 %Calculate the coarse error matrix.
47 eCoarse = vNew - v;
48
49 %Interpolate the coarse error matrix to the fine level.
50 eFine = enhance(eCoarse);
```

```
51
52 %Add the error matrix to the original matrix u.
53 u(2:end-1,2:end-1) = u(2:end-1,2:end-1) + eFine(2:end-1,2:end-1);
54
55 %Come back up to the fine level.
56 n = 2*n-1;
57 h = h/2;
58
59 %Do a few more Gauss-Seidel iterations on u and calculate the residual
60 %matrix.
61 [u,resMat] = GaussSeidel(F,g,iterVec,h,u,xa,xb,ya,yb,0,mex);
62 res = norm(resMat(:),inf);
63
64 n
65
66 if plotFigs == 1
67     fig2 = figure(36);
68     surf(resMat,'linestyle','none')
69     title(sprintf('n = %d, res = %f, count = %d',n,res,count))
70     drawnow
71 end
72
73 %If we're on the finest level, calculate the error. Otherwise, set it to
74 %zero so as to avoid a 'not enough output arguments' error.
75 if n == N
76     [X,Y] = meshgrid(xa:h:xb,ya:h:yb);
77     G = g(X,Y);
78     err = G-u;
79 else
80     err = 0;
81 end
82
83 end
```

```
1  function [uFine] = enhance(u)
2  % ENHANCE does interpolation when moving from coarse to fine level
3
4  n = 2*length(u)-1;
5  uFine = zeros(n);
6
7  uFine(1:2:end,1:2:end) = u(:,:);
8
9  uFine(2:2:end-1,1:2:end) = .5*(u(1:end-1,:) + u(2:end,:));
10 uFine(1:2:end,2:2:end-1) = .5*(u(:,1:end-1) + u(:,2:end));
11 uFine(2:2:end-1,2:2:end-1) = .25*(u(1:end-1,1:end-1) + u(2:end,1:end-1)...
12                                 + u(1:end-1,2:end) + u(2:end,2:end));
13
14 end
```

```
1  function [uCoarse] = restrict(u)
2  % RESTRICT
3
4  U = u;
5
6  %Pad u with zeros to make the calculation simpler.
```

```
u = padarray(u,[1,1],0);

%Set uCoarse according to a linear combination of points from u.
uCoarse = .5*u(2:2:end-1,2:2:end-1) + .125*(u(1:2:end-2,2:2:end-1)...
                                          + u(3:2:end,2:2:end-1)...
                                          + u(2:2:end-1,1:2:end-2)...
                                          + u(2:2:end-1,3:2:end));

%Restrict u the old way by siply deleting every other point.
u = U(1:2:end,1:2:end);

%Reset the boundary values of uCoase with the correct boundary values.
uCoarse(:,1) = u(:,1);
uCoarse(:,end) = u(:,end);
uCoarse(1,:) = u(1,:);
uCoarse(end,:) = u(end,:);

end
```

```
function [u,resMat,err] = GaussSeidel(F,g,iterVec,h,u0,xa,xb,ya,yb,coarse,mex)
% GaussSeidel does Gauss Seidel iterations

%Lambda is a constant used for calculating a weighted combination of the
%existing u and the updated u.
lambda = 0.05;

%Set the grid on which the exact solution g(x,y) will be applied.
[X,Y] = meshgrid(xa:h:xb,ya:h:yb);
G = g(X,Y);

%Initialize the matrix u.
u = u0;

%Set the right-hand side of the MA equation and the number of iterations
%in the loop depending on whether the input matrix u is on the coarse or
%the fine level.
if coarse == 1
    maxCount = iterVec(1);
else
    maxCount = iterVec(2);
end

for cuenta = 1:maxCount

    %mex is always 0 for these runs
    if mex == 0
        uNew = notJacobi(u,F,h,2);
    else
        [m,n] = size(u);
        [uNew,hInv] = notNotGaussSeidel(u,F,h,m,n);
        uNew = uNew(2:end-1,2:end-1);
    end

%     figure(18)
%     u_Mat = notJacobi(u,F,h,2);
%
%     [m,n] = size(u);
```

```
%     u_C = notNotGaussSeidel(u,F,h,m,n);
%     u_C = u_C(2:end-1,2:end-1);

%     surf(abs(u_Mat - u_C))
%     surf(u_C)

%     uNew = notJacobi(u,F,h,2);

    %Update u with a weighted sum of points from the already-extant u and
    %the newly calculated uNew.
    u(2:end-1,2:end-1) = lambda*u(2:end-1,2:end-1) + (1-lambda)*uNew;

    resMat = padarray(F(2:end-1,2:end-1) - A_solver(u,h,2),[1,1],0);

%     figure(18)
%     surf(resMat)
%     drawnow

end

%Subtract u from the exact solution to find the error matrix.
err = G - u;

%Pad the residual matrix with zeros to make it the right size again.
resMat = padarray(F(2:end-1,2:end-1) - A_solver(u,h,2),[1,1],0);

end
```

```
function [u] = notJacobi(u,F,h,basesNum)
% notJacobi performs Gauss-Seidel iterations

indices = zeros(size(u));
indices(2:end-1,2:end-1) = 1;

indices = find(indices == 1);

for k = 1:length(indices)

    [i,j] = ind2sub(size(u),indices(k));

    if basesNum == 2

        A_xy = (1/h)^4*(u(i-1,j)+u(i+1,j)-2*u(i,j))...
                      *(u(i,j-1)+u(i,j+1)-2*u(i,j));

        A_vw = 1/(4*h^4)*(u(i-1,j-1)+u(i+1,j+1)-2*u(i,j))...
                        *(u(i+1,j-1)+u(i-1,j+1)-2*u(i,j));

        if A_xy ≤ A_vw

            u(i,j) = 0.25*(u(i+1,j)+u(i-1,j)+u(i,j-1)+u(i,j+1))...
                -0.5*sqrt(0.25*((u(i+1,j)+u(i-1,j)-u(i,j-1)-u(i,j+1))^2)...
                +h^4*F(i,j));

        else

            u(i,j) = 0.25*(u(i-1,j+1)+u(i+1,j-1)+u(i+1,j+1)+u(i-1,j-1))...
```

```
                -0.5*sqrt(0.25*((u(i-1,j+1)+u(i+1,j-1)-u(i+1,j+1)-u(i-1,j-1))^2)...
                +4*h^4*F(i,j));

        end

    else

            u(i,j) = 0.25*(u(i+1,j)+u(i-1,j)+u(i,j-1)+u(i,j+1))...
                -0.5*sqrt(0.25*((u(i+1,j)+u(i-1,j)-u(i,j-1)-u(i,j+1))^2)...
                +h^4*F(i,j));

    end

end

u = u(2:end-1,2:end-1);

end
```

```
/* MyMEXFunction
 * Adds second input to each
 * element of first input
 * a = MyMEXFunction(a,b);
 */

#include "mex.hpp"
#include "mexAdapter.hpp"
#include <cmath>

using namespace matlab::data;
using namespace std;
using matlab::mex::ArgumentList;
using matlab::engine::convertUTF8StringToUTF16String;

class MexFunction : public matlab::mex::Function {
public:
    void operator()(ArgumentList outputs, ArgumentList inputs) {
        //checkArguments(outputs, inputs);
        TypedArray<double> u = move(inputs[0]);
        TypedArray<double> F = move(inputs[1]);
        const double h = inputs[2][0];
        const int m = inputs[3][0];
        const int n = inputs[4][0];

        double A_xy, A_vw;

        //mexPrintf("The reciprocal of h is %f\n",3.5);
        //mexEvalString("drawnow;");

        for (int i = 1; i < m - 1; i++) {

            for (int j = 1; j < n - 1; j++) {

                A_xy = (1/h)*(1/h)*(1/h)*(1/h)
                      *(u[i-1][j]+u[i+1][j]-2*u[i][j])
                      *(u[i][j-1]+u[i][j+1]-2*u[i][j]);

```

```
                A_vw = 0.25*(1/h)*(1/h)*(1/h)*(1/h)
                      *(u[i-1][j-1]+u[i+1][j+1]-2*u[i][j])
                      *(u[i+1][j-1]+u[i-1][j+1]-2*u[i][j]);

                if (A_xy ≤ A_vw) {

                    u[i][j] = 0.25*(
                                    u[i+1][j]+u[i-1][j]+u[i][j-1]+u[i][j+1]
                                   )
                    -0.5*sqrt(
                              0.25*(
                                    (u[i+1][j]+u[i-1][j]-u[i][j-1]-u[i][j+1])
                                   *(u[i+1][j]+u[i-1][j]-u[i][j-1]-u[i][j+1])
                                   )
                    +h*h*h*h*F[i][j]);

                }   else   {

                    u[i][j] = 0.25*(
                                    u[i-1][j+1]+u[i+1][j-1]
                                   +u[i+1][j+1]+u[i-1][j-1]
                                   )
                    -0.5*sqrt(
                              0.25*(
                                    (u[i-1][j+1]+u[i+1][j-1]-u[i+1][j+1]-u[i-1][j-1])
                                   *(u[i-1][j+1]+u[i+1][j-1]-u[i+1][j+1]-u[i-1][j-1])
                                   )
                    +4*h*h*h*h*F[i][j]);

                }

            }

        }

        outputs[0] = u;
        outputs[1][0] = 1/h;

    /*
    void checkArguments(ArgumentList outputs, ArgumentList inputs) {
        // Get pointer to engine
        std::shared_ptr<matlab::engine::MATLABEngine> matlabPtr = getEngine();

        // Get array factory
        ArrayFactory factory;

        // Check first input argument
        if (inputs[0].getType() != ArrayType::DOUBLE ||
            inputs[0].getType() == ArrayType::COMPLEX_DOUBLE ||
            inputs[0].getNumberOfElements() != 1)
        {
            matlabPtr->feval(convertUTF8StringToUTF16String("error"),
                             0,
                             std::vector<Array>({ factory.createScalar("First ...
                                 input must scalar double") }));
        }

        // Check second input argument
        if (inputs[1].getType() != ArrayType::DOUBLE ||
```

```
97              inputs[1].getType() == ArrayType::COMPLEX_DOUBLE)
98          {
99              matlabPtr->feval(convertUTF8StringToUTF16String("error"),
100                               0,
101                               std::vector<Array>({ factory.createScalar("Input ...
                                     must double array") }));
102         }
103         // Check number of outputs
104         if (outputs.size() > 1) {
105             matlabPtr->feval(convertUTF8StringToUTF16String("error"),
106                              0,
107                              std::vector<Array>({ factory.createScalar("Only ...
                                     one output is returned") }));
108         }
109     }
110      */
111     }
112
113 };
```