Efficient Contouring on Unstructured Meshes

by

Hassan Goldani Moghaddam

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Efficient Contouring on Unstructured Meshes

Hassan Goldani Moghaddam

Master of Science

Graduate Department of Computer Science

University of Toronto

2004

A PCI (Pure Cubic Interpolant) is a piecewise bicubic polynomial interpolant over an unstructured mesh based on the DEI (Differential Equation Interpolant) approach. It can approximate the solution of a PDE (Partial Differential Equation) at off-mesh points with the same order of accuracy as those provided by an underlying numerical PDE solver at mesh points. It satisfies the continuity property that is very important in the visualization area. In addition to continuity, the PCI is efficient in terms of time and accuracy. We compare the PCI with the ADEI (Alternate Differential Equation Interpolant) and some other DEI-based interpolants. Furthermore, we introduce three fast contouring algorithms based on the PCI and an underlying unstructured mesh. Unlike standard contouring functions, our contouring algorithms do not need a fine structured approximation and work efficiently with the original scattered data.

Dedication

This thesis would be incomplete without a mention of the support given me by my supervisor, Professor Wayne Enright. I also dedicate it to my parents, for their guidance, support, love and enthusiasm. Without these things this thesis could not have been possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Most Partial Differential Equations (PDEs) do not have a closed form solution. In these cases, numerical methods can be used to approximate the solution at only a discrete set of mesh points in the domain associated with the problem definition. These approximations usually provide more accuracy at the mesh points than is available (using piecewise linear approximation, for example) at off-mesh points. Although many numerical methods have the capability of fixing the location of some or all the mesh points, this can significantly affect the efficiency and accuracy of the computed solution. In other words, these methods are most efficient if they are allowed to adjust dynamically the location of the mesh points to match the local behavior of the solution. Since the best distribution of mesh points is usually non-uniform, when one wants to visualize the solution (one of the important post-processing tasks), some extra data must be generated. One approach is to determine a mesh that is fine enough to visualize the solution and force this mesh to be a subset of the mesh of the numerical method. This approach often results in much more accuracy than what is needed and can be very expensive. To avoid this inefficiency, Enright introduced the Differential Equation Interpolant (DEI) as an alternative [2]. The DEI approximates the solution of a PDE such that the approximations at off-mesh points have the same order of accuracy as those at mesh points provided by the underlying numerical PDE solver [2].

For a two dimensional problem, the DEI is a piecewise bivariate polynomial interpolant that approximates the solution at any point in the domain associated with the problem definition. The number of unknown coefficients determined by the degree and type of the interpolant (tensor product, pure, ...) might be greater than the number of independent linear constraints defined by the information provided by the PDE solver. Consequently, we might require extra independent constraints to uniquely determine the associated piecewise polynomial. The standard approach for determining extra constraints is based on continuity of higher derivatives (smoothness of the piecewise polynomial) at mesh points, whereas with the DEI approach these extra constraints are based on almost satisfying the PDE at a prescribed set of 'collocation' points. The DEI generates more accurate but less smooth global approximations [2].

In the case that the underlying numerical method produces approximations on an unstructured mesh, the DEI can be considered to be a scattered data interpolant. In a more general setting, scattered data either can be produced by a numerical PDE solver or can be real data from other sources such as measurement for experiments. In [4], Ramos and Enright presented a valuable and important study of this problem. They introduced the Alternate Differential Equation Interpolant (ADEI), a piecewise polynomial interpolant over a scattered data set. Instead of choosing random points or fixed points, they developed an iterative algorithm to find suitable collocation points based on the magnitude of the coefficients of the resulting interpolant. Although the ADEI almost provides a smooth surface, it can still suffer from the appearance of discontinuities. Moreover, finding a suitable set of collocation points may take a long time. We will introduce the PCI to overcome these difficulties.

After approximating the solution at a discrete set of mesh points by a numerical PDE solver, one might plot contour lines in order to visualize the solution. Standard contouring algorithms require knowing the function to be contoured on a regular spaced rectangular mesh. In cases where the original mesh is unstructured, a regular rectangular mesh must be introduced first. In addition, most standard contouring algorithms use linear interpolation inside each element. In these algorithms the more mesh points we have, the more accurate contour plot we get. Consequently, to obtain a smooth contour plot, the numerical method should provide the approximations of the solution at a very fine mesh. In the case that we have an interpolant such as the DEI, we can obtain the refined mesh data directly from the DEI. Bradbury and Enright investigated the application of the DEI to visualize the solution of PDEs using this idea when the underlying mesh is rectangular and structured [1]. In addition, they introduced three fast algorithms to compute contour lines efficiently directly from the DEI. In this thesis, we will introduce such fast algorithms to compute contour lines directly from the PCI. In addition, we will improve these algorithms to overcome some of the difficulties that Bradbury and Enright identified.

1.2 Objectives

In this thesis, we will focus on two particular tasks:

- Introducing the PCI, an interpolant for scattered data that accurately approximates the solution of the PDE at off-mesh points and overcomes some deficiencies of the ADEI (discontinuities and excessive compute time).

- Introducing three fast contouring algorithms based on the PCI.

This thesis is organized as follows. In chapter 2, some previous and related work is presented. We will then introduce the PCI in chapter 3 and our contouring algorithms in chapter 4. Finally, in chapter 5, the numerical results for both tasks are presented.

Chapter 2

Previous and Related Work

In this chapter, an overview of some previous work will be presented.

2.1 Differential Equation Interpolant (DEI)

In [2], Enright introduced the DEI, an approach to accurately approximate the solution of a Partial Differential Equation (PDE) at off-mesh points. Although the approach can be applied to any dimension, we will restrict our attention to two dimensional problems in this thesis. These approximate solutions at off-mesh points are designed to have the same order of accuracy as that provided by the numerical PDE solver at mesh points [2]. The idea is to allocate a multi-variate polynomial to each mesh element and consequently, the collection of such polynomials over all mesh elements will define a piecewise polynomial approximation. The mesh elements can be rectangles or triangles depending on the distribution of the associated mesh points. For each element e, the bivariate polynomial $p_{d,e}(x, y)$ is represented by $(d + 1)^2$ unknown coefficients,

$$p_{d,e}(x, y) = \sum_{i=0}^{d} \sum_{j=0}^{d} c_{ij} s^i t^j, \tag{2.1}$$

where

$$s = \frac{(x - x_1)}{D_1}, t = \frac{(y - y_1)}{D_2} \tag{2.2}$$

and D_1 and D_2 depend on the size of the mesh element e in the x and y direction, d is the degree and (x_1, y_1) is a mesh point associated with e (or the lower left corner of the rectangle defined by e for triangular meshes).

In [2], the underlying PDE was assumed to be a two-dimensional, second-order problem of the form

$$Lu = g(x, y, u, u_x, u_y), \tag{2.3}$$

where L is a given semi-linear differential operator of the form

$$L = a_1(x,y)\frac{\partial^2}{\partial x^2} + a_2(x,y)\frac{\partial^2}{\partial y^2} + a_3(x,y)\frac{\partial^2}{\partial x \partial y}. \tag{2.4}$$

The approach can be extended in an obvious way to higher dimensional problems as well as higher order problems. Enright also considered *pure* bivariate polynomial approximation by restricting the approximating polynomial to be a bivariate polynomial of total degree d. A *pure* bivariate polynomial is represented by

$$\hat{p}_{d,e}(x, y) = \sum_{i=0}^{d} \sum_{j=0}^{d-i} c_{ij} s^i t^j. \tag{2.5}$$

One of the advantages of such a polynomial is that there are fewer unknown coefficients to be determined for the same order of accuracy. The pure bivariate polynomial has $\frac{(d+1)(d+2)}{2}$ coefficients rather than the usual $(d+1)^2$. Another advantage of a pure bivariate polynomial interpolant that could be very useful for rendering and visualization is that better continuity properties may result.

Enright considered, as a specific example, the two-dimensional operator $L = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ and investigated how the approach could be effectively implemented. He identified the maximum number of independent collocation constraints for different degree interpolations. An extension of the approach for three-dimensional operator, $L = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$, was also investigated [2].

In chapter 3, we will introduce the Pure Cubic Interpolant (PCI) that is based on the pure bivariate approach over an unstructured triangular mesh with d=3.

2.2 Alternate Differential Equation Interpolant (ADEI)

The ADEI, introduced by Ramos and Enright [4], is an interpolant based on the DEI approach with some modifications in order to obtain better performance on triangular meshes. Ramos and Enright investigated the performance of the DEI on scattered data and unstructured triangular meshes. They found that the associated DEI could occasionally exhibit some noticeable discontinuities on the edges of some mesh elements. They analyzed the problem and observed that such discontinuities are usually associated with a large value of the norm of coefficients [4]. An algorithm was developed to locate elements with considerably large value of this norm and then try to decrease the associated norm for such elements by iteratively choosing alternative sets of collocation points. They compared the results of the ADEI with two algorithms from the ACM repository (acm752 and acm790) and showed that the ADEI approach produces comparable or more accurate results than the other two algorithms [4].

Although the ADEI produces a piecewise polynomial that is relatively smooth, it doesn't necessary generate a continuous piecewise polynomial interpolant. In addition, the iteration that is used to find the improved set of collocation points sometimes takes a long time. To address these disadvantages of the ADEI, we will consider the use of a PCI in chapter 3.

2.3 Contouring Problem

One way to visualize the approximate solution obtained by a numerical PDE solver is to draw contour lines of the associated approximate solution. Contour lines help us to investigate the behavior of the solution and to analyze important characteristics of approximate solutions.

In order to have a relatively smooth contour plot, a fine mesh approximation of the solution is needed. One way to obtain such a fine mesh approximation is to require the numerical method to solve the PDE on this fine mesh. Such a requirement might result in more accuracy than what is necessary and can often be very expensive. A better approach is to solve the problem by the numerical method for the requested accuracy on an associated coarse mesh and then generate a fine mesh approximation from the coarse mesh approximation by using the DEI associated with the underlying PDE and the coarse mesh. Bradbury and Enright [1] investigated this approach (MATLAB/DEI approach) to generate a fine mesh approximation and then plotted contour lines using a standard contouring algorithm like the built-in MATLAB function *contour*. In addition, they introduced three fast direct contouring algorithms based on the use of the DEI that avoid the introduction of a fine mesh. To illustrate the fast contouring algorithms, they considered an underlying parabolic PDE of the form

$$Lu = g(x, y, u, u_x), \tag{2.6}$$

$$L = \frac{\partial}{\partial y} - \beta(x, y)\frac{\partial^2}{\partial x^2} \tag{2.7}$$

where the DEI is a piecewise bicubic polynomial over a rectangular mesh. Their results show that the fast contouring algorithms will be better than the MATLAB/DEI approach if the refining factor (the ratio of the fine mesh to the coarse mesh) is relatively large. In chapter 4, we improve their fast contouring algorithms and extend them to unstructured triangular meshes. Furthermore, we will present some ideas to improve the fast algorithms on arbitrary problems and meshes.

Chapter 3

Scattered Data Interpolation

In this chapter we will first define the scattered data interpolation problem and then we will introduce an approach based on the use of DEI to efficiently solve the problem. Furthermore, we will discuss other similar approaches and compare the efficiency of our approach. Finally, we will investigate an application arising in the numerical solution of parabolic PDEs.

3.1 Problem Definition

In this chapter, we focus on a two-dimensional second-order elliptic PDE of the form

$$Lu = g(x, y, u, u_x, u_y), \tag{3.1}$$

where L is a given differential operator of the form

$$L = a_1(x, y)\frac{\partial^2}{\partial x^2} + a_2(x, y)\frac{\partial^2}{\partial y^2} + a_3(x, y)\frac{\partial^2}{\partial x \partial y}. \tag{3.2}$$

We assume that there are some accurate numerical results (approximate function values as well as approximate derivative values with respect to x and y) at some mesh points that are not necessary structured. Although this discrete data may have high order accuracy, they might not generate a smooth and accurate surface plot or contour plot due to an insufficient number of mesh points or an inappropriate distribution of these points. In scientific visualization, where one might want to zoom in on a special region, this deficiency may be more obvious.

Ramos and Enright [4] introduced the ADEI, an interpolant over scattered data based on the DEI but with an improved smoothness property. Although the ADEI approach generates an interpolant that accurately approximates the solution at off-mesh points, it has two main difficulties:

- The iteration to find an improved choice for coefficients sometimes takes a long time.

- Even with spending a long time for this iteration, it does not necessary generate continuous approximations along the boundaries of the mesh elements.

In the next section, we will investigate the use of a pure piecewise bivariate cubic interpolant introduced by Enright [2]. Moreover, we will present some other approaches and make a comparison among the most promising alternatives.

3.2 Pure Cubic Interpolant (PCI)

The pure bivariate interpolant restricts the approximating polynomial to be a bivariate polynomial of total degree d. For each triangular element e, the interpolant, $p_{d,e}(x, y)$, is defined by

$$p_{d,e}(x, y) = \sum_{i=0}^{d} \sum_{j=0}^{d-i} c_{ij} s^i t^j, \tag{3.3}$$

where

$$s = \frac{(x - x_1)}{D_1}, t = \frac{(y - y_1)}{D_2}, \tag{3.4}$$

and, as can be seen in figure 3.1, D_1 and D_2 are related to the size of the triangle e, in the x and y directions, and (x_1, y_1) is the lower left corner of the rectangle defined by the triangle e. For a pure interpolant of degree d, there are $\frac{(d+1)(d+2)}{2}$ unknown coefficients to be determined.

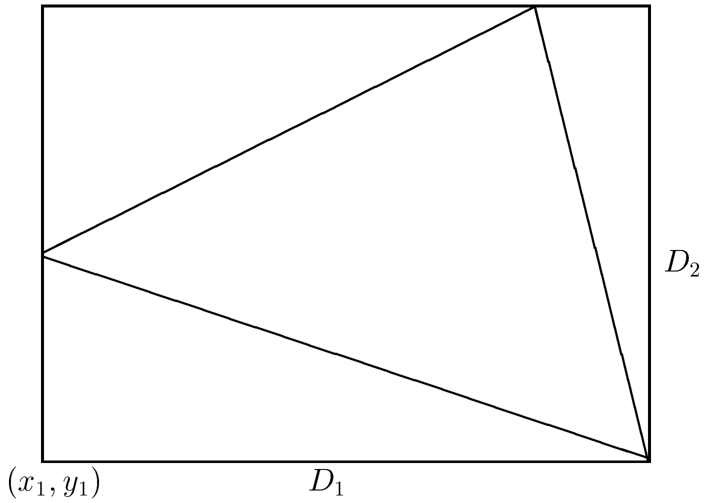


Figure 3.1: A triangular element.

For the class of problems we are considering, fourth order accuracy seems to be standard and we have therefore chosen piecewise bicubic ($d = 3$) for our interpolant. In this case, we can represent the unknown coefficients by a matrix C of the form

$$C = \begin{pmatrix} 0 & 0 & 0 & c_{30} \\ 0 & 0 & c_{21} & c_{20} \\ 0 & c_{12} & c_{11} & c_{10} \\ c_{03} & c_{02} & c_{01} & c_{00} \end{pmatrix},$$

and therefore the cubic polynomial is

$$p_{3,e}(x,y) = SCT = \begin{pmatrix} s^3 & s^2 & s & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & c_{30} \\ 0 & 0 & c_{21} & c_{20} \\ 0 & c_{12} & c_{11} & c_{10} \\ c_{03} & c_{02} & c_{01} & c_{00} \end{pmatrix} \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix}.$$

We call this interpolant the Pure Cubic Interpolant (PCI). For the PCI, we have 10 unknown coefficients. For each of the three mesh points of a triangle we know approximations to the function, U, and the values of the derivatives, U_x and U_y. This results in

9 linear equations, each of which is in one of the following forms:

$$\sum_{i=0}^{3}\sum_{j=0}^{3-i} c_{ij}\overline{s}^{i}\overline{t}^{j} = U, \tag{3.5}$$

$$\sum_{i=0}^{3}\sum_{j=0}^{3-i} i c_{ij}\overline{s}^{i-1}\overline{t}^{j} = D_1 U_x, \tag{3.6}$$

$$\sum_{i=0}^{3}\sum_{j=0}^{3-i} j c_{ij}\overline{s}^{i}\overline{t}^{j-1} = D_2 U_y, \tag{3.7}$$

where $\overline{s} = \frac{x-x_1}{D_1}$ and $\overline{t} = \frac{y-y_1}{D_2}$. We still require another independent linear constraint to uniquely determine the 10 coefficients of $p_{3,e}(x,y)$. This can be accomplished by choosing one collocation point, (x_m, y_m), inside of the triangle e and imposing a condition that the interpolant almost satisfies the underlying PDE at that point. From 3.3 we have

$$\frac{\partial^2 p_{3,e}(x,y)}{\partial x^2} = \frac{1}{D_1^2}\sum_{i=0}^{3}\sum_{j=0}^{3-i} i(i-1)c_{ij}\Big(\frac{x-x_0}{D_1}\Big)^{i-2}\Big(\frac{y-y_0}{D_2}\Big)^{j}, \tag{3.8}$$

$$\frac{\partial^2 p_{3,e}(x,y)}{\partial y^2} = \frac{1}{D_2^2}\sum_{i=0}^{3}\sum_{j=0}^{3-i} j(j-1)c_{ij}\Big(\frac{x-x_0}{D_1}\Big)^{i}\Big(\frac{y-y_0}{D_2}\Big)^{j-2}, \tag{3.9}$$

$$\frac{\partial^2 p_{3,e}(x,y)}{\partial x \partial y} = \frac{1}{D_1 D_2}\sum_{i=0}^{3}\sum_{j=0}^{3-i} ij c_{ij}\Big(\frac{x-x_0}{D_1}\Big)^{i-1}\Big(\frac{y-y_0}{D_2}\Big)^{j-1}, \tag{3.10}$$

and therefore $Lp_{3,e}(x_m, y_m)$ is a linear combination of the unknown coefficients and from 3.1 for the point (x_m, y_m) we can impose the linear constraint

$$Lp_{3,e}(x_m, y_m) = g(x_m, y_m, \hat{U}_m, \frac{\partial \hat{U}_m}{\partial x}, \frac{\partial \hat{U}_m}{\partial y}), \tag{3.11}$$

where $\hat{U}_m$, $\frac{\partial \hat{U}_m}{\partial x}$ and $\frac{\partial \hat{U}_m}{\partial y}$ are linear approximations to $U(x_m, y_m)$, $U_x(x_m, y_m)$ and $U_y(x_m, y_m)$, respectively. Since we need only one collocation point, the obvious choice is the middle point of the triangle and we have done this. To determine the unknown coefficients, we then solve the system of linear equations that is formed by 9 linear constraints 3.5, 3.6 and 3.7 and the other linear constraint 3.11. This system can be written,

$$W\overline{c} = b \tag{3.12}$$

where W is a matrix that depends on the mesh points, the collocation point, and the definition of L; while $\overline{c}$ is the vector of unknown coefficients; and b is a vector that depends on the mesh data and the linear approximations associated with the collocation point.

One of the advantages of the PCI is its continuity along the boundaries of mesh elements. The standard DEI finds an accurate interpolation over each element but might suffer from discontinuity along the boundaries of the mesh elements. This will not be the case for the PCI, which always produces continuous results. This follows since the associated pure cubic polynomials for two neighbor triangles produce the same value for each arbitrary point on the shared side. For each of the two mesh points on the shared side, there are three pieces of information (U, U_x and U_y). If the side is a horizontal or vertical line, either U_x or U_y expresses the derivative of the function along that side. In general, where the side is an arbitrary line, we are able to state the derivative of the function in the direction of the side, U_S, in terms of U_x and U_y by

$$U_S = \frac{\partial U}{\partial x} \cdot x_S + \frac{\partial U}{\partial y} \cdot y_S. \tag{3.13}$$

Therefore, for each mesh point we know the value of the function and the value of the derivative in the direction of the sides and all this information is used to define the polynomial. Since the PCI is of degree 3, that means it has four unknown coefficients, and for the shared side we know exactly four pieces of information, the associated polynomials for both triangles produce the same value for each point located on the shared side. In this way, we have satisfied continuity, one of the most important aspects with regard to visualization.

In addition to continuity, the PCI is efficient in terms of time and accuracy. Since the linear system that must be solved has only 10 linear independent equations, the PCI needs less time than the DEI of degree 3, which has 16 linear independent equations, and much less time than the ADEI that may have to iterate many times to choose the best location for the collocation points. In chapter 5, we will compare time and accuracy of the PCI with those of the DEI and the ADEI and also some other approaches that will be introduced in the next section.

3.3 Other Approaches

Although the PCI generally produces excellent results in terms of time, accuracy and continuity, we are not sure if it is the best interpolant based on using a DEI for our problem. In order to obtain such confidence, we will investigate some other approaches based on using a DEI. Since the underlying PDE we are considering is symmetric in the x and y directions, intuitively we consider only symmetric approaches. We call these approaches DEIm_n where m is the degree of the corresponding interpolant and n is the number of the unknown coefficients. The first two approaches, DEI3-15 and DEI3-13, are based on the DEI of degree 3 but total degree 5 and 4 respectively (the standard DEI is basically of total degree 6) and the last one, DEI4-15, is a pure quartic interpolant of total degree 4. C_1, C_2 and C_3 are the matrices of unknown coefficients for DEI3-15, DEI3-13 and DEI4-15 respectively and are represented by

$$C_1 = \begin{pmatrix} 0 & c_{32} & c_{31} & c_{30} \\ c_{23} & c_{22} & c_{21} & c_{20} \\ c_{13} & c_{12} & c_{11} & c_{10} \\ c_{03} & c_{02} & c_{01} & c_{00} \end{pmatrix},$$

$$C_2 = \begin{pmatrix} 0 & 0 & c_{31} & c_{30} \\ 0 & c_{22} & c_{21} & c_{20} \\ c_{13} & c_{12} & c_{11} & c_{10} \\ c_{03} & c_{02} & c_{01} & c_{00} \end{pmatrix},$$

$$C_3 = \begin{pmatrix} 0 & 0 & 0 & 0 & c_{40} \\ 0 & 0 & 0 & c_{31} & c_{30} \\ 0 & 0 & c_{22} & c_{21} & c_{20} \\ 0 & c_{13} & c_{12} & c_{11} & c_{10} \\ c_{04} & c_{03} & c_{02} & c_{01} & c_{00} \end{pmatrix}.$$

The results of the comparison between the PCI approach and these approaches will be presented in chapter 5. As we will observe, the PCI is the best approach in both time and accuracy. It is also the only approach that always generates continuous results. We will use the interpolant generated by this approach for implementing our fast contouring algorithms.

3.4 The DEI and Parabolic PDEs

Up to now, we have assumed that the underlying PDE is an elliptic PDE and accurate approximations to U, U_x and U_y are given for each mesh point of an irregular unstructured mesh. In this section, we consider another type of PDE, a parabolic PDE, of the form

$$Lu = g(x, y, u, u_x), \tag{3.14}$$

where L is a given differential operator of the form

$$L = a_1(x, y)\frac{\partial^2}{\partial x^2} + a_2(x, y)\frac{\partial}{\partial y}. \tag{3.15}$$

For this problem, we assume that approximations to U and U_x are given. Consequently for a triangular mesh, we have 6 pieces of information for each triangle. Considering the PCI, we need to choose 4 collocation points to uniquely determine the coefficients of the polynomial associated with the triangle. One method for choosing such extra points is to apply the idea introduced by Ramos and Enright in [4], making random choices and testing the corresponding norm iteratively in order to obtain the best collocation points. However we will consider another method because of the required time and inefficiency of this random approach. An obvious method is to find some fixed points that work relatively well for most situations (different problems and different meshes). This method was used to choose the middle point of a triangle as the only collocation point needed in the previous section. We need 4 collocation points and an obvious choice is the middle point of a triangle as well as three middle points of each side of the triangle. We call this 1-1-0 situation since the position of the middle point of each side depends only on two mesh points. In general, in a l-m-n situation l, m and n are the ratios of the influence of each of mesh points on the position of a collocation point. Unfortunately, the 1-1-0 situation yields a singular matrix W for some triangles. Another obvious choice is the middle point of a triangle as well as three mesh points (1-0-0 situation). In addition to this, we consider other 'natural' choices for the set of collocation points. Figure 3.2 shows all situations we have considered.

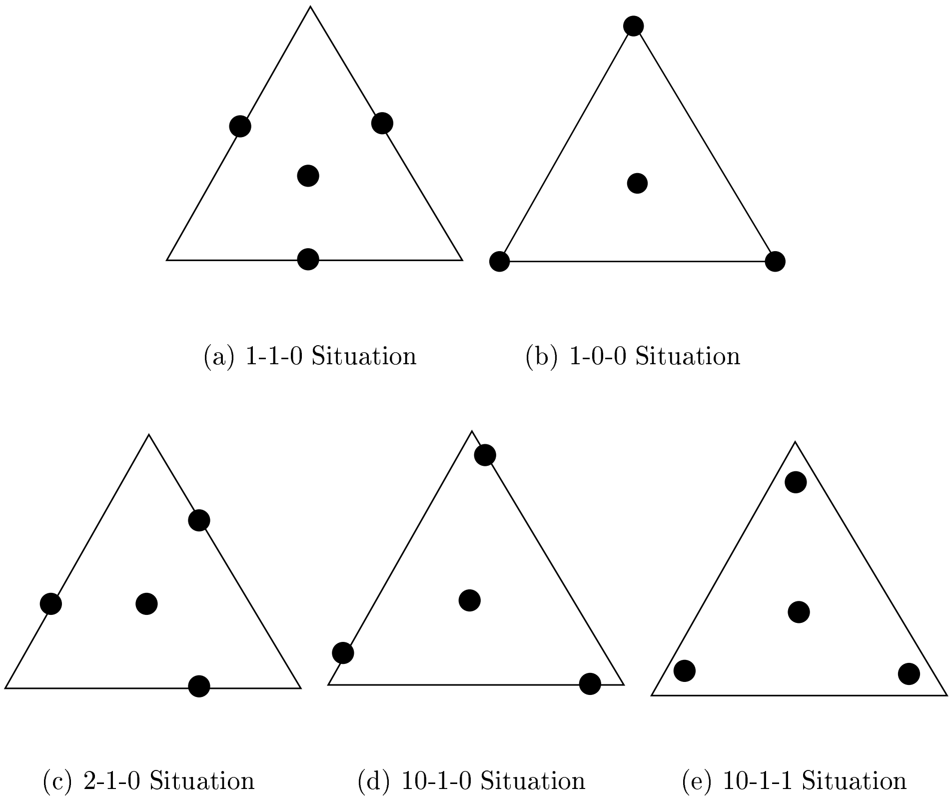
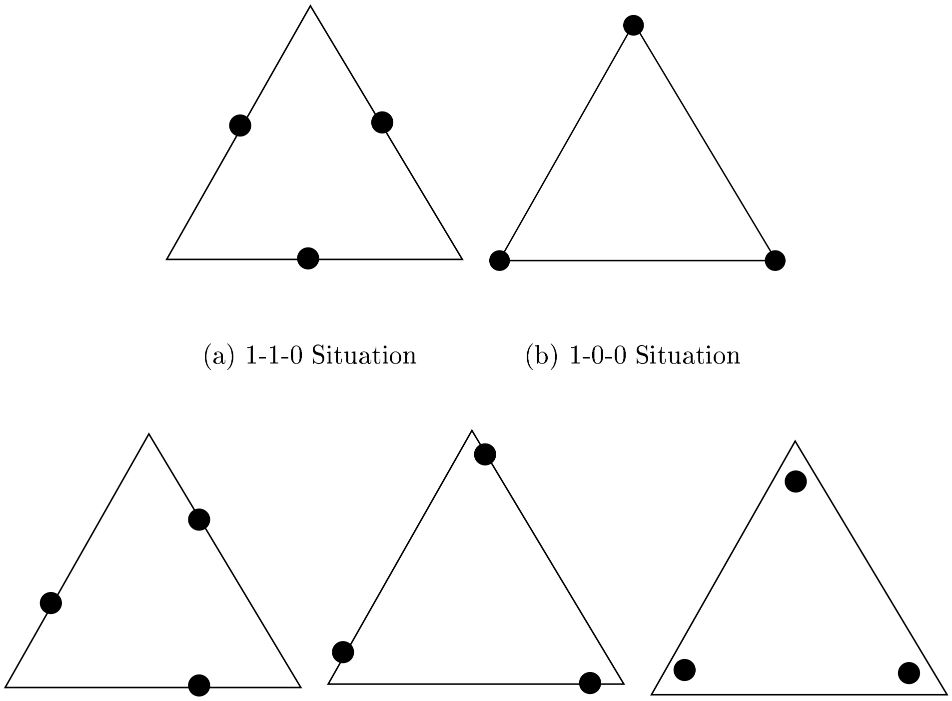


Figure 3.2: Five situations for selecting the four collocation points.

Since the underlying PDE is not symmetric and includes different order of derivatives for the x and y directions, it is likely that a non-symmetric matrix of coefficients will lead to better results than a symmetric one. We investigate this by introducing a new



(a) 1-1-0 Situation (b) 1-0-0 Situation

(c) 2-1-0 Situation (d) 10-1-0 Situation (e) 10-1-1 Situation

Figure 3.3: Five situations for selecting the three collocation points.

interpolant, $\overline{p}_{d,e}(x, y)$, represented by

$$\overline{p}_{d,e}(x, y) = \sum_{i=0}^{2} \sum_{j=0}^{3-i} c_{ij} s^i t^j \tag{3.16}$$

with the coefficient matrix, $\overline{C}$, of the form,

$$\overline{C} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & c_{21} & c_{20} \\ 0 & c_{12} & c_{11} & c_{10} \\ c_{03} & c_{02} & c_{01} & c_{00} \end{pmatrix}$$

Matrix $\overline{C}$ is the same as matrix C (for the PCI) but with $C_{30} = 0$. Due to 9 unknown coefficients, we only need 3 collocation points. Figure 3.3 shows some situations for selecting these three collocation points.

In chapter 5 we will see that non-symmetric matrix $\overline{C}$ produces relatively better results than symmetric matrix C. Both approaches can produce discontinuous results along the triangles' sides, so finding a continuous approach for this problem can be an open problem for future work.

Chapter 4

Fast Contouring Algorithms

In [1], Bradbury and Enright investigated how one can compute and plot contour lines directly from the piecewise polynomial of the DEI. They considered the contouring problem on a rectangular mesh and introduced three fast algorithms for computing contour lines. In this chapter, we will introduce three fast contouring algorithms for a triangular mesh based on the approaches introduced by Bradbury and Enright.

Each algorithm has three stages with the same first two stages for all three algorithms. These three stages are as follows:

1. Computing the minimum and maximum values of the interpolant for each triangle. There are at least three advantages in computing the minimum and maximum values. First of all, it simplifies subsequent intersection tests for all triangles to know if it contains the contour line as we can determine this by simply comparing the contour level with the minimum and maximum values. Second, the situation that a contour curve lies completely inside of a triangle can be easily distinguished if the minimum and maximum values are known. For this situation there is no intersection between the contour curve and the triangle's sides. Third, global minimum and maximum values can be computed from local information and if the contour levels are not provided by the user, suitable default values can be determined dynamically

based on this global information.

2. Identifying the intersection points between contour lines and triangle boundaries and recursively dividing the triangles containing more or less than two such intersections into several triangles such that all triangles have two or zero intersections. A situation where a triangle has exactly two intersections with a contour line is called a *desired* situation.

3. Computing the accurate contour line between the two intersection points for each triangle. We will introduce three algorithms for this stage.

The interpolant used for computing contour lines is the PCI that produces continuous contour plots. In the following sections, we will present the three stages of our fast contouring algorithms in more detail.

4.1 Stage One: Computing the Minimum and Maximum Values

Finding the minimum and maximum values of our interpolant over each triangle can be the most expensive stage if we try to identify them to full machine accuracy [1]. The most accurate method is to find the points inside of triangle e such that $p_{e_x}(x, y)$ and $p_{e_y}(x, y)$ are simultaneously zero but this can be very expensive in terms of time. A fast method is to assume that the extrema values occur at triangles' sides. This method is relatively fast (it includes only finding zeros of three polynomial for each triangle) and much better than MATLAB contouring algorithm that assumes the extrema values occur at the mesh points [3]. We can also improve this method to obtain better results by a simple idea. The idea is a recursive approach such that the user can specify the number of recursive steps. The recursive algorithm is presented in figure 4.1.

Figure 4.1: The recursive algorithm to approximate the extrema values

for each triangle e of the mesh
  lMin(e) ← recFindMin(e,1)
  lMax(e) ← recFindMax(e,1)
end for
gMin ← min(lMin)
gMax ← max(lMax)

function recFindMin(e,level)
  recFlag ← False
  for each of three side of triangle e (i=1,2,3)
    find the extrema of $p_e(x, y)$ along the ith side
    minPoint(i) ← select the minimum point from the extrema points and two end points
    if minpoint(i) is selected from the extrema points
      recFlag ← True
    end if
  end for
  if level < maxLevel **and** recFlag=True
    minValue ← recFindMin(triangle(minPoint),level+1)
  else
    minValue ← the minimum value of $p_e(x, y)$ in minPoint
  end if
  return minValue
end function

function recFindMax(e,level)
⋮

In addition, figure 4.2 shows how the algorithm finds the minimum (or maximum) values of the interpolant in a triangle through a two-level recursion. The cost of this algorithm is a multiple ($\leq 2*$ number of recursive steps) of the cost of the primary algorithm (looking at the triangle's sides only).

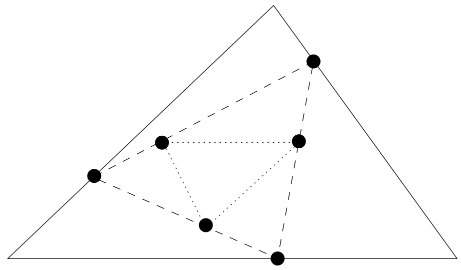


Figure 4.2: Finding the minimum (or maximum) values of the interpolant.

4.2 Stage Two: Identifying Intersection Points and Dividing Triangles

After computing the minimum and maximum values of each triangle, we can determine whether a given contour line intersects with a triangle by simply comparing the contour level with the minimum and maximum values. The user can specify either the contour levels explicitly or only the desired number of contour levels. In the latter situation, the contour levels are specified by equally dividing the range between global minimum and maximum values. Then, for each triangle e, if the contour level is between the minimum and maximum values over e, an intersection test is performed for each of three sides of triangle e. The result of this test can be then classified in terms of total number of intersections for each triangle as follows:

- *No intersection*: the contour line completely lies inside of the triangle.

- *One intersection*: the contour line is either an inner or outer tangent to one side.
- *Two intersections*: a single contour line passes through the triangle.
- *Three intersections*: a single contour line passes through the triangle and it is also tangent to one side.
- *Four intersections*: two contour lines pass through the triangle.
- *More than four intersections*: more than two contour lines pass through the triangle.

For all cases except *'two intersections'*, the triangle should be divided into two or more triangles such that each new triangle has exactly two intersections with the contour level (or zero if a triangle contains no contour line). In other words, in the final situation, each triangle has zero or two intersections with the contour line. We implement this technique using a recursive function that takes a triangle and the contour level and other necessary parameters and then compute the contour line. For each case, an appropriate approach must be developed. We classify these approaches into four categories as follows:

1. **No intersection:** The contour curve will completely lie inside a triangle. The solution is to find a line between a vertex of the triangle and its corresponding opposite side such that it intersects the contour curve in two points. This is easy to do since the location of the maximum and minimum values of the interpolant on this triangle is known. Figure 4.3 shows the situation and the solution. We will then have two triangles in a desired situation (each has exactly two intersections with the contour curve) and we will call the recursive function for each new triangle, separately.

2. **One intersection:** Figure 4.4 illustrates the situations that can result in one intersection corresponding to an inner tangent and an outer tangent. There should

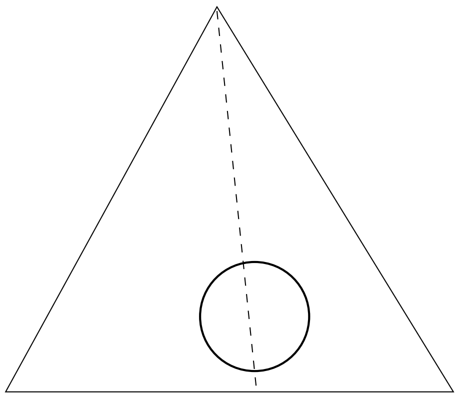
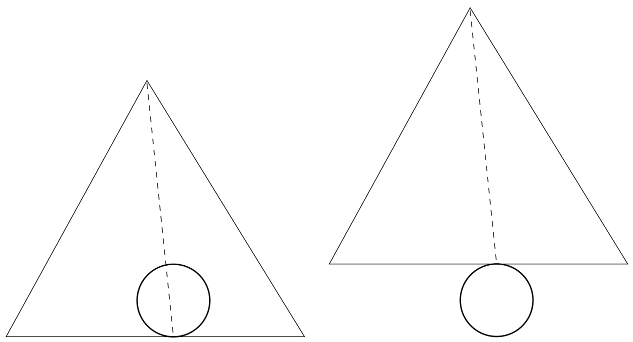


Figure 4.3: No intersection: Two new triangles.

be considered two different treatments for these two situations. In order to distinguish between the inner tangent and outer tangent situations, we consider the line between the intersection point and the corresponding opposite vertex. Then, we compute the number of intersections between this line and the contour curve. In case that there is only one intersection (outer tangent), we will do nothing and the contour curve will be considered by the neighbor triangle. In the other case, when there are two intersection points (inner tangent), we divide the triangle into two new triangles and then call the recursive function for each of them separately. Each new triangle will have exactly two intersections with the contour curve and consequently we have a desired situation for each of the new triangles.

3. **Four intersections:** There are several situations that can arise when there are four intersections with the contour curve. We classify these situations into four categories according to the position of the intersection points as follows:

   (a) Two intersections on one side and one intersection on each of the other sides (2-1-1 situation): Figure 4.5 shows two such situations. We apply the following approach to compute contour lines in these situations:



(a) Inner tangent (b) Outer tangent

Figure 4.4: One intersection: Two situations, inner tangent and outer tangent.

Find the middle point of two intersections that are in the same side. Divide the triangle into two triangles by adding a line between this point and the corresponding opposite vertex and then call the recursive function for each of two new triangles separately.

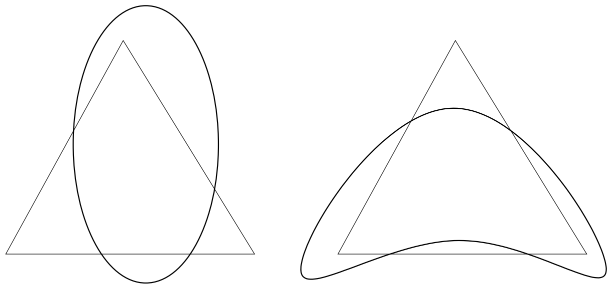


Figure 4.5: Four intersections: 2-1-1 situations.

Figure 4.6 shows the final triangles created by applying this approach. It works properly for the first situation and results in two triangles that are in a desired situation. For the second situation, it creates two triangles, each of which has four intersections with the contour curve. It can be seen from the figure that both of these new triangles are of the first type and the contour lines will be computed by applying the approach one more time.

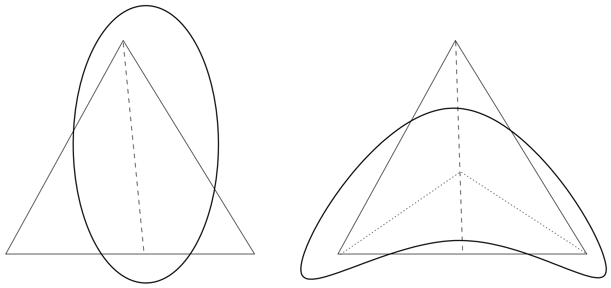


Figure 4.6: Four intersections: 2-1-1 situations (Final triangulation).

(b) Two intersections on two sides and no intersection on the other side (2-2-0 situations): Figure 4.7 illustrates two such situations. We apply the following approach to compute contour lines in these situations:

> *Find the middle points of the intersections that are on the same side. Draw a line between these two points and also draw a line between one of the middle points and the corresponding opposite vertex and then call the recursive function for each of three new triangles separately.*

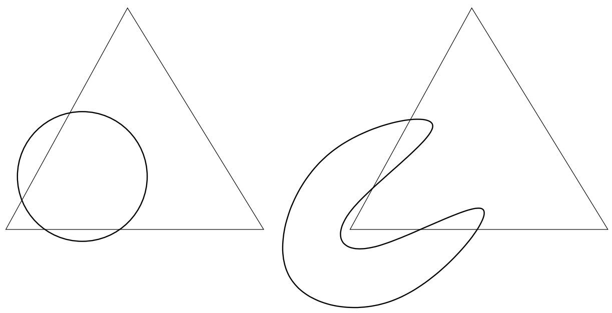


Figure 4.7: Four intersections: 2-2-0 situations.

In the first situation, each of three triangles has exactly two intersections with the contour curve. In the second situation, one triangle has exactly two intersections with the contour curve and the other triangles have four intersections with contour curve (2-1-1 situation) and obviously we need to apply the recursive function one more time. Figure 4.8 shows the final triangles created by applying this approach recursively.

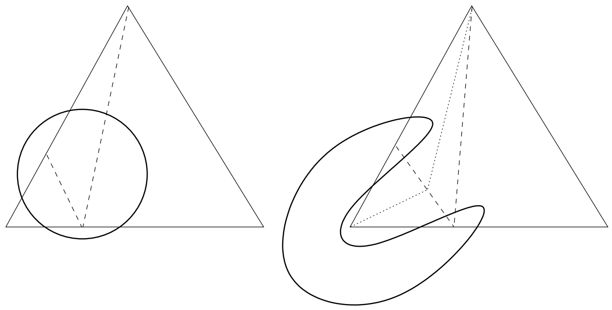


Figure 4.8: Four intersections: 2-2-0 situations (Final triangulation).

(c) Three intersections on one side, one intersection on a second side and no intersection on the last side (3-1-0 situations): Figure 4.9 shows two such situations. We apply the following approach to compute contour lines in these situations:

Find the middle point of three intersections that are on the same side. Draw a line between the middle point and the corresponding opposite vertex and then call the recursive function for each of two triangles separately.

Although this approach does not directly convert the first case of figure 4.9 into a desired situation, it converts a 3-1-0 situation into 2-1-1 and 2-2-0 situation, both of which have been discussed before. This approach converts the first situation into two new triangles whose category depends on the position of the middle point. Figure 4.10 shows the triangles that can be created by the first situation.

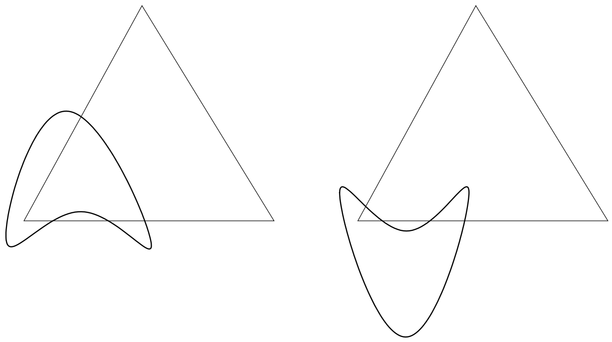


Figure 4.9: Four intersections: 3-1-0 situations.

In the first case of figure 4.10, the triangle is divided into two new triangles, one with exactly two intersections and the other with four intersections of type 2-1-1 that can be converted to a desired situation by calling the recursive function two more times (as discussed before). Therefore, the first case is converted into a desired situation in at most three recursive steps. For the second case of figure 4.10, the triangle is divided into two new triangles, both with four intersections with the contour curve, one of type 2-1-1 and the other of type 2-2-0. Since both new triangles can be obviously converted into a desired situation in one step, the second case can be converted into a desired situation in two recursive steps.

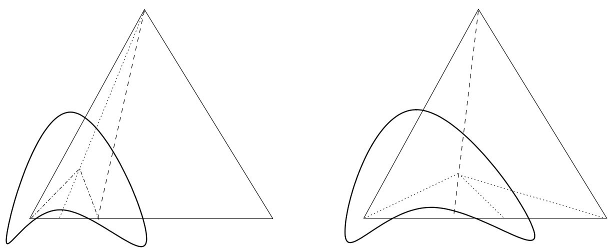


Figure 4.10: Four intersections: the first 3-1-0 situation (Final triangulation).

Figure 4.11 illustrates two new cases that can be created by the second situation. In the first case, the triangle is divided into two new triangles, both in a desired situation. In the second case, the triangle is divided into two new triangles, one with four intersections of type 2-1-1 and the other with two intersections. Therefore, the second situation is also converted into a desired situation in at most two recursive steps.

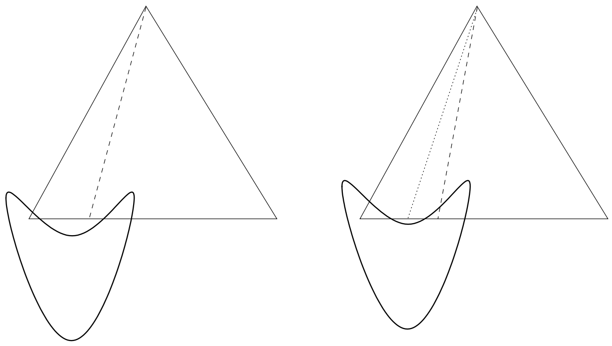


Figure 4.11: Four intersections: the second 3-1-0 situation (Final triangulation).

(d) Four intersections on one side and no intersection on the other two sides (4-0-0 situations): Figure 4.12 shows the two such situations that can arise. We apply the following approach to compute contour lines in these situations:

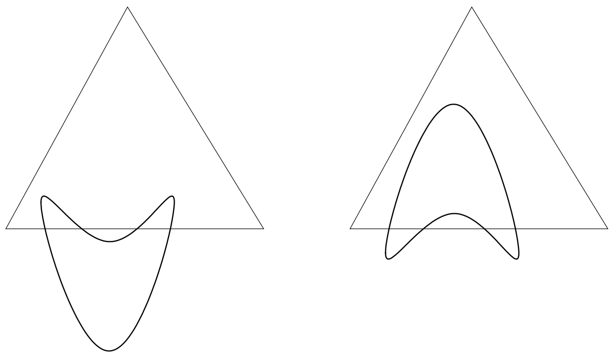


Figure 4.12: Four intersections: 4-0-0 situations.

Find the middle point of the two middle intersection points. Draw a line between the middle point and the corresponding opposite vertex and then call the recursive function for each of two new triangles separately.

In the first situation, each of two triangles has exactly two intersections with the contour curve. In the second situation, each of two triangles has four intersections of type 2-2-0 and obviously we need to apply the recursive step one more time. Figure 4.13 shows the final triangles created by applying this approach recursively.

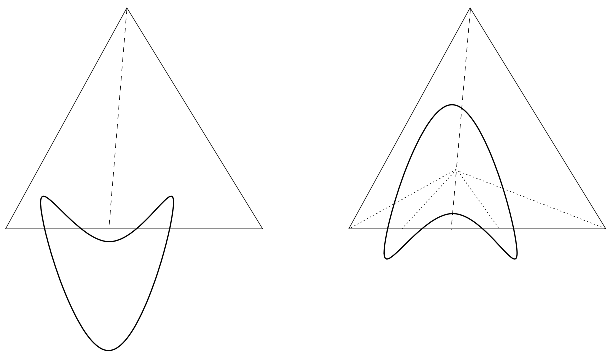


Figure 4.13: Four intersections: 4-0-0 situations (Final triangulation).

4. **More than four intersections:** In the case that the triangulation is relatively coarse, we might face some triangles that have more than four intersections with the contour curve. We propose a simple approach in order to convert these situations into other situations introduced before (especially two and four intersections).

    Find the middle point of the triangle. Draw three lines from this point to all three vertices of the triangle and then call the recursive function for each of three new triangles separately.

Figure 4.14 shows such a situation, a triangle with six intersections with the contour curve, and the final triangles. As can be seen, for this situation our approach converts the triangle into three new triangles such that each of them has four intersections of type 2-1-1.

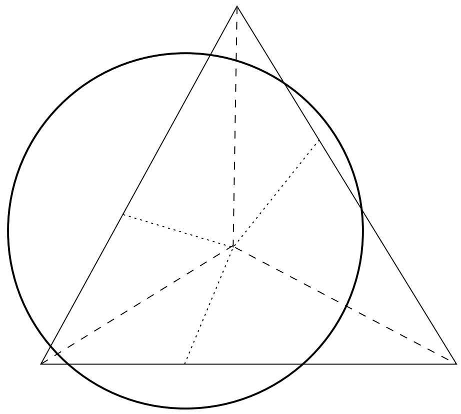


Figure 4.14: More than four intersections.

This classification seems to be appropriate for all the cases that can arise in contours. Practical results confirm our assumption and show that all the cases that can arise have been considered.

4.3 Stage Three: Computing the Accurate Contour Lines

The last stage of our fast contouring algorithms is to compute the contour line between two points located on a triangle's sides. In this section, at first we introduce three approaches based on the approaches implemented by Bradbury and Enright in [1] and then we try to improve each approach separately.

4.3.1 The Intercept Method

The basic idea of the Intercept method is to refine each element in only one direction, the x or y direction, depending on the position of the intersection points. We then find intersections between the new refined lines and the contour line and finally simply connect the intersection points [1]. Since each refined line is horizontal or vertical (i.e. parallel to the x or y axis), $p_e(x, y)$ reduces to a univariate cubic polynomial (along the refined line). For a triangular mesh, since we employ the PCI, the following approach can be applied

> *Connect the two intersection points by a line and then consider some regular spaced points on this line. Draw perpendicular lines to this line at the considered points and then find the intersections between these perpendicular lines and the contour curve and connect the intersection points.*

Figure 4.15 illustrates how this approach approximates the contour line with a refinement of eight.

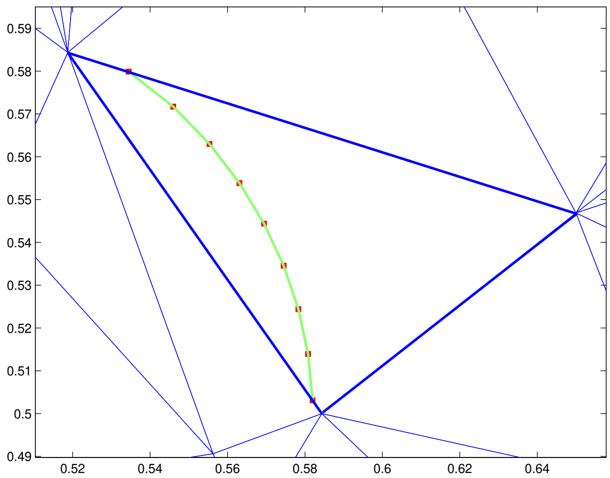


Figure 4.15: The contour curve (in green) created by the basic Intercept method. The points (in red) on the contour curve are the intersections between the contour curve and refined line.

The interpolant used for this problem, $p_e(x,y)$, is the Pure Cubic Interpolant (PCI) of the form

$$p_e(x,y) = SCT = \begin{pmatrix} s^3 & s^2 & s & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & c_{30} \\ 0 & 0 & c_{21} & c_{20} \\ 0 & c_{12} & c_{11} & c_{10} \\ c_{03} & c_{02} & c_{01} & c_{00} \end{pmatrix} \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix}.$$

Although the refined lines are not necessary horizontal or vertical (i.e., parallel to the x or y axis), the polynomial for each refined line will be a cubic univariate since the PCI is of total degree three. For each arbitrary line we have $s = at + b$ and if we replace s with $at + b$, we will have

$$p_e(x,y) = \begin{pmatrix} a^3 & 3a^2b & a^2 & 3ab^2 & 2ab & a & b^3 & b^2 & b & 1 \end{pmatrix} \begin{pmatrix} c_{30} & 0 & 0 & 0 \\ 0 & c_{30} & 0 & 0 \\ c_{21} & c_{20} & 0 & 0 \\ 0 & 0 & c_{30} & 0 \\ 0 & c_{21} & c_{20} & 0 \\ c_{12} & c_{11} & c_{10} & 0 \\ 0 & 0 & 0 & c_{30} \\ 0 & 0 & c_{21} & c_{20} \\ 0 & c_{12} & c_{11} & c_{10} \\ c_{03} & c_{02} & c_{01} & c_{00} \end{pmatrix} \begin{pmatrix} t^3 \\ t^2 \\ t \\ 1 \end{pmatrix}, \tag{4.1}$$

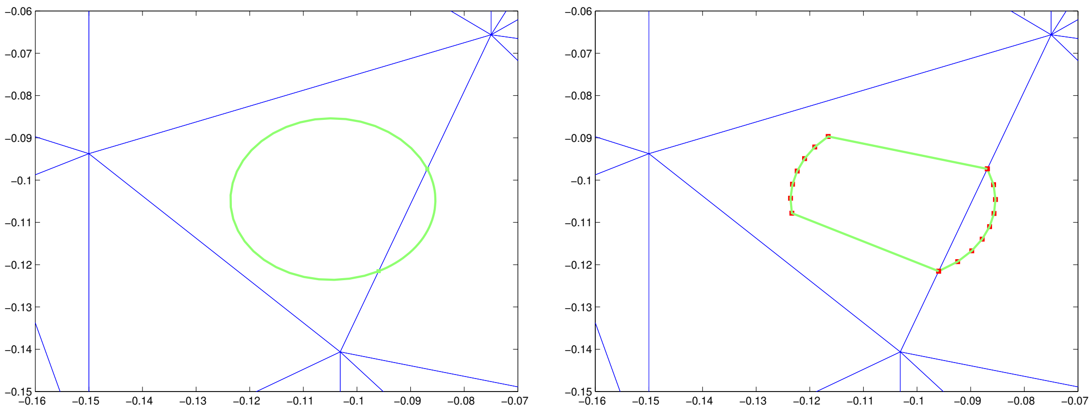
and $p_e(x,y)$ will be a cubic univariate polynomial. Then for determining the intersections between a refined line and the contour curve with contour level v, we should find the roots of equation 4.2

$$k_3t^3 + k_2t^2 + k_1t + k_0 = v, \tag{4.2}$$

where

$$
\begin{pmatrix} k_3 & k_2 & k_1 & k_0 \end{pmatrix} = \begin{pmatrix} a^3 & 3a^2b & a^2 & 3ab^2 & 2ab & a & b^3 & b^2 & b & 1 \end{pmatrix} \begin{pmatrix} c_{30} & 0 & 0 & 0 \\ 0 & c_{30} & 0 & 0 \\ c_{21} & c_{20} & 0 & 0 \\ 0 & 0 & c_{30} & 0 \\ 0 & c_{21} & c_{20} & 0 \\ c_{12} & c_{11} & c_{10} & 0 \\ 0 & 0 & 0 & c_{30} \\ 0 & 0 & c_{21} & c_{20} \\ 0 & c_{12} & c_{11} & c_{10} \\ c_{03} & c_{02} & c_{01} & c_{00} \end{pmatrix} . \tag{4.3}
$$

This approach works properly as long as there is only one intersection between each refined line and the contour curve. Bradbury and Enright addressed this difficulty and showed a simple example where the whole contour curve lies inside of one element [1]. We will not encounter these situation because in our algorithm there will be exactly two intersections between the contour curve and a triangle containing a part of the contours. We might still have the situation that there is more than one intersection between a refined line and the contour curve. Note that the PCI can find at most three intersections. Figure 4.16(a) shows a simple situation and figure 4.16(b) shows the result of our basic approach. As can be seen, we lose some sections of the contour curve due to the fact that the refinement is performed only between two intersections. On the other hand, refining the area outside of two intersections does not help because there is more than one intersection and it is not obvious how the intersection points should be connected. In order to surmount this difficulty, we introduce a recursive approach that attempts to find some equally spaced points located along the contour curve.



(a) True contour curve.

(b) The contour curve plotted by the basic Intercept method.

Figure 4.16: An example of the basic Intercept method's disadvantage.

The algorithm presented in figure 4.17 shows an overview of this approach. The idea is to first find the middle point of the contour curve between the initial two interception points and consider this point as one of the refined points (known to be on the contour curve) and then call the recursive function for each of two new intervals (the intervals between the new point and each of the initial points). If n is the number of recursive levels, finally we will have $2^n + 1$ points (including the two initial points).

Figure 4.18 illustrates how this approach draws contour curves by applying a three-level recursion. The most important advantage of this approach is to locate almost equally spaced points on the contour curve. However, there is still a remaining difficulty. As can be seen in figure 4.18, although the points are located equally spaced inside of each triangle, they are not totally equally spaced if we consider all triangles. In other words, the number of extra points will not be equal for all triangles. For example in figure 4.18, the number of extra points (7 points) is not enough for the left triangle but more than enough for the right triangle. In order to overcome this problem, instead of having a fixed level of recursion for all triangles, we consider a threshold for the distance between the extra points and try to locate the extra points such that the distance between

Figure 4.17: The Recursive Algorithm to Find middle points

function recAddMiddlePoint(p_1,p_2,cnt,$\cdots$)

$p_m \leftarrow$ the middle point of p_1 and p_2

$m_1 \leftarrow$ the slope of the tangent line of the contour curve at p_1

$m_2 \leftarrow$ the slope of the tangent line of the contour curve at p_2

$m \leftarrow \frac{m_1+m_2}{2}$

$p \leftarrow$ the intersection between the contour curve and the line passing p_m with a slope m

if cnt > 0

$p_L \leftarrow$ recAddMiddlePoint(p_1,p,cnt-1,$\cdots$)

$p_R \leftarrow$ recAddMiddlePoint(p,p_2,cnt-1,$\cdots$)

return $p_L + p + p_R$

else

return p

end if

end function

each two points is less than the desired threshold. Figure 4.19 illustrates the effect of considering a threshold instead of the same level of recursion for all triangles. The value of the threshold can be specified by the user.

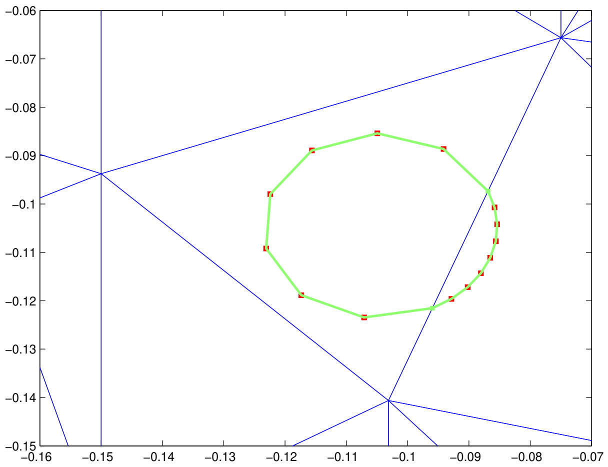


Figure 4.18: The contour curve created by the improved Intercept method using a fixed number of recursions to control the spacing.

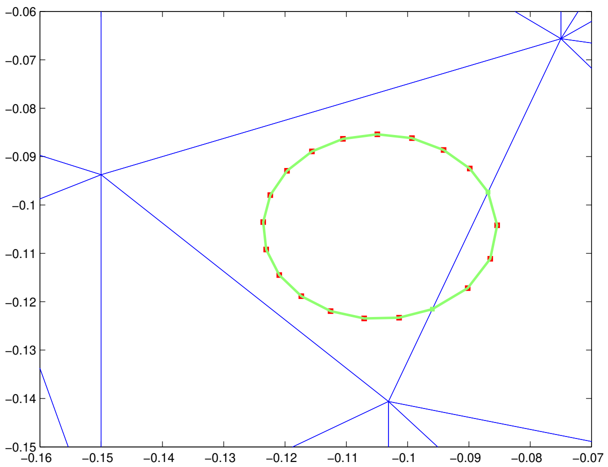


Figure 4.19: The contour curve created by the final Intercept method using a threshold value to control the spacing.

4.3.2 the Simple ODE Method (SODE)

The contouring problem can be converted to an initial value problem [1]. The problem for the contour level v is to solve

$$u(x, y(x)) = v, \tag{4.4}$$

where y is a function of x. Then if we differentiate u with respect to x, we have

$$u_x(x, y(x)) + u_y(x, y(x)) \cdot y_x(x) = 0, \tag{4.5}$$

or

$$y_x(x) = -\frac{u_x(x, y(x))}{u_y(x, y(x))}. \tag{4.6}$$

If we consider x as a function of y, we similarly have

$$x_y(y) = -\frac{u_y(x(y), y)}{u_x(x(y), y)}. \tag{4.7}$$

In order to compute the value of $u(x, y)$,$u_x(x, y)$ and $u_y(x, y)$ at prescribed values of x and y, we can then solve equation 4.6 or 4.7 by applying a numerical IVP solver starting from a known intersection point. Furthermore, rather than applying an IVP solver, we can compute approximations to $y(x)$ and $y_x(x)$ (or $x(y)$ and $x_y(y)$) at prescribed values directly from the PCI since the PCI provides approximations to $u(x, y)$ as well as $u_x(x, y)$ and $u_y(x, y)$ at any arbitrary off-mesh point. We can then interpolate the contour curve by applying an appropriate order Hermite interpolant. The minimum number of necessary extra points necessary to define the Hermite interpolant is dependent on the contour curve and some properties of two intersection points. Bradbury and Enright considered one middle point for all situations [1]. However, for some situations that can arise, we need no middle point and the contour curve can be approximated accurately with only the two intersection points.

In the case that the gradient of u at the two intersection points has the same sign (for both x and y directions), we need no middle points and the contour curve can be computed precisely by applying a cubic Hermite interpolant. Figure 4.20 shows such a situation. For the other cases, we will use the approach that will be introduced in the next section, the ODE with arclength method. In addition, the Hermite interpolant may not approximate the solution of this IVP very well when the magnitude of the respective derivative, (y_x or x_y), is much larger than one. We apply the ODE with arclength method for such situations as well.

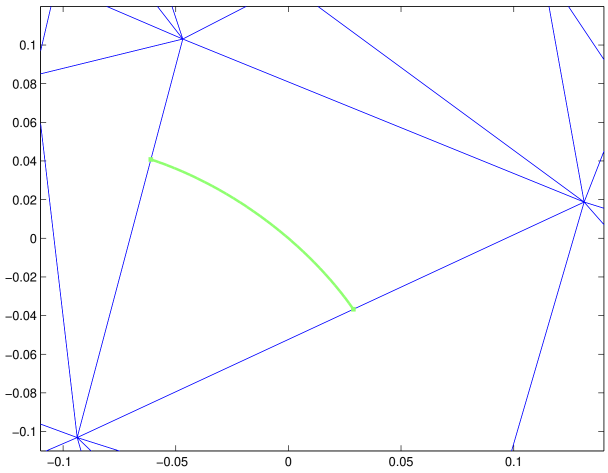


Figure 4.20: The common situation in contour curves.

4.3.3 The ODE with Arclength Method (ODEA)

In the SODE method, we assumed that y is a function of x (or x is a function of y). An alternative method is to parameterize with respect to arclength and assume both x and y as a function of arclength [1]. In this case, we will have

$$u(x(s), y(s)) = v, \tag{4.8}$$

where v is the contour level and s is the arclength. Then if we differentiate the equation 4.8 with respect to s, we will obtain

$$u_x \cdot x_s + u_y \cdot y_s = 0, \tag{4.9}$$

and if we consider a normalization condition for the arclength, we will get

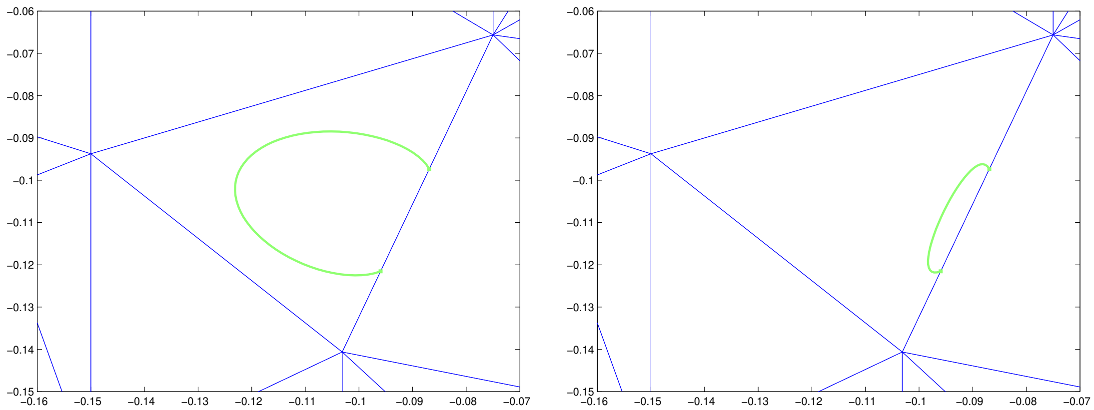
$$x_s{}^2 + y_s{}^2 = 1. \tag{4.10}$$

From equations 4.9 and 4.10 we will have

$$x_s = \frac{\pm u_y}{\sqrt{u_x{}^2 + u_y{}^2}}, \tag{4.11}$$

$$y_s = \frac{\mp u_x}{\sqrt{u_x{}^2 + u_y{}^2}}. \tag{4.12}$$

Note that the sign of $x_s(\pm)$ differs from the sign of $y_s(\mp)$. In other words, if x_s and u_y have the same sign, y_s and u_x should have different sign and vice versa. In order to identify the proper sign in equations 4.11 and 4.12, an intersection test can be done along a single line of refinement. Like the SODE approach, we can obtain the approximations of $x(s)$, $y(s)$, $x_s(s)$ and $y_s(s)$ at each arbitrary point directly from the PCI.



(a) True contour curve.

(b) The contour curve plotted by the ODEA method with no middle point.

Figure 4.21: A situation where the ODEA method fails if no middle point is considered.

Two Hermite interpolants are employed to approximate $x(s)$ and $y(s)$ respectively. In most cases, the data from two intersection points is adequate to accurately approximate the contour curve. However for some cases, the Hermite interpolants fail to determine a suitable approximation and at least one extra point is required. Figure 4.21(a) illustrates one of these situations and figure 4.21(b) shows the result of the Hermite interpolant if we apply it with only two intersection points. By analyzing the problem and observing the result of some special situations, we realized that if there is more than one extrema point on the contour curve (with respect to x or y), the Hermite interpolant may fail to accurately interpolate the contour curve. Note that since the PCI is of degree 3 in x and y, the corresponding polynomial can have at most two extrema points in each direction x or y (obviously one minimum and one maximum). We can assume that the middle point of the contour curve makes these two extrema points place in two different sections.

In order to find the middle point of the contour curve, a similar approach to that employed in the SODE method can be used. After identifying the middle point we can use two cubic Hermite interpolants for computing the contour curve. Figure 4.22 illustrates the result of the situation corresponding to figure 4.21 when we consider the middle point.

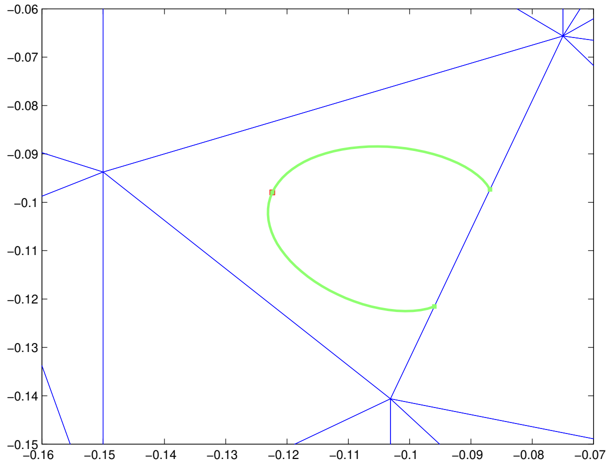


Figure 4.22: The contour curve plotted by the ODEA method with one middle point.

One advantage of the ODEA method is its consistency with the Hermite interpolation.

As we mentioned in the previous section, the Hermite interpolant fails to approximate the solution of the associated IVP accurately if the derivative is much larger than one in magnitude. However, for the ODEA method, due to the normalization condition, it is guaranteed that both x_s and y_s are always less than or equal one.

Since the ODEA method needs fewer middle points than the Intercept method, it computes contours much faster than the Intercept method. However, its results may not be as precise as those of the Intercept method since it is approximating the contours of the interpolant. In order to obtain more accurate results with the ODEA method, we can add some extra middle points and then apply a higher order Hermite interpolant to compute the contour curve. We call this method, the Alternate ODE with Arclength method (AODEA). Like the Intercept method, the user can specify a threshold as a minimum distance of each two neighbor points, obviously greater than the threshold considered for the Intercept method.

Chapter 5

Numerical Results

In this chapter, we will first introduce three test problems. We will then present test results in three separate sections: the results of comparing the PCI with other similar interpolants, the results of the approaches used for a parabolic PDE and the results of comparing our three fast contouring algorithms with other contouring methods. In each section, a testing strategy will be defined.

5.1 Test Problems

We will consider three test problems to measure the performance of the PCI and our contouring algorithms. Each problem has a closed-form solution and hence we will use such a solution to generate the required data at mesh points.

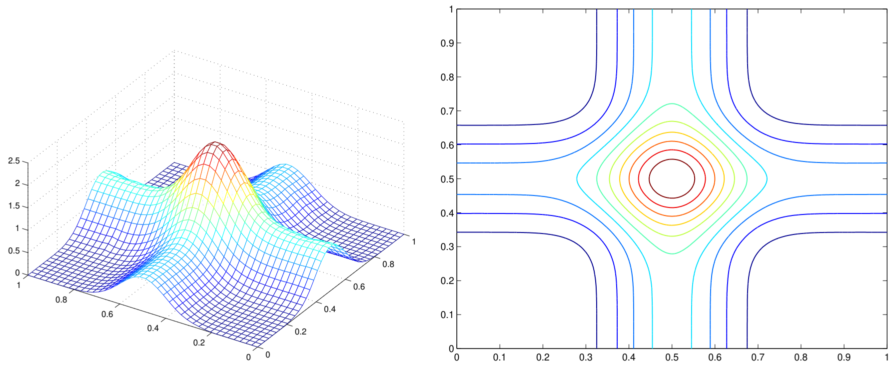
The first test problem is

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= -100e^{(-\frac{(5-10x)^2}{2})} + (50-100x)u_x - 150e^{(-\frac{(5-10x)^2}{2})}e^{(-\frac{(5-10y)^2}{2})} \\ &\quad -75e^{(-\frac{(5-10y)^2}{2})} + (50-100y)u_y \end{aligned} \tag{5.1}$$

and its closed-form solution is

$$u(x,y) = e^{(-\frac{(5-10x)^2}{2})} + 0.75e^{(-\frac{(5-10y)^2}{2})} + 0.75e^{(-\frac{(5-10x)^2}{2})}e^{(-\frac{(5-10y)^2}{2})} \tag{5.2}$$

and figure 5.1 shows its surface and contour plots.



(a) The surface plot (b) The contour plot

Figure 5.1: First test problem: The surface plot and the contour plot.

The second test problem is

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -100cos(10y) - 200sin(10(x-y)) \tag{5.3}$$

and its closed-form solution is

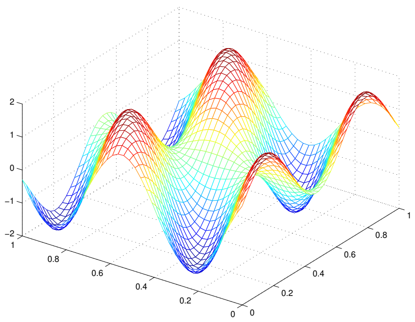
$$u(x,y) = cos(10y) + sin(10(x-y)) \tag{5.4}$$

and figure 5.2 shows its surface and contour plots.

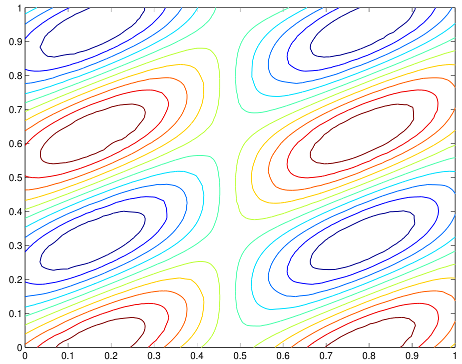
Both of these problems are examples of two-dimensional second-order elliptic PDEs. In addition, we consider a third test problem, a parabolic PDE,

$$\begin{aligned}\frac{\partial u}{\partial y} - \frac{\partial^2 u}{\partial x^2} \quad &= \quad (50-100y)(0.75e^{(-\frac{(5-10y)^2}{2})} + 0.75e^{(-\frac{(5-10x)^2}{2})}e^{(-\frac{(5-10y)^2}{2})}) + \\ &\quad 100(e^{(-\frac{(5-10x)^2}{2})} + 0.75e^{(-\frac{(5-10x)^2}{2})}e^{(-\frac{(5-10y)^2}{2})}) - (50-100x)u_x \end{aligned} \tag{5.5}$$

and the closed-form solution is the same as that of the first test problem i.e. equation 5.2.



(a) The surface plot



(b) The contour plot

Figure 5.2: Second test problem: The surface plot and the contour plot.

5.2 Test Results: The PCI

In chapter 3, where the PCI was introduced, we argued that the PCI is efficient in terms of time and accuracy. In this section, we compare the PCI with the standard DEI, the ADEI, DEI3-15, DEI3-13 and DEI4-15. In addition to time and accuracy, we consider visual appearance to show how the PCI generates a continuous contour or surface plot.

Table 5.1 shows the average error of the PCI and the other interpolants for the first and second test problems and different number of mesh points. It does not contain the average error of the ADEI for 2000 and 5000 mesh points due to the long time required for the computation. The scattered data is generated by using a random approach and triangulized by the built-in MATLAB function *delaunay*. As can be seen from the table, for the first problem, the PCI generates the smallest average error for all cases. For the second problem, the ADEI and the DEI4-15 generate more accurate results for some cases. It seems that for the case with a large number of mesh points (1000 and more), the DEI4-15 is more accurate than the other approaches. Table 5.2 presents the corresponding required CPU time. The PCI is the most efficient interpolant in terms of time for both test problems. It also can be seen that the ADEI, that is almost the second best one in terms of accuracy, needs much more time than the others because of

Problem	# Mesh Points	*Algorithm*					
		PCI	DEI	ADEI	DEI3-13	DEI3-15	DEI4-15
#1	100	0.034112	0.105088	0.039668	0.901402	0.161128	0.036194
	200	0.004943	0.292204	0.005256	0.005956	0.033420	0.007144
	500	0.000703	0.004561	0.000876	0.001542	0.004836	0.000916
	1000	0.000208	0.003134	0.000278	0.000426	0.001198	0.000210
	2000	0.000071	0.001873	——	0.000917	0.000407	0.000085
	5000	0.000009	0.000077	——	0.000019	0.000116	0.000011
#2	100	0.050886	0.283763	0.054867	1.083352	0.203339	0.053035
	200	0.008985	5.637942	0.005246	0.008887	0.045101	0.008788
	500	0.001628	0.015582	0.001577	0.002702	0.006574	0.000745
	1000	0.000320	0.001114	0.000360	0.000476	0.001252	0.000083
	2000	0.000125	0.000644	——	0.000168	0.000430	0.000019
	5000	0.000015	0.000081	——	0.000035	0.000088	0.000001

Table 5.1: Average error for the PCI and the other interpolants on an unstructured triangular mesh with different number of mesh points for the two test problems.

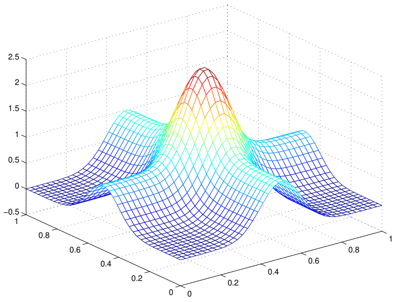
the large number of iterations.

Figures 5.3 and 5.4 show the surface plots and the contour plots generated by different interpolants for the first test problem on a triangular mesh with 500 mesh points. Figures 5.5 and 5.6 show the same plots on a triangular mesh with 2000 mesh points. In addition, Figures 5.7 and 5.8 show the surface plots and the contour plots generated by different interpolants for the second test problem on a triangular mesh with 500 mesh points and figures 5.9 and 5.10 show the same plots on a triangular mesh with 2000 mesh points. The PCI is the only one that always generates a continuous surface or contour

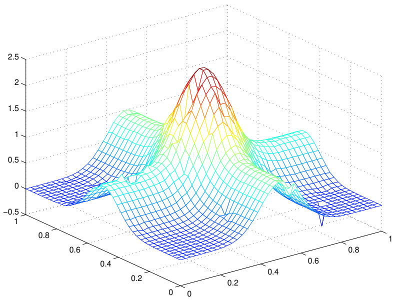
Problem	# Mesh Points	*Algorithm*					
		PCI	DEI	ADEI	DEI3-13	DEI3-15	DEI4-15
#1	100	0.38	1.21	95.63	0.77	1.10	1.10
	200	0.77	2.48	628.13	1.59	2.25	2.19
	500	2.31	6.54	3570.60	4.22	5.77	5.82
	1000	3.85	13.35	19422.59	8.68	11.92	11.81
	2000	7.58	26.64	——	17.25	23.72	23.51
	5000	18.68	69.92	——	44.66	62.23	61.90
#2	100	0.39	1.27	170.05	0.77	1.10	1.09
	200	0.82	2.64	405.51	1.76	2.36	2.31
	500	1.87	6.54	1189.80	4.18	5.77	5.77
	1000	3.85	13.19	4979.88	8.62	11.75	11.70
	2000	7.91	26.81	——	17.25	23.72	23.57
	5000	18.46	69.20	——	44.27	61.68	61.24

Table 5.2: Total required time (in terms of seconds) for the PCI and the other interpolants on an unstructured triangular mesh with different number of mesh points for the two test problems.

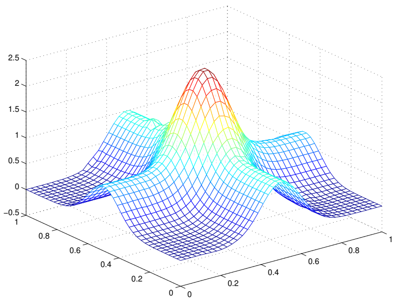
plot for these test problems. The contour plots have been generated by the built-in MATLAB function *contour* on a fine grid of size 40*40.



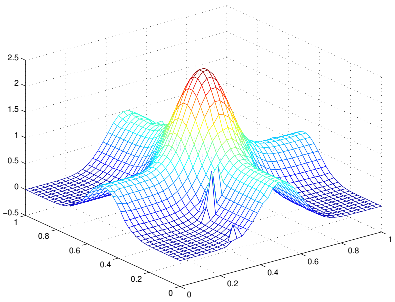
(a) The PCI



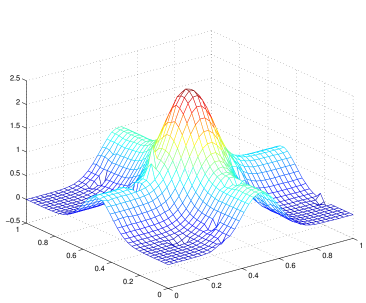
(b) The Standard DEI



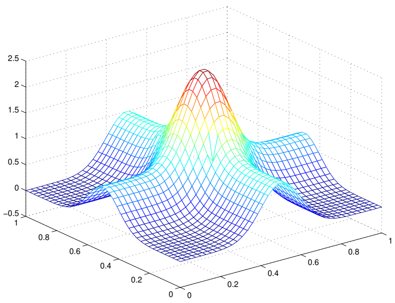
(c) The ADEI



(d) The DEI3-13

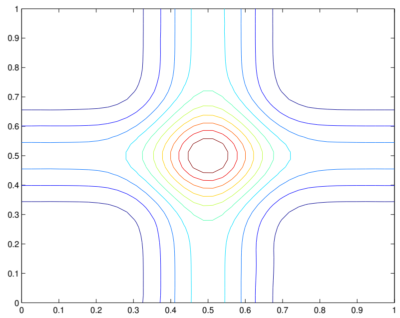


(e) The DEI3-15

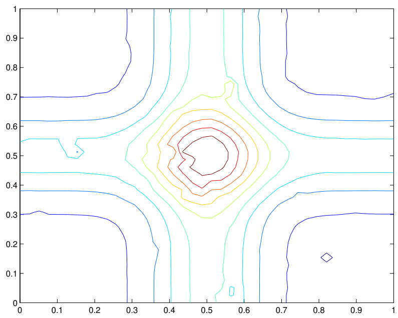


(f) The DEI4-15

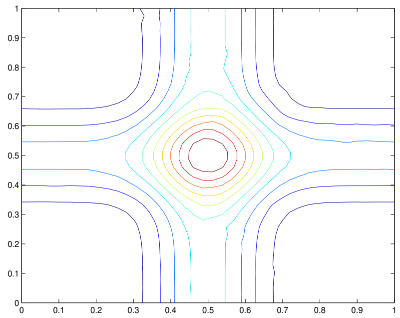
Figure 5.3: The surface plots of the PCI and the other interpolants for the first test problem on a triangular mesh with 500 mesh points.



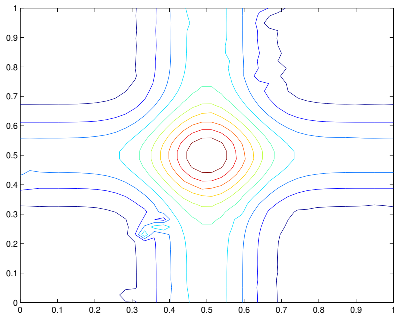
(a) The PCI



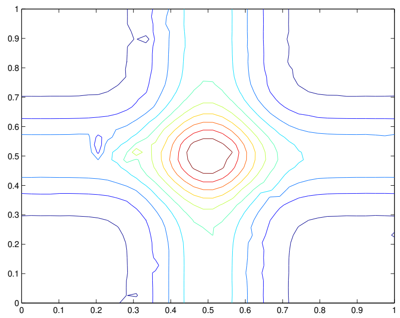
(b) The Standard DEI



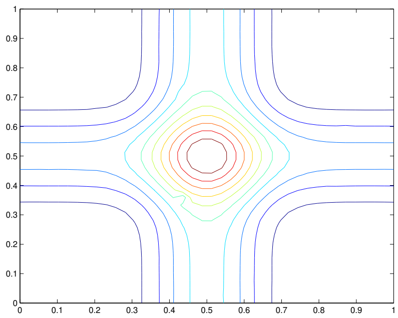
(c) The ADEI



(d) The DEI3-13

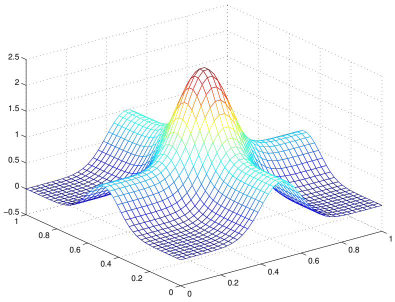


(e) The DEI3-15

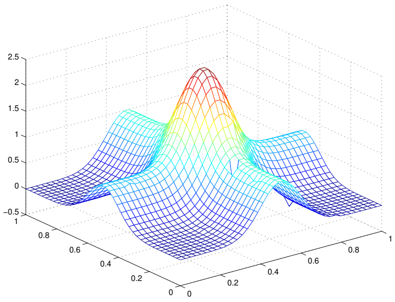


(f) The DEI4-15

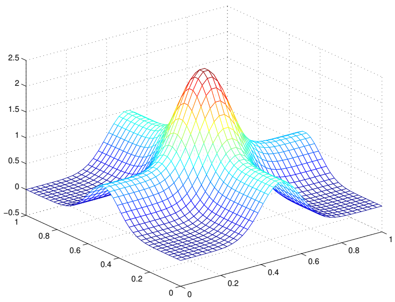
Figure 5.4: The contour plots of the PCI and the other interpolants for the first test problem on a triangular mesh with 500 mesh points.



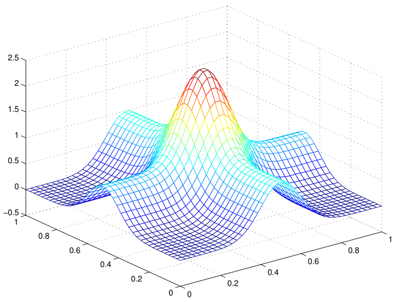
(a) The PCI



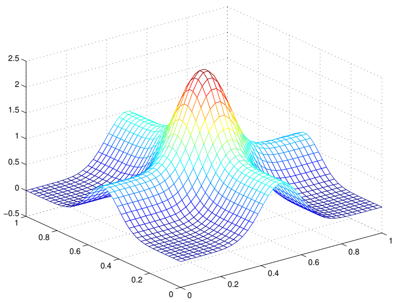
(b) The Standard DEI



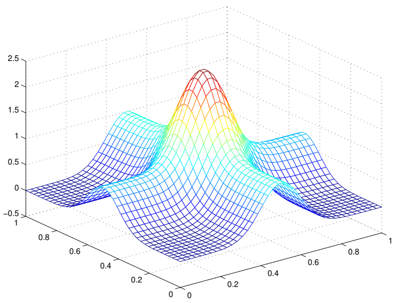
(c) The ADEI



(d) The DEI3-13

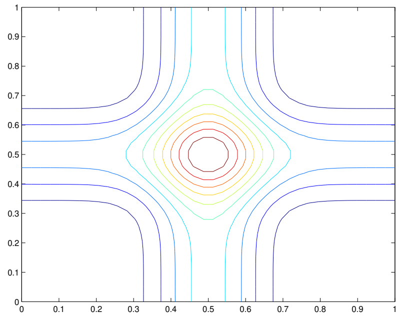


(e) The DEI3-15

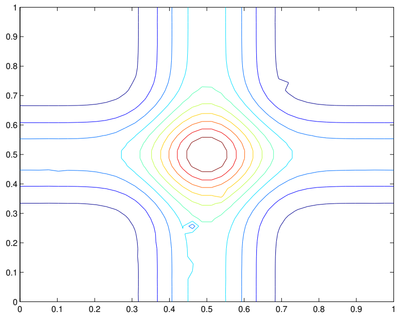


(f) The DEI4-15

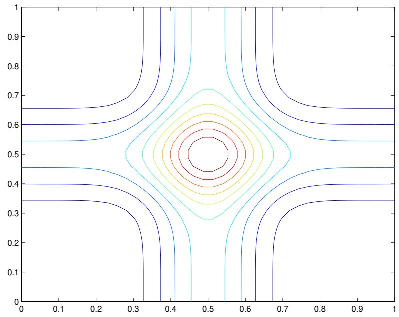
Figure 5.5: The surface plots of the PCI and the other interpolants for the first test problem on a triangular mesh with 2000 mesh points.



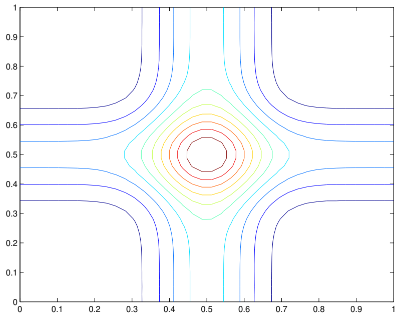
(a) The PCI



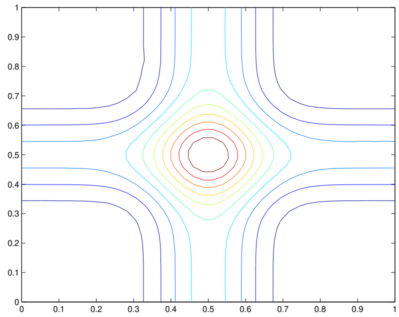
(b) The Standard DEI



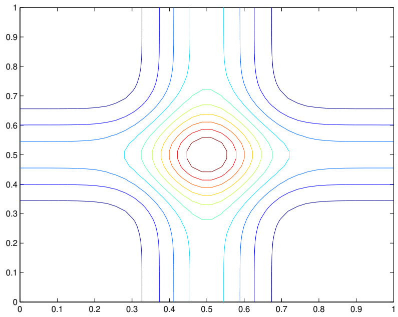
(c) The ADEI



(d) The DEI3-13

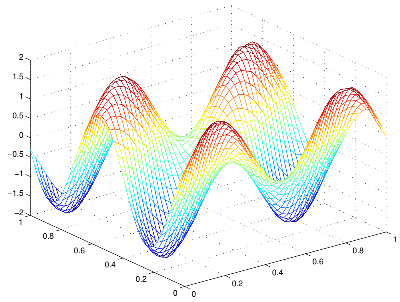


(e) The DEI3-15

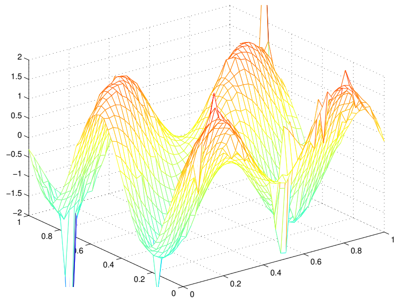


(f) The DEI4-15

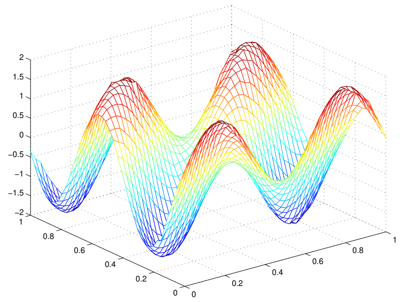
Figure 5.6: The contour plots of the PCI and the other interpolants for the first test problem on a triangular mesh with 2000 mesh points.



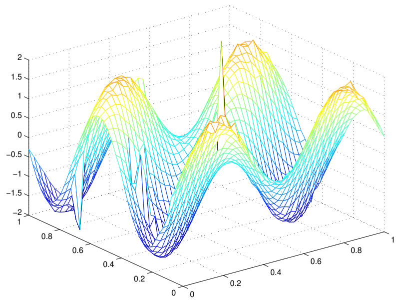
(a) The PCI



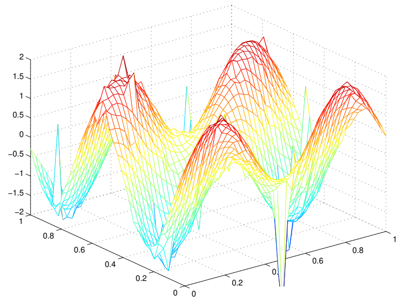
(b) The Standard DEI



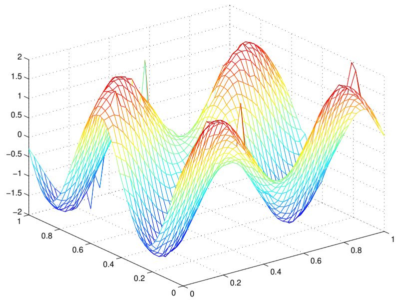
(c) The ADEI



(d) The DEI3-13

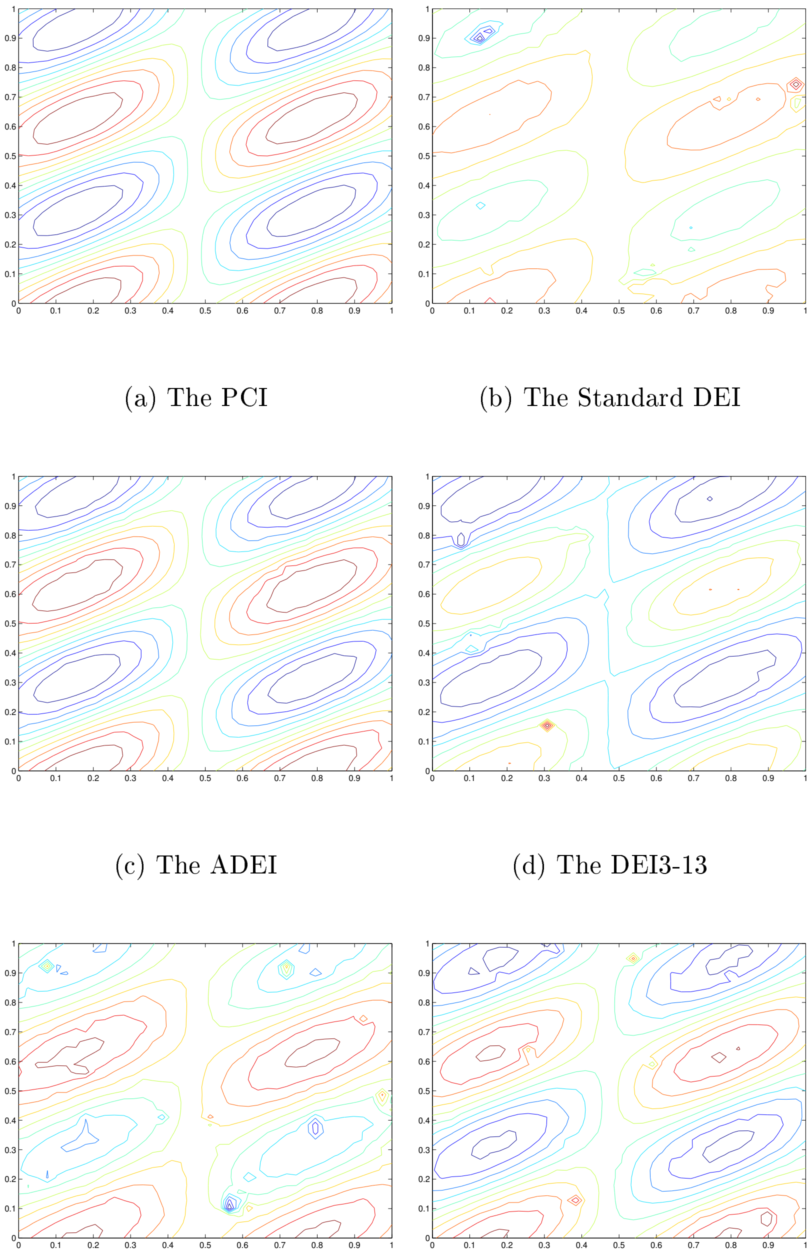


(e) The DEI3-15



(f) The DEI4-15

Figure 5.7: The surface plots of the PCI and the other interpolants for the second test problem on a triangular mesh with 500 mesh points.



(a) The PCI

(b) The Standard DEI

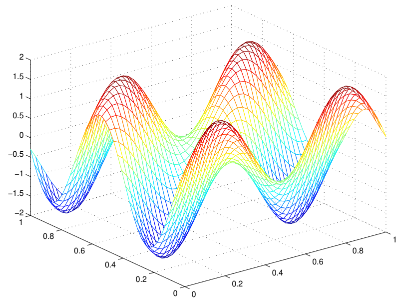
(c) The ADEI

(d) The DEI3-13

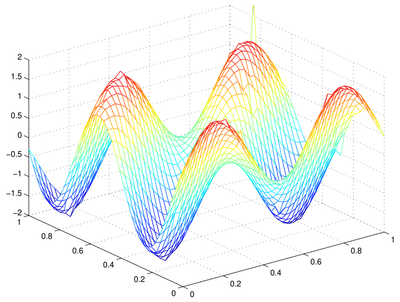
(e) The DEI3-15

(f) The DEI4-15

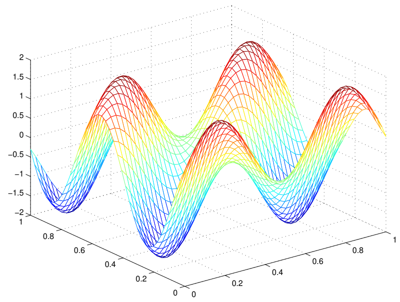
Figure 5.8: The contour plots of the PCI and the other interpolants for the second test problem on a triangular mesh with 500 mesh points.



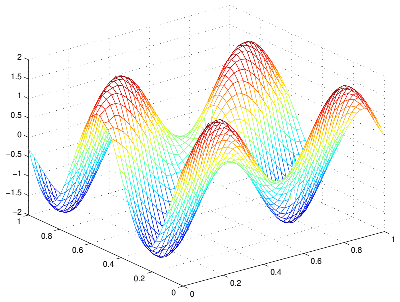
(a) The PCI



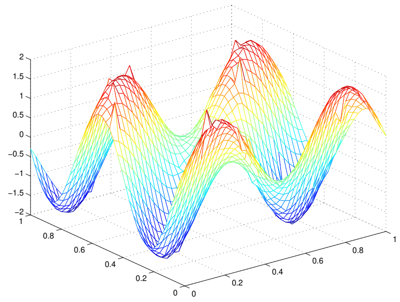
(b) The Standard DEI



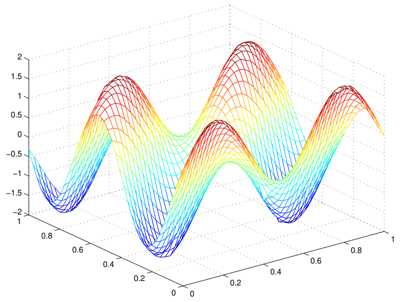
(c) The ADEI



(d) The DEI3-13

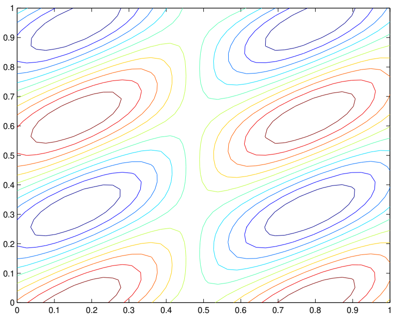


(e) The DEI3-15

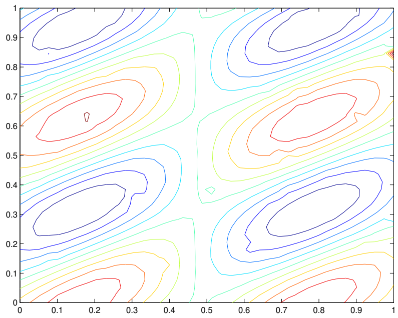


(f) The DEI4-15

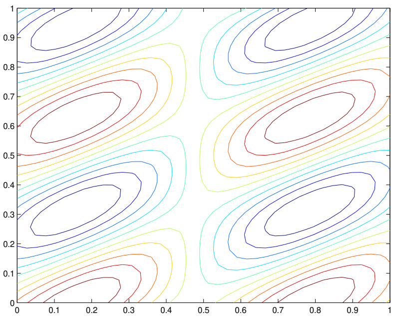
Figure 5.9: The surface plots of the PCI and the other interpolants for the second test problem on a triangular mesh with 2000 mesh points.



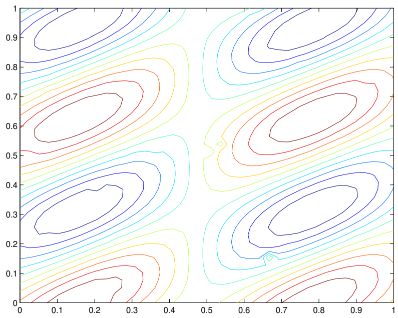
(a) The PCI



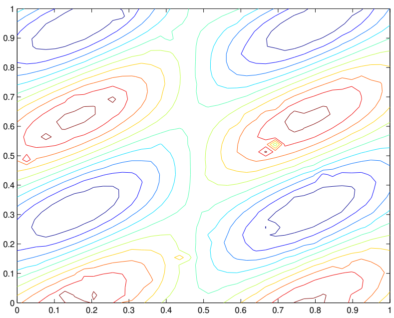
(b) The Standard DEI



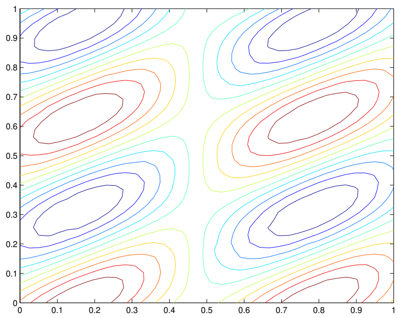
(c) The ADEI



(d) The DEI3-13



(e) The DEI3-15



(f) The DEI4-15

Figure 5.10: The contour plots of the PCI and the other interpolants for the second test problem on a triangular mesh with 2000 mesh points.

5.3 Test Results: The DEI and Parabolic PDEs

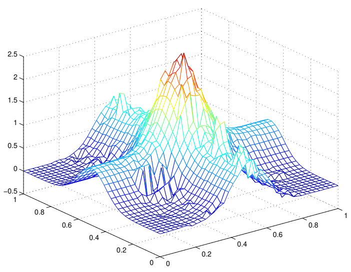
In chapter 3, we introduced 10 different situations to apply the DEI approach for a parabolic PDE based on solving a linear system with nine or ten unknown coefficients. These situations have been shown in figures 3.2 and 3.3 in chapter 3. Unfortunately, the 1-1-0 situations (figures 3.2(a) and 3.3(a)), that corresponds to the more accurate and continuous results, resulted in the matrix W being singular for some mesh elements. In tables 5.3 and 5.4, the average error and the corresponding CPU time for the other eight situations have been presented for different number of mesh points. In addition, figures 5.11 and 5.12 show the surface plots and the contour plots generated by the six different methods based on the location of the collocation points and symmetric/non-symmetric matrices on a triangular mesh with 500 mesh points. Figures 5.13 and 5.14 show the same plots on a triangular mesh with 2000 mesh points. Although both approaches (symmetric and non-symmetric) produce discontinuous results, the non-symmetric approach generates relatively better results with regard to accuracy and visualization.

# Mesh	*Algorithm*							
Points	S:1-0-0	S:2-1-0	S:10-1-0	S:10-1-1	N:1-0-0	N:2-1-0	N:10-1-0	N:10-1-1
100	0.2599	0.4527	0.2786	0.2943	0.1063	0.2468	0.0816	0.0809
200	0.0979	0.1368	0.1051	0.1023	0.0337	0.1084	0.0284	0.0269
500	0.0302	0.0336	0.0303	0.0307	0.0089	0.0136	0.0078	0.0073
1000	0.0108	0.0141	0.0111	0.0109	0.0043	0.0081	0.0034	0.0025
2000	0.0042	0.0059	0.0042	0.0041	0.0024	0.0038	0.0019	0.0015
5000	0.0021	0.0025	0.0021	0.0021	0.0016	0.0018	0.0015	0.0014

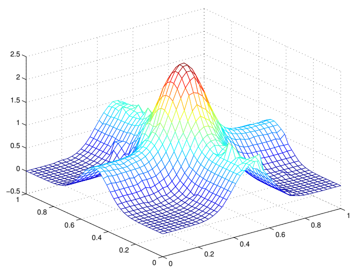
Table 5.3: Average error for the different approaches for the third test problem on an unstructured triangular mesh with different number of mesh points(S: Symmetric, N: Non-symmetric).

# Mesh	*Algorithm*							
Points	S:1-0-0	S:2-1-0	S:10-1-0	S:10-1-1	N:1-0-0	N:2-1-0	N:10-1-0	N:10-1-1
100	1.69	1.55	1.55	1.69	1.12	1.13	1.46	1.43
200	3.59	3.18	3.87	3.01	2.70	1.93	2.60	2.31
500	10.96	7.78	8.51	7.80	7.10	6.72	6.96	6.14
1000	17.17	16.98	16.71	16.23	12.17	15.21	14.68	13.29
2000	33.56	31.51	32.14	30.56	25.51	26.19	27.45	24.49
5000	82.51	81.29	81.63	77.92	64.54	66.07	57.02	59.59

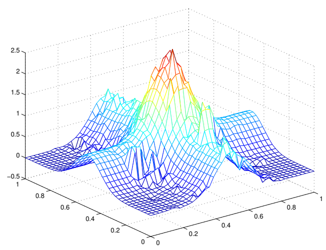
Table 5.4: Total required time (in terms of seconds) for the different approaches for the third test problem on an unstructured triangular mesh with different number of mesh points(S: Symmetric, N: Non-symmetric).



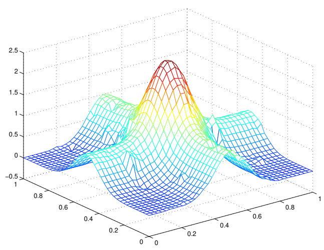
(a) Symmetric 1-0-0



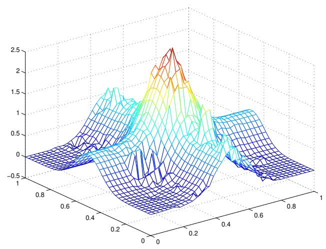
(b) Non-symmetric 1-0-0



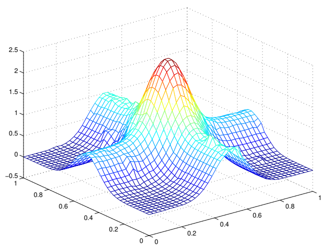
(c) Symmetric 2-1-0



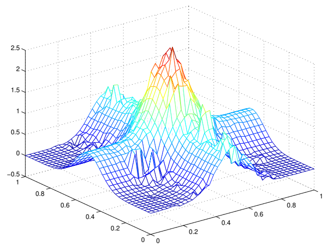
(d) Non-symmetric 2-1-0



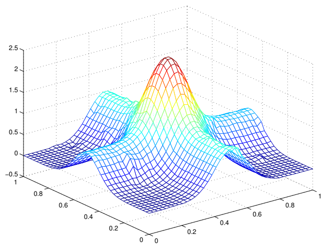
(e) Symmetric 10-1-0



(f) Non-symmetric 10-1-0

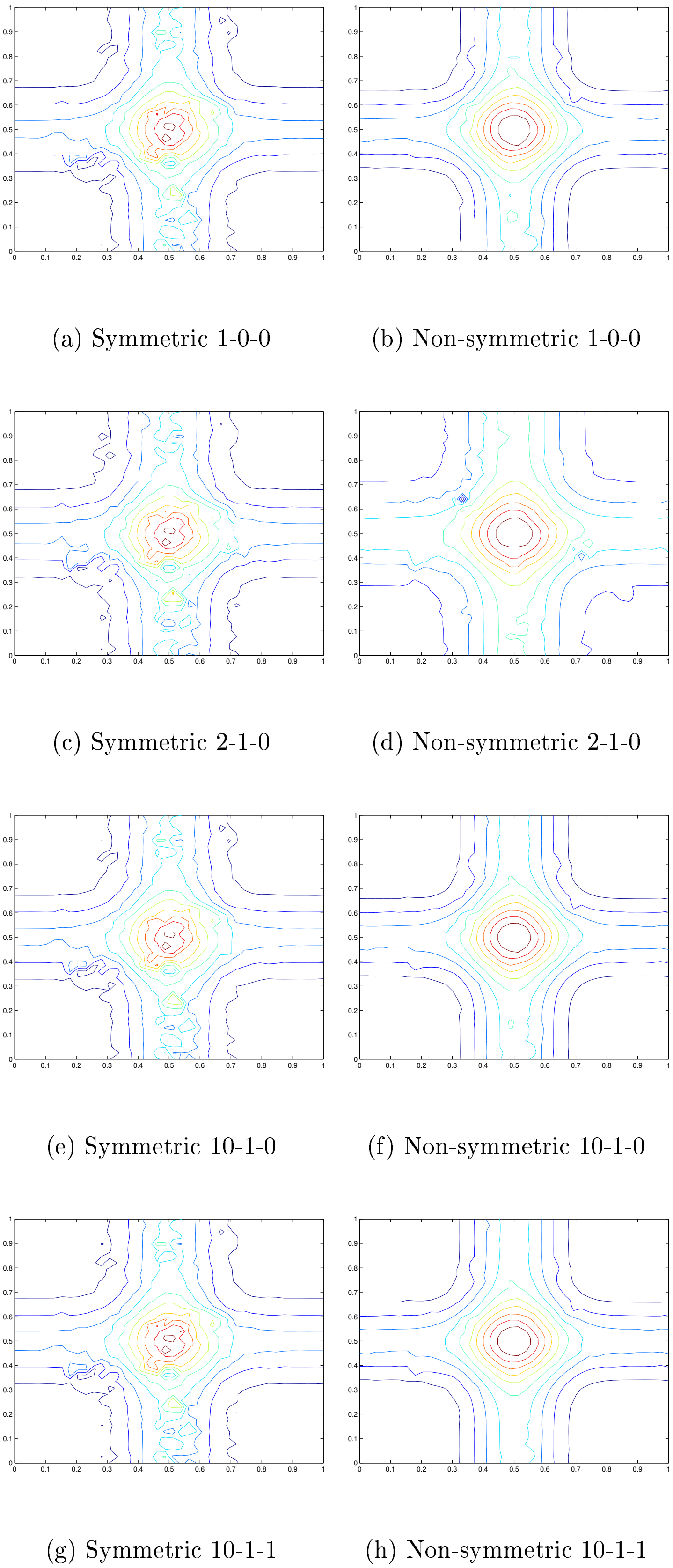


(g) Symmetric 10-1-1



(h) Non-symmetric 10-1-1

Figure 5.11: The surface plots of the different approaches for the third test problem on a triangular mesh with 500 mesh points.



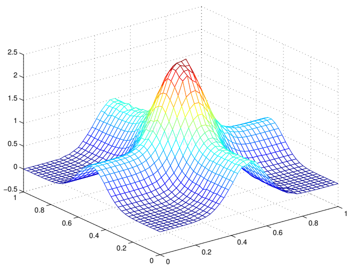
(a) Symmetric 1-0-0 (b) Non-symmetric 1-0-0

(c) Symmetric 2-1-0 (d) Non-symmetric 2-1-0

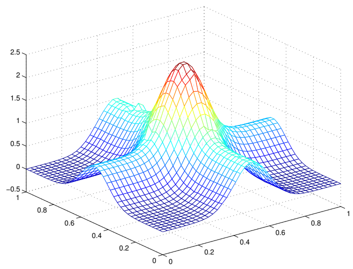
(e) Symmetric 10-1-0 (f) Non-symmetric 10-1-0

(g) Symmetric 10-1-1 (h) Non-symmetric 10-1-1

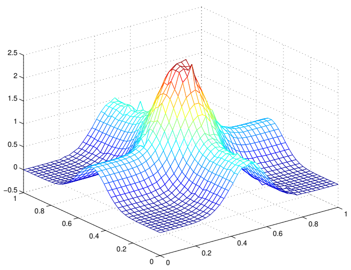
Figure 5.12: The contour plots of the different approaches for the third test problem on a triangular mesh with 500 mesh points.



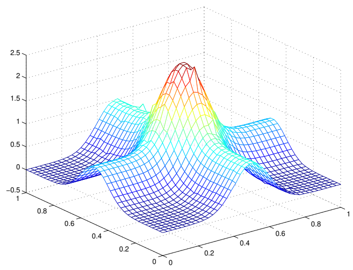
(a) Symmetric 1-0-0



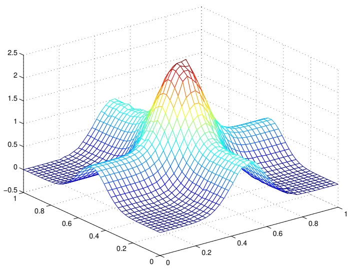
(b) Non-symmetric 1-0-0



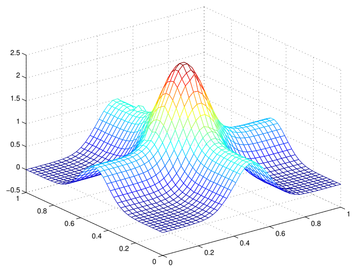
(c) Symmetric 2-1-0



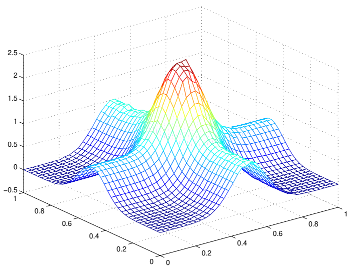
(d) Non-symmetric 2-1-0



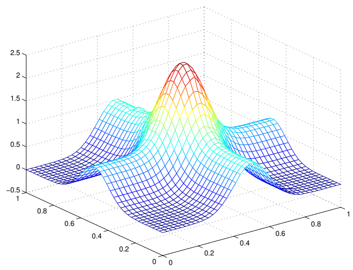
(e) Symmetric 10-1-0



(f) Non-symmetric 10-1-0

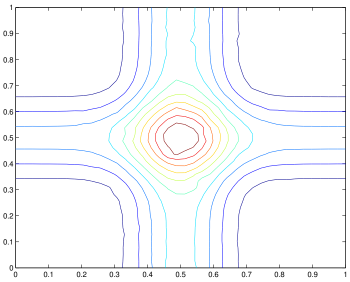


(g) Symmetric 10-1-1

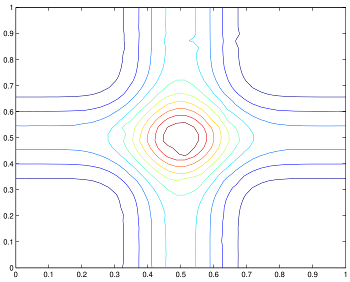


(h) Non-symmetric 10-1-1

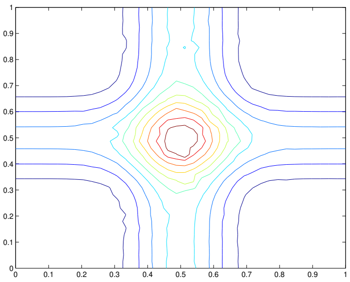
Figure 5.13: The surface plots of the different approaches for the third test problem on a triangular mesh with 2000 mesh points.



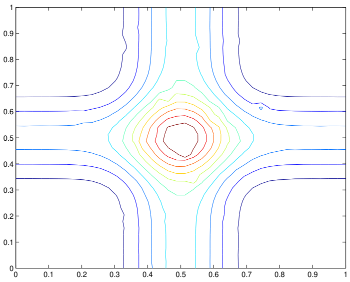
(a) Symmetric 1-0-0



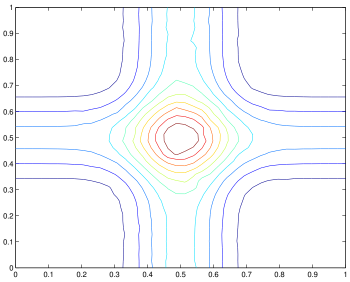
(b) Non-symmetric 1-0-0



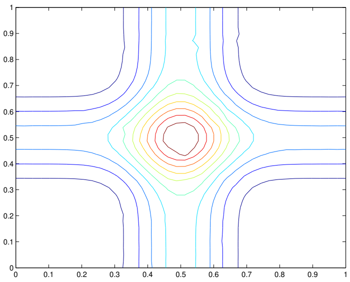
(c) Symmetric 2-1-0



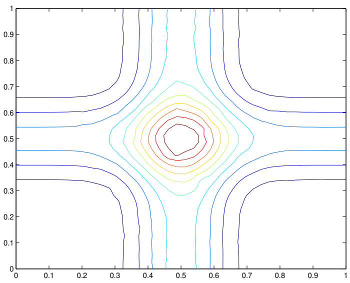
(d) Non-symmetric 2-1-0



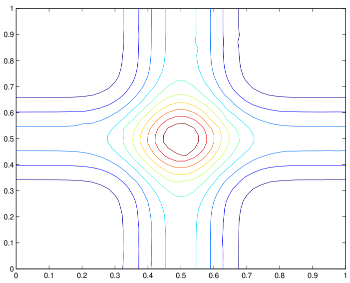
(e) Symmetric 10-1-0



(f) Non-symmetric 10-1-0



(g) Symmetric 10-1-1



(h) Non-symmetric 10-1-1

Figure 5.14: The contour plots of the different approaches for the third test problem on a triangular mesh with 2000 mesh points.

5.4 Test Results: Fast Contouring Algorithms

In this section, we will first compare our fast contouring algorithms, introduced in chapter 4, with the MATLAB function *contour* and those addressed by Bradbury and Enright in [1]. In order to compare the methods, a uniform distribution of the data is considered because the MATLAB function *contour* and their contouring algorithms need the data on a rectangular mesh. Both contouring algorithms implemented by Bradbury and Enright, FCINT and FCODE, are used for comparison [1]. Table 5.5 shows the average error of applying the methods on a rectangular mesh with different number of mesh points for the third test problem. For each part of a contour curve that lies in a mesh element, a random point on the curve is considered as a sample point. Therefore the total number of sample points depends on the size of the underlying mesh and the number of contour curves. In addition, table 5.6 shows the corresponding required time. Since FCINT and FCODE are based on a parabolic PDE, the comparison has been done only for the third test problem. From tables 5.5 and 5.6, it can be inferred that our algorithms can be both more accurate and faster than the MATLAB function *contour*, FCINT and FCODE.

# Mesh Points	*Algorithm*					
	MATLAB	FCINT	FCODE	Intercept	SODE	ODEA
30*30	0.3066	0.001900	0.002300	0.00001270	0.00001060	0.00001010
40*40	0.4522	0.000741	0.000811	0.00000798	0.00000291	0.00000235
50*50	0.4605	0.000273	0.000234	0.00000766	0.00000213	0.00000183

Table 5.5: Average error for the different methods for a rectangular mesh with 30*30, 40*40 and 50*50 mesh points.

# Mesh	*Algorithm*					
Points	Matlab	FCINT	FCODE	Intercept	SODE	ODEA
30*30	14.66	9.78	9.50	9.61	5.33	5.54
40*40	23.18	17.19	16.59	13.40	9.33	9.50
50*50	35.40	22.08	21.75	18.95	14.77	15.21

Table 5.6: Total required time (in terms of seconds) for the different methods for a rectangular mesh with 30*30, 40*40 and 50*50 mesh points.

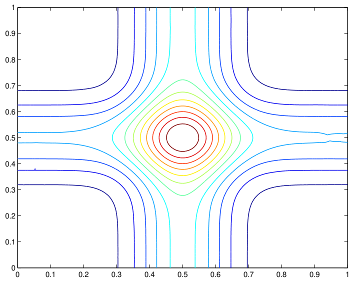
Although the Intercept method is not as accurate as the SODE and ODEA methods for this rectangular mesh, it was more accurate than the others for the unstructured triangular meshes we investigated. However the total required time for the Intercept method is more. Tables 5.7 and 5.8 show the average error and corresponding required time of applying our fast contouring algorithms on an unstructured triangular mesh with 900, 1600 and 2500 mesh points. In chapter 4, an advanced ODEA algorithm, the AODEA, was introduced to improve the accuracy of ODEA method. The results of the AODEA method are also presented in tables 5.7 and 5.8. As can be seen, the AODEA method is more accurate than the ODEA but needs more time as well. The accuracy of the AODEA method can be controlled by the value of the applied threshold. Figure 5.15 shows the contour plots generated by our fast contouring algorithms on unstructured triangular meshes with 900 and 2500 mesh points.

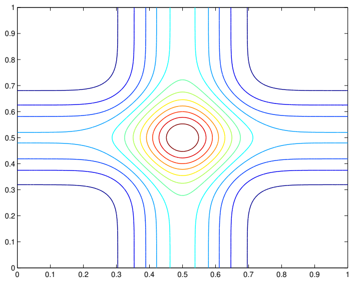
# Mesh	*Algorithm*			
Points	Intercept	SODE	ODEA	AODEA
900	0.00000903	0.000687	0.000687	0.0000247
1600	0.00000655	0.0000299	0.0000226	0.0000156
2500	0.00000546	0.0000128	0.0000123	0.00000663

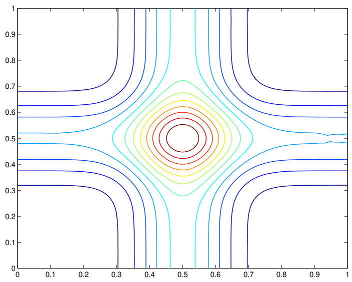
Table 5.7: Average error for the different methods for an unstructured triangular mesh with 900, 1600 and 2500 mesh points.

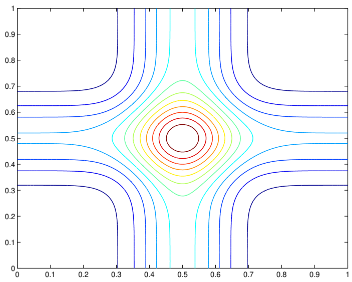
# Mesh	*Algorithm*			
Points	Intercept	SODE	ODEA	AODEA
900	10.98	5.43	5.55	6.65
1600	14.11	9.28	9.39	10.49
2500	18.35	13.62	13.79	15.54

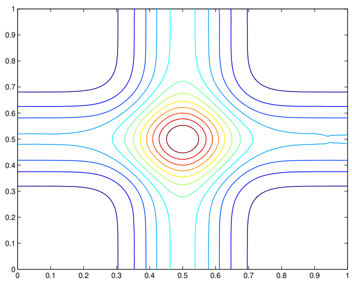
Table 5.8: Total required time (in terms of seconds) for the different methods for an unstructured triangular mesh with 900, 1600 and 2500 mesh points.

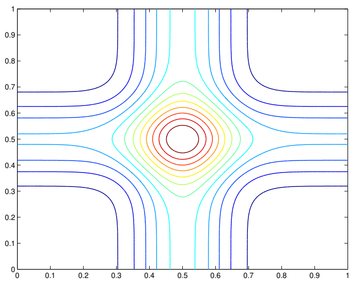

(a) Intercept (900 points)

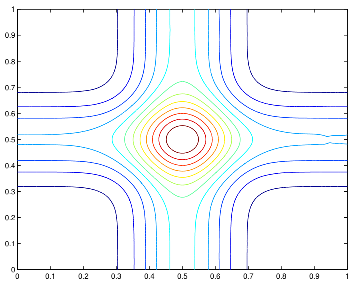

(b) Intercept (2500 points)


(c) SODE (900 points)


(d) SODE (2500 points)


(e) ODEA (900 points)


(f) ODEA (2500 points)


(g) AODEA (900 points)

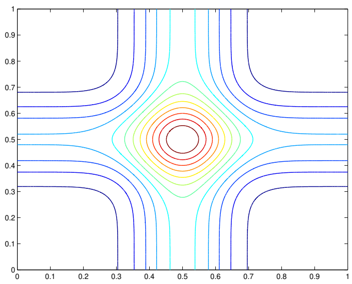

(h) AODEA (2500 points)

Figure 5.15: The contour plots for the first test problem on triangular meshes with 900 and 2500 mesh points.

5.5 Conclusions and Future Work

We introduced the PCI, an interpolant over scattered data based on the DEI that accurately approximates the solution of PDEs at off-mesh points. The most significant advantage of the PCI is that it can be used to generate a continuous plot. We compared the PCI with some methods derived from the DEI approach and demonstrated that the PCI is the best one in terms of continuity and time. It is also one of the best in terms of accuracy. In addition, we introduced three fast contouring algorithms based on the PCI. These contouring algorithms were compared to the built-in MATLAB function *contour* and the contouring algorithms introduced by Bradbury and Enright. For rectangular meshes, we demonstrated that our algorithms are both more accurate and faster than the others. We also showed that they are fast and accurate for arbitrary distributed data sets.

We have continued the investigation of using the DEI to accurately approximate and visualize the solution of PDEs at off-mesh points. To continue our work, we will consider:

- Extending the PCI to 3 dimensional scattered data sets.
- Finding a continuous approach based on the DEI for parabolic PDEs.
- Modifying the basic triangulation in order to obtain a better piecewise interpolant rather than satisfying delaunay triangulation constraints [5].
- Working on contouring algorithms in higher dimensions.

Bibliography

[1] Emma L. Bradbury and Wayne H. Enright. Fast contouring of solutions to partial differential equations. *ACM Transaction on Mathematical Software*, 29(4):418–439, December 2003.

[2] Wayne H. Enright. Accurate approximate solution of partial differential equations at off-mesh points. *ACM Transaction on Mathematical Software*, 26(2):274–292, June 2000.

[3] MathWorks. *MATLAB online documentation*, 12 edition.

[4] Gonzalo A. Ramos and Wayne H. Enright. Interpolation of surfaces over scattered data. *Proceedings of the IASTED International Conference VISUALIZATION, IMAGING, AND IMAGE PROCESSING*, pages 219–224, September 2001.

[5] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.