A Numerical Method and Solver for the Elliptic Monge-Ampère Equation in Two Dimensions

UNCG
Department of Mathematics & Statistics

University of North Carolina at Greensboro

Author:
I.D.I.G. Gunawardana

Advisor:
Dr. Thomas Lewis

Spring 2018

Contents

Abstract

We propose a new finite difference method and solver for approximating the viscosity solution of the Monge-Ampère equation paired with Dirichlet boundary data. Simple discretizations are often plagued by the conditional uniqueness of the Monge-Ampère equation. As such, the corresponding algebraic system of nonlinear equations often has many false solutions. We propose a new scheme based upon adding a numerical moment. The numerical moment helps preserve the uniqueness of the viscosity solution through the discretization process and can handle the potential low-regularity of solutions to the Monge-Ampère equation. We also propose a modified Gauss-Seidel solver that helps select a sufficiently large stabilization parameter linked to the numerical moment. The method and solver are tested for various problems that show the reliability of the method for approximating the viscosity solution of the Monge-Ampère equation and the robustness of the solver.

1 Introduction

The Monge-Ampère (MA) equation is a fully nonlinear second order partial differential equation that appears in differential geometry, Riemann geometry, and optimal mass transport. We introduce the MA equation using the derivation from differential geometry. Let $u : \Omega \to \mathbb{R}$ be a continuous function, and define the subdifferential of u at x_0 by

$$\partial u(x_0) := \{p \mid u(x) \geq u(x_0) + p \cdot (x - x_0) \, \forall x \in \Omega\}.$$

Let $\partial u(E) := \cup_{x \in E} \partial u(x)$ for all $E \subset \Omega$. Then, the MA measure associated with u is defined by

$$\mathcal{M}_u(E) := \mathcal{L}^d \left(\partial u(E)\right) \quad \forall \text{ Borel sets } E \subset \Omega,$$

where $\mathcal{L}^d$ denotes the Lebesgue measure on $\mathbb{R}^d$. The MA problem then involves finding a continuous convex function u with given Dirichlet boundary data such that $\mathcal{M}_u = \mu$ for a given Radon measure μ. Suppose

$$\int_E f d\mathcal{L}^d = \mu(E)$$

for some function $f : \Omega \to \mathbb{R}$. It can be shown that if μ is absolutely continuous with respect to $\mathcal{L}^d$ and $u \in C^2(\overline{\Omega})$, then there holds

$$\det \left(D^2 u\right) = f \tag{1}$$

pointwise. To see this, use the fact $\partial u(x_0) = \nabla u(x_0)$ for $u \in C^1(\overline{\Omega})$ and Sard's Theorem [6] to obtain

$$\int_E f d\mathcal{L}^d = \mu(E) = \mathcal{M}_u(E) = \int_{\nabla u(E)} d\mathcal{L}^d = \int_E \det \left(D^2 u\right) d\mathcal{L}^d \tag{2}$$

for all Borel sets $E \subset \Omega$. Equation (1) is a strong form of the MA equation while (2) is a variational form of the MA equation. A more complete derivation can be found in [9] and [10].

A solution u that satisfies the variational form of the MA equation $\mathcal{M}_u = \mu$ along with the given Dirichlet data is called an Aleksandrov solution to the MA equation. In general, for a non-strictly convex domain Ω, the MA equation may not have a classical solution even if the source f, Dirichlet boundary data, and $\partial\Omega$ are smooth (cf. [5]). However, Aleksandrov solutions exist uniquely when $f > 0$ (cf. [7]). Furthermore, the Aleksandrov solution satisfies the viscosity solution framework for fully nonlinear second order PDEs provided μ is absolutely continuous with respect to $\mathcal{L}^d$ and has a continuous density f. In fact, when $f > 0$, the two solution concepts are equivalent. We do note that other continuous nonconvex solutions of (1) with given Dirichlet data may exist even when $f > 0$, and the MA operator is only degenerate elliptic when restricted to the class of convex functions (cf. [5]).

The MA operator also arises in other application areas such as Riemannian geometry and optimal mass transport. Suppose a hypersurface of $\mathbb{R}^{d+1}$ is the graph of some function u such that, at each point of the surface, the Gauss curvature equals to a prescribed constant K. Then, we have u satisfies the second order fully nonlinear PDE

$$\det\left(D^2u\right) = K\left(1 + |\nabla u|^2\right)^{(d+2)/2},$$

referred to as the prescribed Gauss curvature equation.

The Monge-Kantorovich optimal transport equation is another PDE that involves the MA operator. Let two sets $X_1, X_2 \subset \mathbb{R}^d$, with mass density functions f_1, f_2, respectively, have equal mass. Then, the optimal mass-preserving mapping between the two sets, u, subjected to a given positive quadratic cost density involves the fully nonlinear second order PDE constraint equation

$$\det\left(D^2u\right) = \frac{f_1}{f_2}. \tag{3}$$

More information can be found in [11].

When considering the MA equation, two cases are typically considered. When the right hand side is strictly positive (and bounded) the operator is uniformly elliptic and solutions are regular (assuming the domain is strictly convex). When the right hand side touches zero, the operator is degenerate elliptic and solutions can be singular. Naturally, it is considered more challenging to solve the equation numerically in the degenerate elliptic case [1].

We focus on the two-dimensional strong form of the MA equation with Dirichlet boundary conditions

$$\begin{aligned} \det(D^2u(x)) &= f(x), \qquad \text{if } x \in \Omega, \\ u(x) &= g(x), \qquad \text{if } x \in \partial\Omega, \end{aligned} \tag{MA}$$

where D^2u is the Hessian of the function u, g is a continuous function, and $f \geq 0$. Since we will be restricting to domains $\Omega \subset \mathbb{R}^2$, we can rewrite the PDE as

$$\left(\frac{\partial^2 u}{\partial x^2}\frac{\partial^2 u}{\partial y^2} - \left(\frac{\partial^2 u}{\partial x \partial y}\right)^2\right)(x,y) = f(x,y) \qquad \text{in} \quad \Omega \subset \mathbb{R}^2. \tag{4}$$

It has been shown that a unique viscosity solution u exists in the class of continuous convex functions.

2 Viscosity Solutions of the Monge-Ampère Equation

In this section we introduce the concepts of ellipticity and viscosity solution theory. Some partial differential equations do not have a solution in the classical sense, i.e., there is no function smooth enough to satisfy the PDE pointwise. In some cases, such as for the MA equation, the PDEs do have a solution in a generalized sense. These generalized solutions are functions which do not possess the required regularity to satisfy the equation pointwise. The notion of viscosity solutions allows us to make sense of how a non smooth continuous function may solve an elliptic PDE.

In the following, we consider the generic second order fully nonlinear partial differential equation

$$F[u] = F(D^2u(x), \nabla u(x), u(x), x) = 0 \tag{5}$$

defined on an open set $\Omega \subset \mathbb{R}^d$ for $F : \mathbb{R}^{d\times d} \times \mathbb{R}^d \times \mathbb{R} \times \Omega \to \mathbb{R}$ a continuous (nonlinear) function. In order to guarantee the existence and the uniqueness of a solution, it is typically assumed that the operator F is elliptic and satisfies a comparison principle, as reflected by the following definitions:

Definition 1. The operator F in (5) is said to be elliptic if, for all $(p, \lambda, x) \in \mathbb{R}^d \times \mathbb{R} \times \Omega$, there holds

$$F(A, p, \lambda, x) \geq F(B, p, \lambda, x) \qquad \forall A, B \in \mathcal{S}^{d\times d}, \qquad A \geq B,$$

where $\mathcal{S}^{d\times d}$ denotes the set of $d \times d$ symmetric real matrices and $A \geq B$ means $A - B$ is a nonnegative definite matrix.

We note that when F is differentiable, the ellipticity condition can also be defined by requiring that the matrix $\frac{\partial F}{\partial D^2 u}$ is positive semi-definite [5]. Thus, we have the MA equation is elliptic when the solution u is restricted to the class of convex functions.

We now introduce the definition of a viscosity solution.

Definition 2. $u \in A \subset C(\Omega)$ is called a viscosity subsolution of $F[u] = 0$ if $\forall \varphi \in C^2(\Omega)$, when $u - \varphi$ has a local minimum at $x_0 \in \Omega$, there holds

$$F(D^2\varphi(x_0), \nabla\varphi(x_0), u(x_0), x_0) \leq 0.$$

$u \in A \subset C(\Omega)$ is called a viscosity supersolution of $F[u] = 0$ if $\forall \varphi \in C^2(\Omega)$, when $u - \varphi$ has a local maximum at $x_0 \in \Omega$, there holds

$$F(D^2\varphi(x_0), \nabla\varphi(x_0), u(x_0), x_0) \geq 0.$$

$u \in A \subset C(\Omega)$ is called a viscosity solution of $F[u] = 0$ if u is both a viscosity sub and supersolution of $F[u] = 0$.

The above definitions can be informally interpreted as follows. Without loss of generality, we may assume $u - \varphi$ has a local maximum at x_0 with $u(x_0) = \varphi(x_0)$, or $u - \varphi$ has a local minimum at x_0 with $u(x_0) = \varphi(x_0)$. Then, u is a viscosity solution of (5) if for all smooth functions φ such that φ "touches" the graph of u from above at x_0,
we have $F(D^2\varphi(x_0), \nabla\varphi(x_0), u(x_0), x_0) \geq 0$, and for all smooth functions φ such that φ "touches" the graph of u from below at x_0,
we have $F(D^2\varphi(x_0), \nabla\varphi(x_0), u(x_0), x_0) \leq 0$. The informal geometric interpretation of a viscosity solution for second order problems in one dimension are pictured in Figure 1 and 2.

Definition 3. Problem (5) is said to satisfy a *comparison principle* if the following statement holds. For any upper semicontinuous function u and lower semicontinuous function v on $\overline{\Omega}$, if u is a viscosity subsolution and v is a viscosity supersolution of (5), then $u \leq v$ on $\overline{\Omega}$.

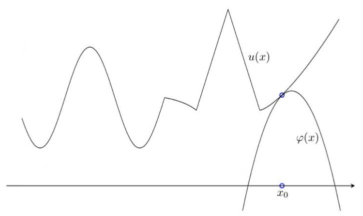


Figure 1: φ is a viscosity subsolution that touches the graph of u from below at x_0 with $F(D^2\varphi(x_0), \nabla\varphi(x_0), x_0) \leq 0$.

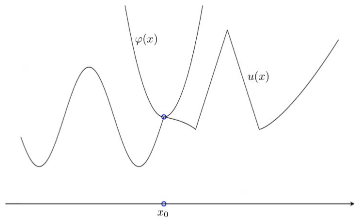


Figure 2: φ is a viscosity supersolution that touches the graph of u from above at x_0 with $F(D^2\varphi(x_0), \nabla\varphi(x_0), x_0) \geq 0$.

3 A Local Convexity Preserving Solver

In this section we present a method for approximating the viscosity solution of the two-dimensional MA equation using a simple discretization and a Gauss-Seidel solver that selects a locally convex solution when it converges. In general, the simple discretization can result in a nonlinear system of algebraic equations with multiple solutions. Thus the solver is designed to alleviate this issue by choosing the correct solution. This section is based on the work of Jean-David Benamou, Brittany D. Froese and Adam M. Oberman in [4]. We first introduce the finite difference notation that will be used throughout the remainder of the paper.

3.1 Difference Operators and Notation

Our goal is to approximate solutions to differential equations, i.e.,to find a function (or some discrete approximation to this function) that satisfies a given relationship between its derivatives on some given region of space along with some boundary conditions along the edges of this domain. In general this is a difficult problem, and only rarely can an analytic formula be found for the solution. A finite difference method proceeds by replacing the derivatives in the differential equation with finite difference approximations. This yields a large but finite algebraic system of equations to be solved in place of the differential equation, something that can be done on a computer.

Before discretizing the MA equation, we first consider the more basic question of how we can approximate the derivatives of a known function by finite difference formulas based only on values of the function itself at discrete points. Let $v(x, y)$ represent a function of two variables that, unless otherwise stated, is assumed to be smooth.

Suppose we want to approximate $D^2 v(x_0, y_0)$ by a finite difference approximation based only on the values of v at a finite number of points near (x_0, y_0). Let $h > 0$. Then, we have the following standard second order accurate central difference operators for approximating v_{xx}, v_{yy}, and v_{xy}, respectively,

$$\begin{aligned}
D^2_{xx,h} v(x_0, y_0) &:= \frac{v(x_0 - h, y_0) - 2v(x_0, y_0) + v(x_0 + h, y_0)}{h^2}, \\
D^2_{yy,h} v(x_0, y_0) &:= \frac{v(x_0, y_0 - h) - 2v(x_0, y_0) + v(x_0, y_0 + h)}{h^2}, \\
D^2_{xy,h} v(x_0, y_0) &:= \frac{v(x_0 - h, y_0 - h) - v(x_0 - h, y_0 + h) - v(x_0 + h, y_0 - h) + v(x_0 + h, y_0 + h)}{4h^2}.
\end{aligned}$$

We then approximate D^2 with the discrete Hessian operator D^2_h defined by

$$\left[D^2_h v(x, y)\right]_{ij} = \begin{cases} D^2_{xx,h} v(x, y) & \text{if } (i, j) = (1, 1), \\ D^2_{yy,h} v(x, y) & \text{if } (i, j) = (2, 2), \\ D^2_{xy,h} v(x, y) & \text{otherwise.} \end{cases} \tag{6}$$

3.2 A Simple Method

We first consider a trivial discretization of the MA equation formed simply by replacing the Hessian operator with the discrete Hessian operator in (6). Here, we assume that the mesh size is uniform and use the grid function $u_{ij} \approx u(x_i, y_j)$. Then, the method is defined by finding the nodal values u_{ij} such that

$$\det\left(D^2_h u_{ij}\right) = f(x_{ij}) \tag{7}$$

for all $x_{ij} \in \mathcal{M}_h \cap \Omega$, where $\mathcal{M}_h$ denotes the underlying grid for $\overline{\Omega}$.

Since we are using central differences, this discretization is consistent with equation (7) and has second order local truncation error.

3.3 Conditional Uniqueness and Multiple Solutions

By the nature of viscosity solutions, we see that an approximation method must be able to capture low-regularity functions. However, to make the situation more difficult, an approximation method must also have a mechanism for filtering the possibly infinite number of lower-regularity functions that satisfy the PDE almost everywhere whenever the viscosity solution has higher regularity. We will see that such low-regularity almost everywhere solutions can correspond to algebraic solutions of the system of equations that results from discretizing a fully nonlinear PDE. These false algebraic solutions are referred to as *numerical artifacts* resulting from the discretization, and these numerical artifacts are known to plague the numerical discretization of fully nonlinear PDEs when using or adapting standard numerical methods for linear, semi-linear, and quasi-linear PDEs. In the context of the MA equation, this issue arises due to the conditional uniqueness of the solution based on restricting the solution space to the class of convex functions. We can see this when considering the simple FD method defined by (7).

A good example of numerical artifacts can be found in [2], where the MA equation in two-dimensions with a C^∞ solution is approximated on the unit square with the simple nine-point FD method defined by (7). By using a Newton solver and varying the initial guess, the authors demonstrate the ability to capture multiple false solutions that are not locally convex, as seen in Figure 2. Thus, either a new discretization or a special solver is necessary.

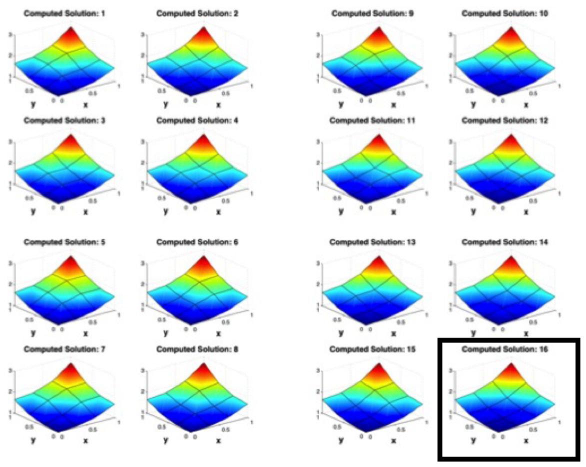


Figure 3: All $2^{(N-2)^2}$ numerical solutions when using an $N \times N$ grid with $N = 4$. Only one of the solutions (inside the box) corresponds to the C^∞ convex solution of the PDE.

3.4 A Convexity-Preserving Solver

While using the simple discretization in (7), the authors of [8] designed a local convexity preserving Gauss-Seidel solver. Pick an index pair (i, j). Let a_1, a_2, a_3, and a_4 be defined by

$$\begin{aligned} a_1 &= \frac{1}{2}\left(u_{i+1,j} + u_{i-1,j}\right), \\ a_2 &= \frac{1}{2}\left(u_{i,j+1} + u_{i,j-1}\right), \\ a_3 &= \frac{1}{2}\left(u_{i+1,j+1} + u_{i-1,j-1}\right), \\ a_4 &= \frac{1}{2}\left(u_{i-1,j+1} + u_{i+1,j-1}\right). \end{aligned} \tag{8}$$

Now we can rearrange the equation (7) as a quadratic form for u_{ij} as follows:

$$4u_{ij}^2 - 4(a_1 + a_2)u_{ij} + 4a_1a_2 - \frac{1}{4}(a_3 - a_4)^2 - h^4 f_{ij} = 0.$$

In order to select the locally convex solutions, we choose u_{ij} by

$$u_{ij} = \frac{1}{2}(a_1 + a_2) - \frac{1}{2}\sqrt{(a_1 - a_2)^2 + \frac{1}{4}(a_3 - a_4)^2 + h^4 f_{ij}}. \tag{9}$$

Observe that, by the definitions of a_1 and a_2, there holds

$$u_{ij} \leq \frac{u_{i,j+1} + u_{i,j-1}}{2}$$

and

$$u_{ij} \leq \frac{u_{i+1,j} + u_{i-1,j}}{2}$$

since $f \geq 0$. Thus, the choice in (9) preserves the local convexity of the function in the Cartesian directions. Using (9) to define a fixed-point iteration, we can naturally define a corresponding Gauss-Seidel solver.

4 A New Finite Difference Method and a Solver

In this section we present a new finite difference (FD) method for solving the two dimensional MA problem. The method is based on using a numerical moment to possibly remove many of the numerical artifacts without having to rely upon monotonicity requirements that lead to wide-stencil approximations. We will test the method along with a new solver based on the Gauss-Seidel iteration introduced in Section 4.3.

4.1 Formulation

In this section we formulate the new FD method. The method is based on adding four extra nodes to the local stencil of the simple FD method in Section 3.4. The extra nodes in the Cartesian directions smooth the approximation as a way to remove numerical artifacts corresponding to lower regularity solutions of the PDE. The scheme finds a grid function u_{ij} such that

$$\det\left(D_h^2 u_{ij}\right) + \gamma\left(\Delta_{2h} - \Delta_h\right) u_{ij} = f(x_{ij}), \tag{10}$$

where $f \geq 0$, $\gamma \geq 0$, and

$$\begin{aligned}\Delta_{2h} u_{ij} &= \frac{1}{4h^2}\{u_{i+2,j} + u_{i-2,j} - 2u_{ij}\} + \frac{1}{4h^2}\{u_{i,j+2} + u_{i,j-2} - 2u_{ij}\},\\ \Delta_h u_{ij} &= \frac{1}{h^2}\{u_{i+1,j} + u_{i-1,j} - 2u_{ij}\} + \frac{1}{h^2}\{u_{i,j+1} + u_{i,j-1} - 2u_{ij}\}.\end{aligned}$$

We also have the Dirichlet boundary condition $u(x_{ij}) = g(x_{ij})$ and auxiliary boundary conditions $D^2_{xx,h} u_{ij} = 0$ if $\mathbf{e}_1$ is normal to $\partial\Omega$ and $D^2_{yy,h} u_{ij} = 0$ if $\mathbf{e}_2$ is normal to $\partial\Omega$. The auxiliary boundary conditions are needed to account for the ghost nodes introduced by the operator Δ_{2h}.

4.2 The Numerical Moment

In this section, we focus on the numerical moment term $(\Delta_{2h} - \Delta_h)\, u_{ij}$. To this end, we use Taylor's expansions to show that the numerical moment approximates a fourth-order differential operator scaled by h^2.

Lemma 4.1. For $v \in C^4(\mathbb{R}^d)$ and $h > 0$, there holds

$$(\Delta_{2h} - \Delta_h)\, v(x) = \frac{1}{4} \sum_{i=1}^{d} h_i^2 v_{x_i x_i x_i x_i}(x) + \mathcal{O}(h^3).$$

Proof. Pick an index $i \in \{1, 2, \ldots, d\}$. There holds

$$\left(D^2_{x_i x_i, 2h_i} - D^2_{x_i x_i, h_i}\right) v(x) = \tfrac{1}{4h_i^2} \left(v(x + 2h_i e_i) - 4v(x + h_i e_i) + 6v(x) - 4v(x - h_i e_i) + v(x - 2h_i e_i)\right). \tag{11}$$

By using Taylor's theorem for each term in equation (11), we have

$$\begin{aligned}
v(x + 2h_i e_i) &= v(x) + 2h_i v_{x_i}(x) + 2h_i^2 v_{x_i x_i}(x) \\
&\quad + \frac{4}{3} h_i^3 v_{x_i x_i x_i}(x) + \frac{2}{3} h_i^4 v_{x_i x_i x_i x_i}(x) + \mathcal{O}(h_i^5), \\
v(x + h_i e_i) &= v(x) + h_i v_{x_i}(x) + \frac{1}{2} h_i^2 v_{x_i x_i}(x) + \frac{1}{6} h_i^3 v_{x_i x_i x_i}(x) \\
&\quad + \frac{1}{24} h_i^4 v_{x_i x_i x_i x_i}(x) + \mathcal{O}(h_i^5), \\
v(x - h_i e_i) &= v(x) - h_i v_{x_i}(x) + \frac{1}{2} h_i^2 v_{x_i x_i}(x) - \frac{1}{6} h_i^3 v_{x_i x_i x_i}(x) \\
&\quad + \frac{1}{24} h_i^4 v_{x_i x_i x_i x_i}(x) + \mathcal{O}(h_i^5), \\
v(x - 2h_i e_i) &= v(x) - 2h_i v_{x_i}(x) + 2h_i^2 v_{x_i x_i}(x) \\
&\quad - \frac{4}{3} h_i^3 v_{x_i x_i x_i}(x) + \frac{2}{3} h_i^4 v_{x_i x_i x_i x_i}(x) + \mathcal{O}(h_i^5).
\end{aligned}$$

By plugging the equations into (11), there holds

$$(\Delta_{2h} - \Delta_h)\, v(x) = \frac{1}{4} \sum_{i=1}^{d} h_i^2 \frac{\partial^4 v(x)}{\partial x_i^4} + \mathcal{O}(h_i^3).$$

$\square$

Thus, by Lemma 4.1, we have the proposed finite difference method is a direct realization of the vanishing moment method of Feng and Neilan (see [3]).

4.3 A Modified Gauss-Seidel Solver

In this section we discuss how to update the Gauss-Seidel solver to find the solution for equation (10) in Section 4.1. Let

$$\begin{aligned} a_5 &= (u_{i+2,j} + u_{i-2,j}), \\ a_6 &= (u_{i,j+2} + u_{i,j-2}). \end{aligned}$$

Rearranging the terms in (10), we have

$$u_{ij}^2 - \left(a_1 + a_2 + \tfrac{3\gamma h^2}{4}\right) u_{ij} - \tfrac{1}{16} h^2 \gamma\, (a_1 + a_2 - 8a_5 - 8a_6) - \tfrac{1}{16}(a_3 - a_4)^2 + a_1 a_2 - \tfrac{1}{4} h^4 f_{ij} = 0.$$

Again, we choose the minus sign when solving the quadratic equation for u_{ij} yielding

$$u_{ij} = \frac{a_1 + a_2 + \frac{3\gamma h^2}{4}}{2} - \frac{\sqrt{\frac{9}{16} h^4 \gamma^2 + \frac{3}{2}(a_1 + a_2) h^2 \gamma + (a_1 - a_2)^2 + \frac{1}{4}(a_3 - a_4)^2 + h^4 f_{ij} + \frac{h^2}{4}\gamma(a_5 + a_6 - 8a_1 - 8a_2)}}{8}. \tag{12}$$

Note that in order for equation (12) to make sense, we need to ensure that the discriminant is nonnegative. Observe that the discriminant is a quadratic, concave up function with respect to γ. Thus, for γ large enough, we can guarantee that the discriminant is positive for all indices (i, j).

We now consider how to use a Gauss-Seidel based solver to find the fixed-point of (12) while simultaneously finding $\gamma > 0$ sufficiently large to ensure the discriminant in (12) is nonnegative for the fixed point. To this end, we consider a three-phase approach. In the first phase we choose $\gamma >> 0$ and seek a fixed-point of (12) using a Gauss-Seidel update. An error flag is recorded if the discriminant is ever negative during the iteration. The first phase finds a good initial guess for the second phase that starts with a given value of γ and increases it whenever the discriminant is negative. Note that the second phase does not ensure a single value for γ works at every node since it simply updates the value for γ as the solver progresses through the domain. Finally, phase 3 verifies that the largest value of γ used in phase 2 can be used uniformly for all nodes and checks for convergence.

For the first phase we use the stopping criteria

$$\max_{ij}|u_{ij}^{(k)} - u_{ij}^{(k-1)}| < \text{Tol.}$$

For the second phase we use the stopping criteria

$$\max|\gamma^{(k)} - \gamma^{(k-1)}| + \max_{ij}|u_{ij}^{(k)} - u_{ij}^{(k-1)}| < \text{Tol.}$$

For the third phase we use the stopping criteria

$$\max_{ij}|u_{ij}^{(k)} - u_{ij}^{(k-1)}| < \text{Tol.}$$

Note that γ is fixed in phase 1 and phase 3.

We now consider how the choice for γ is updated in phase 2. Let $d(\gamma)$ denote the discriminant in (12). Since the leading coefficient is positive we can guarantee that the graph of $d(\gamma)$ is concave up. We illustrate possible graphs for $d(\gamma)$ in Figure 4. Graph (1),(2), and (3) are possible graphs for $d(\gamma)$ that would imply the current value $\gamma > 0$ is sufficient since $d(\gamma) > 0$. In a situation such as that represented by graph (4), we choose γ to be the biggest zero of the graph if it is bigger than the current value for γ.

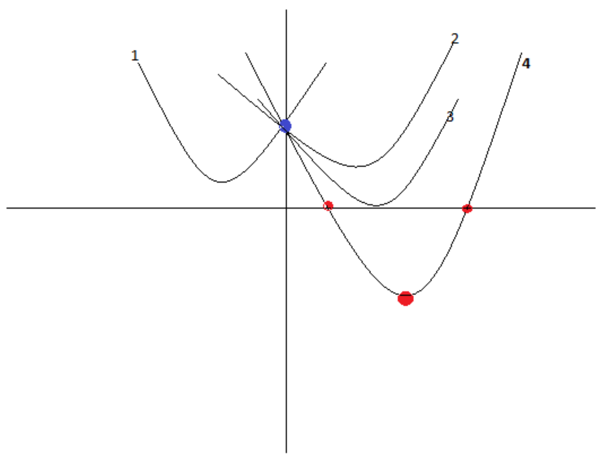


Figure 4: Possible graphs for the discriminant $d(\gamma)$.

4.4 Numerical Tests and Results

We now implement and test the new FD method and solver defined in Section 4.1 and 4.3. All of the tests were performed in Matlab using the initial guess $u^{(0)}$ defined by $u_{ij}^{(0)} = 0$ if $x_{ij} \in \mathcal{M}_h \cap \Omega$ and $u^{(0)}$ satisfies the Dirichlet boundary condition defined by (10) and the auxiliary boundary condition defined by (10) for the ghost nodes. The tolerance for the stopping criteria was set to be 10^{-12}. We record the error for the approximation as well as the norm of the residual to check for convergence of the solver.

Example 4.1. Consider the MA problem (10) with
$f(x,y) = (1 + x^2 + y^2)e^{(\frac{x^2+y^2}{2})}$ $, \Omega = (0,1) \times (0,1)$, and g chosen such that the *viscosity solution* is given by $u(x,y) = e^{(\frac{x^2+y^2}{2})}$.
The numerical results are recorded in Table 1 and Figure 5. Here we observe that as the grid size increases the errors decrease and the number of iterations necessary for the Gauss-Seidel solver to converge increases.

Example 1 Results					
Pass	Nodes	Iterations	Gamma	$\|\|\text{Residual}\|\|_\infty$	Error
1	25	1057	5025	2.534757e-007	2.647816e-001
	100	10220	5025	9.354102e-007	2.741542e-001
	225	42002	5025	1.926306e-006	2.717076e-001
	400	116457	5025	3.349229e-006	2.665818e-001
2	25	616	25	1.335154e-009	1.324966e-001
	100	2680	25	4.832378e-009	5.411973e-002
	225	5950	25	1.041267e-008	2.659325e-002
	400	10381	25	1.808623e-008	1.584917e-002
3	25	1	25	1.285617e-009	1.324966e-001
	100	1	25	4.795593e-009	5.411973e-002
	225	1	25	1.038584e-008	2.659325e-002
	400	5	25	1.766768e-008	1.584917e-002

Table 1: Numerical results for Example 4.1.

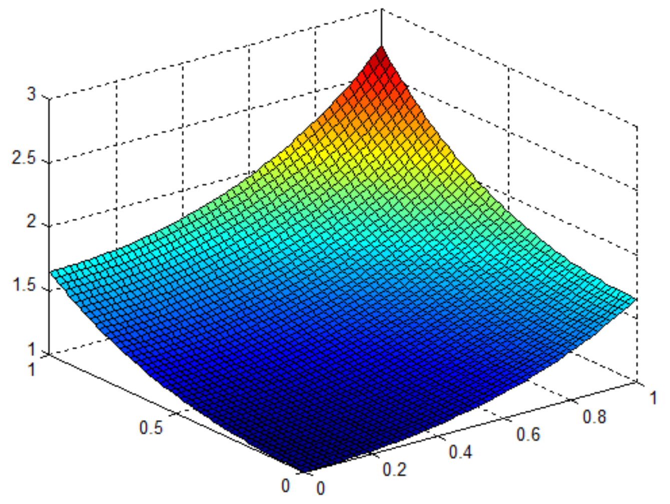


Figure 5: Approximation for Example 4.1 using h = 1.960784e-002 and γ= 25.

Example 4.2. Consider the MA problem (10) with

$$f(x,y) = \begin{cases} 1 & \text{if} \quad \frac{(x-0.5)^2+(y-0.5)^2}{2} > 0.16^2, \\ 0 & \quad \text{otherwise,} \end{cases}$$

$\Omega = (0,1) \times (0,1)$, and g chosen such that the *viscosity solution* is given by $u(x,y) = \max\{\frac{(x-0.5)^2+(y-0.5)^2}{2}, 0.008\}$. Note that f is discontinuous and zero-valued in parts of the domain. The numerical results are recorded in Table 2 and Figure 6.

Example 2 Results					
Pass	Grid	Iterations	Gamma	$\|\|$Residual$\|\|_\infty$	Error
1	25	1023	5025	6.401274e-008	6.006634e-001
	100	9937	5025	2.336658e-007	5.986595e-001
	225	41006	5025	4.829705e-007	6.070408e-001
	400	114412	5025	8.396559e-007	5.993812e-001
2	25	780	25	3.215962e-010	4.124437e-001
	100	4251	25	1.191539e-009	2.130046e-001
	225	9815	25	2.499138e-009	1.139827e-001
	400	17197	25	4.354597e-009	6.945773e-002
3	25	1	25	3.124320e-010	4.124437e-001
	100	1	25	1.185120e-009	2.130046e-001
	225	1	25	2.493969e-009	1.139827e-001
	400	1	25	4.349814e-009	6.945773e-002

Table 2: Numerical results for Example 4.2.

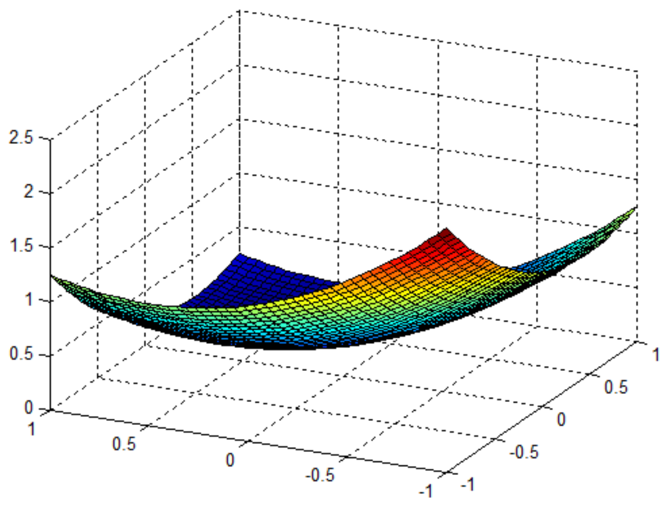
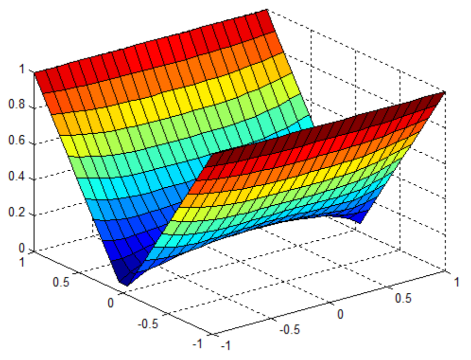
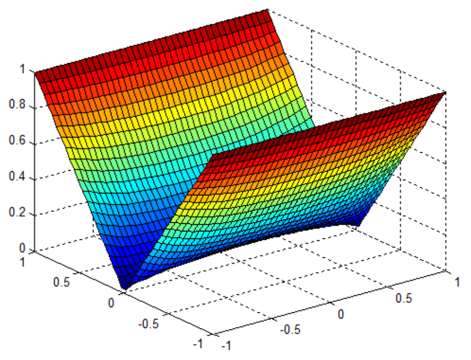


Figure 6: Approximation for Example 4.2 using h = 4.878049e-002 and γ= 25.

Example 4.3. Consider the MA problem (10) with $f(x, y) = 0$,$\Omega = (0, 1) \times (0, 1)$, and g chosen such that the *viscosity solution* is given by $u(x, y) = |x|$. Here we have a lower regularity solution $u \in C^0\backslash C^1$. Note that the approximations graphed in Figure 7 are not convex for a given value of h. However, they do appear to be converging to the convex viscosity solution. The approximations appear to have been artificially smoothed due to the presence of the numerical moment. We hypothesize this is how the method removes numerical artifacts.



(a) Nx=Ny=5



(b) Nx=Ny=20

Figure 7: Approximation for Example 4.3 using γ= 25.

Example 3 Results					
Pass	Grid	Iterations	Gamma	$\|\|\text{Residual}\|\|_\infty$	Error
1	25	1019	5025	6.374959e-008	6.908513e-001
	100	9873	5025	2.356325e-007	6.089617e-001
	225	40670	5025	4.808208e-007	6.978875e-001
	400	113358	5025	8.404391e-007	6.462957e-001
2	25	783	25	3.189340e-010	5.289891e-001
	100	4391	25	1.183082e-009	2.700520e-001
	225	9851	25	2.500846e-009	2.282120e-001
	400	17425	25	4.352363e-009	1.441428e-001
3	25	1	25	3.100413e-010	5.289891e-001
	100	1	25	1.177088e-009	2.700520e-001
	225	1	25	2.495838e-009	2.282120e-001
	400	1	25	4.346500e-009	1.441428e-001

Table 3: Numerical results for Example 4.3.

5 Conclusion

We proposed a new FD method that uses numerical moments to tackle the conditional uniqueness of viscosity solutions for the MA equation. We also proposed a 3-phase Gauss-Seidel solver that helps determine a sufficiently large coefficient for the numerical moment. The method appears to converge even for degenerate problem. The solver appears reliable in our numerical tests that indicated that the sufficient number of iterations appears to increase with the grid size. A future direction is to use multi-grid methods to limit the number of iterations needed since the iteration could be used in the smoothing step.

References

[1] Edward J Dean and Roland Glowinski. Numerical methods for fully nonlinear elliptic equations of the monge–ampère type. *Computer methods in applied mechanics and engineering*, 195(13-16):1344–1386, 2006.

[2] Xiaobing Feng, Roland Glowinski, and Michael Neilan. Recent developments in numerical methods for fully nonlinear second order partial differential equations. *SIAM Review*, 55(2):205–267, 2013.

[3] Xiaobing Feng and Michael Neilan. Vanishing moment method and moment solutions for fully nonlinear second order partial differential equations. *Journal of Scientific Computing*, 38(1):74–98, 2009.

[4] 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. *SIAM Journal on Numerical Analysis*, 49(4):1692–1714, 2011.

[5] David Gilbarg and Neil S Trudinger. Elliptic partial differential equations of second order. 2001.

[6] John W Milnor and David W Weaver. *Topology from the Differentiable Viewpoint; Based on Notes by DW Weaver*. 1965.

[7] Michael Neilan. A nonconforming morley finite element method for the fully nonlinear monge-ampère equation. *Numerische Mathematik*, 115(3):371–394, 2010.

[8] Adam M Oberman. Wide stencil finite difference schemes for the elliptic monge-ampere equation and functions of the eigenvalues of the hessian. *Discrete Contin. Dyn. Syst. Ser. B*, 10(1):221–238, 2008.

[9] AV Pogorelov. Monge-ampere equations of elliptic type, translated from the first russian edition by leo f. *Boron with the assistance of Albert L. Rabenstein and Richard C. Bollinger, P. Noordhoff, Ltd., Groningen*, 1964.

[10] AV Pogorelov. Extrinsic geometry of convex surfaces, volume 35 of translations of mathematical monographs. *American Mathematical Society, Providence, RI*, 1973.

[11] Cédric Villani. *Topics in optimal transportation.* Number 58. American Mathematical Soc., 2003.

6 Appendix

Here we have included Matlab code for Example 4.1 found in Section 4.4. The code includes separate .m files for the following parts of the code:

- First phase
- Second phase
- Third phase
- Initial boundary conditions
- Ghost values
- Convex function
- Exact value
- Residual values

The code records the mesh size h, iteration count for each pass, γ, the residual for the approximation, and the approximation error.

1st pass

```
clear;
clc;

% Boundary conditions
ax = -1;

bx = 1;

ay = -1;

by = 1;

%Grid points
Nx = 5;

Ny = 5;

%Mesh size   --- Nx interior points in x-direction, Ny interior points in
%y-direction

hx = (bx-ax)/(Nx+1);

hy = (by-ay)/(Ny+1);

if (hx ~= hy)

    fprintf('Need hx = hy.\n\n');

end

h = hx;

fprintf('\nNx = %i:\nh = %d.\n',Nx,h);

%Introduced Gamma  (scalar)

gamma1 = 25.0;

gamma = gamma1 + 5000;

N=0;

%x and y mesh

x_mesh=ax-hx:hx:bx+hx;

y_mesh=ay-hy:hy:by+hy;

% Ghost point index = 1, Nx+4 or Ny+4;
% Boundary = 2, Nx+3 or Ny+3

Tol=10^(-12);
```

```
%Initial value
u=zeros(length(x_mesh),length(y_mesh));
u=initialize_BC(u,Nx,Ny,x_mesh,y_mesh);
u=update_ghost(u,Nx,Ny);

u2=u;

u2(3,3) = Tol+1;

D = u2(3:Nx+2,3:Ny+2)-u(3:Nx+2,3:Ny+2);    % Contains interior nodes only
for u2-u
gamma2 = gamma;

while (max(abs(D(:))) > Tol)

    u2 = u;

    for i=3:Nx+2

        for j=3:Ny+2

            a1=(u(i+1,j)+u(i-1,j))/2;

            a2=(u(i,j+1)+u(i,j-1))/2;

            a3=(u(i+1,j+1)+u(i-1,j-1))/2;

            a4=(u(i-1,j+1)+u(i+1,j-1))/2;

            a5=(u(i+2,j)+u(i-2,j));

            a6=(u(i,j+2)+u(i,j-2));

            %Vertex

            a = 9*h*h*h*h/16;
            b = 1.5*(a1+a2)*h*h + (h*h/4)*(a5+a6-8*a1-8*a2);
            c = (a1-a2)*(a1-a2) + (1/4)*(a3-a4)*(a3-a4) +
                h*h*h*h*Test4(x_mesh(i), y_mesh(j));

            vx = -b/(2*a);

            %v = gamma_quad(vx);
            v = a*vx*vx + b*vx + c;

            if (v<10^(-12))

                d = b*b-4*a*c;
                d = d+10^(-11);

                if (d < 0)
                    fprintf('d < 0.\n\n');
                    gamma_temp = vx+10^(-6);
                else

                    gamma_temp = vx + sqrt(d)/(2*a) + 10^(-10);
                end

                disc_temp = a*gamma_temp*gamma_temp + b*gamma_temp + c;

                if (disc_temp < 0)
                    fprintf('Error in disc_temp.\n\n');
                end

                if (gamma_temp > gamma2)

                    gamma2 = gamma_temp;

                    if (gamma2 < 0)
                        fprintf('Error in gamma2 < 0\n\n');
                    end

                end

            end

            if (gamma2 > gamma)

                gamma=gamma2 + 10^(-6);

            else

                gamma2 = gamma;

            end

            disc = a*gamma*gamma + b*gamma + c;

            if (disc < 0)
                fprintf('disc2 < 0.\n\n');
            end

            u(i,j)= 0.5*(a1+a2+(3/4)*gamma*h*h) - 0.5*sqrt(disc);

        end

    end

    u = update_ghost(u,Nx,Ny);

    D = u2(3:Nx+2,3:Ny+2)-u(3:Nx+2,3:Ny+2);

    N=N+1;

end

fprintf('First pass N = %i.\n',N);
fprintf('Gamma = %d after first pass.\n',gamma);
```

```
error = 0.0;

for i=3:Nx+2

    for j=3:Ny+2

        error2 = abs( u(i,j) - u_soln(x_mesh(i),y_mesh(j)));
        if (error2 > error)
            error = error2;
        end

    end

end


value=0.0;

for i=3:Nx+2

    for j=3:Ny+2

        value2=abs(PDE_F(u,i,j,x_mesh(i),y_mesh(j),h) -
            gamma*(1/4/h/h)*(...

            u(i-2,j)-4*u(i-1,j)+6*u(i,j)-4*u(i+1,j)+u(i+2,j)+...
            u(i,j-2)-4*u(i,j-1)+6*u(i,j)-4*u(i,j+1)+u(i,j+2)));

        if (value2>value)

            value=value2;

        end


    end
end


fprintf('Solver Value = %d.\n',value);
fprintf('Error = %d.\n\n\n',error);

%%%%%

ala_pass2;

error = 0.0;
error2 = 0.0;

for i=3:Nx+2

    for j=3:Ny+2

        error2 = abs( u(i,j) - u_soln(x_mesh(i),y_mesh(j)));
        if (error2 > error)
            error = error2;
        end
    end

end


value=0.0;

value2=0.0;

for i=3:Nx+2

    for j=3:Ny+2

        value2=abs(PDE_F(u,i,j,x_mesh(i),y_mesh(j),h) -
            gamma*(1/4/h/h)*(...

            u(i-2,j)-4*u(i-1,j)+6*u(i,j)-4*u(i+1,j)+u(i+2,j)+...
            u(i,j-2)-4*u(i,j-1)+6*u(i,j)-4*u(i,j+1)+u(i,j+2)));

        if (value2>value)

            value=value2;

        end


    end
end

fprintf('Solver Value = %d.\n',value);
fprintf('Error = %d.\n\n',error);

%%%

ala_pass3;

error = 0.0;
error2 = 0.0;

for i=3:Nx+2

    for j=3:Ny+2

        error2 = abs( u(i,j) - u_soln(x_mesh(i),y_mesh(j)));
        if (error2 > error)
            error = error2;
        end

    end

end


value=0.0;
```

```
value2=0.0;

for i=3:Nx+2

    for j=3:Ny+2

        value2=abs(PDE_F(u,i,j,x_mesh(i),y_mesh(j),h) -
            gamma*(1/4/h/h)*(...

            u(i-2,j)-4*u(i-1,j)+6*u(i,j)-4*u(i+1,j)+u(i+2,j)+...
            u(i,j-2)-4*u(i,j-1)+6*u(i,j)-4*u(i,j+1)+u(i,j+2)));

        if (value2>value)

            value=value2;

        end


    end
end

fprintf('Solver Value = %d.\n',value);
fprintf('Error = %d.\n',error);

surf(x_mesh(2:Nx+3), y_mesh(2:Ny+3), u(2:Nx+3,2:Ny+3));
```

2nd pass

```
% Second Pass to decrease gamma and make uniform
uu = u;

u2(3,3) = Tol+1;
D = u2(3:Nx+2,3:Ny+2)-u(3:Nx+2,3:Ny+2);    % Contains interior nodes only
for u2-u
gamma = gamma1;
N2 = 0;
gamma2 = gamma;
gammaold = gamma;
d_flag = 0;

while (max(abs(D(:))) + abs(max(gammaold-gamma)) > Tol)

    gammaold = gamma;
```

```
u2 = u;

gamma_temp = -1;

disc_gamma = -1.0;

for i=3:Nx+2

    for j=3:Ny+2

        a1=(u(i+1,j)+u(i-1,j));

        a2=(u(i,j+1)+u(i,j-1));

        a3=(u(i+1,j+1)+u(i-1,j-1));

        a4=(u(i-1,j+1)+u(i+1,j-1));

        a5=(u(i+2,j)+u(i-2,j));

        a6=(u(i,j+2)+u(i,j-2));

        %Vertex

        a = 9*h*h*h*h;
        b = 12*(a1+a2)*h*h + 4*h*h*(a5+a6-4*a1-4*a2);
        c = 4*(a1+a2)*(a1+a2) - 16*a1*a2 + (a3-a4)*(a3-a4) +
            16*h*h*h*h*Test4(x_mesh(i), y_mesh(j));

        disc = a*gamma*gamma + b*gamma + c;

        while (disc < 0)
            gamma = gamma + 1;
            disc = a*gamma*gamma + b*gamma + c;

            if (gamma > 100000)
                fprintf('Gamma > 100000.\n\n');
                u = uu;
                gamma = 2*gamma1;
                i=3;
                j=3;

                a1=(u(i+1,j)+u(i-1,j));
                a2=(u(i,j+1)+u(i,j-1));
                a3=(u(i+1,j+1)+u(i-1,j-1));
                a4=(u(i-1,j+1)+u(i+1,j-1));
                a5=(u(i+2,j)+u(i-2,j));
                a6=(u(i,j+2)+u(i,j-2));
                a = 9*h*h*h*h;
                b = 12*(a1+a2)*h*h + 4*h*h*(a5+a6-4*a1-4*a2);
                c = 4*(a1+a2)*(a1+a2) - 16*a1*a2 + (a3-a4)*(a3-a4) +
                    16*h*h*h*h*f(x_mesh(i), y_mesh(j));
                disc = a*gamma*gamma + b*gamma + c;

            end
        end

        u(i,j)= (1/8)*(2*(a1+a2)+3*gamma*h*h) - (1/8)*realsqrt(disc);
        end

    end

    u = update_ghost(u,Nx,Ny);

    D = u2(3:Nx+2,3:Ny+2)-u(3:Nx+2,3:Ny+2);

    N2=N2+1;

end

fprintf('Second pass N = %i.\n',N2);
if (d_flag == 1)
    fprintf('Pass 2: d < 0.\n');
end
fprintf('Gamma = %d after second pass.\n',gamma);
```

3rd pass

```
u2(3,3) = Tol+1;
D = u2(3:Nx+2,3:Ny+2)-u(3:Nx+2,3:Ny+2);    % Contains interior nodes only
for u2-u
N3 = 0;


while (max(abs(D(:))) > Tol)

    u2 = u;

    for i=3:Nx+2

        for j=3:Ny+2

            a1=(u(i+1,j)+u(i-1,j));

            a2=(u(i,j+1)+u(i,j-1));

            a3=(u(i+1,j+1)+u(i-1,j-1));

            a4=(u(i-1,j+1)+u(i+1,j-1));

            a5=(u(i+2,j)+u(i-2,j));
            a6=(u(i,j+2)+u(i,j-2));

            %Vertex

            a = 9*h*h*h*h;
            b = 12*(a1+a2)*h*h + 4*h*h*(a5+a6-4*a1-4*a2);
            c = 4*(a1+a2)*(a1+a2) - 16*a1*a2 + (a3-a4)*(a3-a4) +
                16*h*h*h*h*Test4(x_mesh(i), y_mesh(j));

            disc = a*gamma*gamma + b*gamma + c;

            if (disc < 0)

                fprintf('u imaginary.\n\n');
                %disc
                %disc = -disc;

%                 while (disc < 0)
%
%                     gamma = gamma + 1;
%                     disc = a*gamma*gamma + b*gamma + c;
%                 end


            end

            u(i,j)= (1/8)*(2*(a1+a2)+3*gamma*h*h) - (1/8)*realsqrt(disc);

        end

    end

    u = update_ghost(u,Nx,Ny);

    D = u2(3:Nx+2,3:Ny+2)-u(3:Nx+2,3:Ny+2);

    N3=N3+1;

end


fprintf('Third pass N = %i.\n',N3);
if (d_flag == 1)
    fprintf('Pass 3: d < 0.\n');
end
fprintf('Gamma = %d after third pass.\n',gamma);
```

Initial Boundary Conditions

```
function u=initialize_BC(u,Nx,Ny,x_mesh,y_mesh)

for j=2:Ny+3

    u(2,j) = g(x_mesh(2), y_mesh(j));
    u(Nx+3,j) = g(x_mesh(Nx+3), y_mesh(j));
end

for i=2:Nx+3

    u(i,2) = g(x_mesh(i), y_mesh(2));
    u(i,Ny+3) = g(x_mesh(i), y_mesh(Ny+3));

end

end
```

Boundary condition

```
function out = g(x,y)

out = u_soln(x,y);

end
```

Ghost value

```
function [u]= update_ghost(u,Nx,Ny)

for j=2:Ny+3

    u(1,j) = 2*u(2,j)-u(3,j);
    u(Nx+4,j) = 2*u(Nx+3,j)-u(Nx+2,j);

end

for i=2:Nx+3

    u(i,1) = 2*u(i,2)-u(i,3);
    u(i,Ny+4) = 2*u(i,Ny+3)-u(i,Ny+2);
   end
end
```

Convex function

```
function [f]=Test4(x,y)

if (x-0.5)*(x-0.5)+(y-0.5)*(y-0.5)>0.16*0.16

    f=1;

elseif (x-0.5)*(x-0.5)+(y-0.5)*(y-0.5)<=0.16*0.16

    f=0;

end

end
```

Exact value

```
function out = u_soln(x,y)

out = max(((x-0.5)*(x-0.5)+(y-0.5)*(y-0.5))/2,0.08);

end
```

Residual Value

```
function out = PDE_F(u,i,j,x,y,h)


out = (1/h/h)*(u(i+1,j)-2.0*u(i,j)+u(i-1,j));
out = out * (1/h/h)*(u(i,j+1)-2.0*u(i,j)+u(i,j-1));
out = out - (1/16/h/h/h/h)*(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));
out = out - Test4(x,y);
```