Scattered Data Interpolation Using an Alternate Differential Equation Interpolant

by

Gonzalo A. Ramos

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

Abstract

Scattered Data Interpolation Using an

Alternate Differential Equation Interpolant

Gonzalo A. Ramos

Master of Science

Graduate Department of Computer Science

University of Toronto

2001

We investigate the performance of DEI, an approach [9] that computes off-mesh approximations of PDE solutions, and can also be used as a technique for scattered data interpolation. We compare the new approach with two methods from the collection of ACM algorithms: Algorithms 752, and 790. For the general case of unstructured meshes, we found it necessary to modify the original DEI. The resulting method, ADEI, adjusts the parameter of the interpolant, obtaining better performance. Finally, we measure ADEI's performance using different sets of scattered data and test functions. The results show that ADEI is better if not comparable to the best of the available scattered data interpolation techniques.

Dedication

To my mentors -past, present, and future- for all my annoying questions they have endured, continue to bear, and have yet to even anticipate.

To my friends -because they understand my, ahem, somewhat caustic sense of humor.

And to my family, absent and present. They are my inspiration.

Acknowledgements

I wish to thank Ana Ruedin, for letting me express myself through teaching, and to Rebecca Spagnolo, for finishing touches and being more than meet the eye. My gratitude also to Wayne Hayes, for helping me to speak my ideas aloud (whiteboards are good listeners, but not good enough), to the “Alanoman” (Special Agent Alan Skelly) for support on all things LaTeX, and to Christina Christara for invaluable formatting and proofreading.

Special thanks to my supervisor, Wayne Enright, for more insightful talks, debugging sessions, proofreading, and advice than I could ever count.

Contents

Chapter 1

Introduction.

1.1 Motivation.

Scientific visualization can be viewed as the use of computer graphics to display data. It is a discipline that combines among others: computer graphics, numerical analysis, and human computer interaction. By converting raw numerical data into graphic information, Scientific visualization provides researchers with a powerful tool to explore and analyze data, gain scientific knowledge and communicate their findings.

Data can come from measurements in the real word, or from numerical processes; in both cases the data usually originates from a continuous domain, but only a finite number of uniform or non-uniform samples is available. Since most scientific visualization systems need their input data to be defined over a regular grid, an interpolant is often required to generate the information associated with this uniform visualization mesh.

When data is collected in an irregular fashion it is said to be scattered, irregular, or random. Scattered data arise in a number of fields and problems such as:

- Earth Sciences: Geography, Geology, Geodesy, Hydrology, and Geophysics.
- Meteorology: Weather measurements and Forecast.

- Engineering: Structural Analysis and Design.
- Medicine: Image reconstruction from samples.

To visualize results, these applications require the use of interpolation schemes, different from the ones used on uniform data. The interpolants will fit, with different degrees of success, smooth functions to the scattered data, and in the end one expects that the interpolating function obtained in this way will approximate the original underlying function that is the source of the sampled data.

For some applications, like free-form modeling, the user has control over the shape of the interpolant through handles or control points. In that case, the accuracy of the solution is not critical because there is a continuous feedback with the user that allows him or her to adjust the interpolant until a desired level of approximation is reached. On the other hand, for applications like scientific visualization packages, the user requires certain accuracy on the interpolant over a fixed data set. Here, the user is more concerned with precision than with aesthetics.

In this paper, we are concerned with the precision of the approximation, so we focus on the problem of accurate interpolation. The next section defines precisely the problem of scattered data interpolation.

1.2 Scattered data Interpolation

The scattered data interpolation problem can be defined as: Given a set of n irregularly distributed points

$$P_i = (x_i, y_i) \ , \ i = 1, ..., n \tag{1.1}$$

over $\Re^2$, and scalar values F_i associated with each point satisfying $F_i = F(x_i, y_i)$ for some underlying function $F(x, y)$, look for an interpolating function $\overline{F} \approx F(x, y)$ such that for $i = 1, \ldots, n$

$$\overline{F}(x_i, y_i) = F_i \tag{1.2}$$

We assume that all the points P_i (sometimes referred as nodes or mesh points) are distinct and that all the points are not collinear. This formulation can be generalized into higher dimensions in a straightforward way, but for the remainder of this paper, we will concentrate in the case where the scattered points P_i are on a plane, making the function $\overline{F}$ a bivariate function that defines a surface in 3D space.

Given a particular use for the interpolating function $\overline{F}$, one may desire certain properties, such as smoothness, continuity or differentiability. For instance, smooth surfaces with C^1 continuity are sufficient for most of the applications that require visualization of surfaces. But if we are interested in visualizing a vector field, where the gradient of F is of relevance, a higher order of continuity might be needed. We may also want the function to be expressed in explicit, implicit or procedural form. That influences issues such as evaluation, interrogation, rendering and encoding of the interpolant. Restrictions on the speed and efficiency of computation may also be important.

Because there are particular attributes associated with each interpolating method, there is not a unique way to classify them. For example, the method could be global or local, based on the nodes needed to evaluate the function $\overline{F}$ at certain location (x, y). The form of the interpolating function could also be used for classification, since $\overline{F}$ could be a bivariate polynomial, a piecewise bivariate polynomial, or a rational function. In the next section we present techniques and principles that are representative of the current work in the field.

Chapter 2

Previous and Current work

The subject of scattered data interpolation is extensive. This chapter presents an overview of some of the most popular techniques that are widely used in current methods. Emphasis will be on the techniques and principles behind the two ACM algorithms we compare in Chapter 4.

2.1 Shepard's Method

In distance-based methods, the value of the interpolant at a point (x, y) is a weighted sum of functions that depend on the distance from the point to all (or some) of the mesh points. In their initial formulation, distance-based methods are global, because the evaluation of $\overline{F}(x, y)$ requires all the data points (x_i, y_i). However, these techniques can be modified into local schemes, i.e., where $\overline{F}(x, y)$ depends only on mesh points in some neighborhood of (x, y). Global techniques are expensive for large number of mesh points, but it is possible to apply them to overlapping subsets, and blend the solutions into a single interpolant for the whole set.

The *Shepard's Method* is one of the earliest techniques used to generate interpolants for scattered data. It defines an interpolating function S that is the weighted average of the value at the mesh points. Let us define the weights w_i to be inversely proportional to the Euclidean distance from a point (x, y) to a point (x_i, y_i)

$$w_i(x,y) = \|(x,y) - (x_i, y_i)\|_2^{-q} \ , \ q > 0 \tag{2.1}$$

The function S is defined as

$$S(x,y) = \sum_{i=1}^{n} F_i v_i(x,y) \tag{2.2}$$

where

$$v_i(x,y) = \frac{w_i(x,y)}{\sum_{k=1}^{n} w_k(x,y)} \tag{2.3}$$

From (2.2)and (2.3) we can see that S interpolates the function F at the points P_i, i.e.,

$$S(x_i, y_i) = F(x_i, y_i) \equiv F_i \tag{2.4}$$

The method has the disadvantage that the interpolant may not reflect the real shape of the original function F, near the points (x_i, y_i). This can be seen by observing that for $q = 2$

$$\frac{\partial S}{\partial x}(x_i, y_i) = \frac{\partial S}{\partial y}(x_i, y_i) = 0 \tag{2.5}$$

These local extrema introduce flat zones in the 2D interpolating surface. On the other hand, S is infinitely differentiable (except at the points (x_i, y_i) where it is only continuous).

There are variants to the technique, such as the *Modified Shepard's Method* (MSM), developed by Franke and Nielson, which is discussed in [14] and [23]. This variation modifies the weighting function to take into account only the points lying in a disc with radius R, centered at the point of evaluation, (x, y). The modification improves the method both in addressing the preservation of the shape near the points (x_i, y_i), and in making it local to a neighborhood of points. This variation of the MSM proposes an interpolating function S_{mod} of the form

$$S_{mod}(x, y) = \sum_{i=1}^{n} Q_i(x, y) W_i(x, y) \tag{2.6}$$

where the functions Q_i are local approximations to the function F at the point (x_i, y_i). More precisely, Q_i is a quadratic function such that $Q_i(x_i, y_i) = F_i$. The weights are defined as

$$W_i(x, y) = \frac{w_i(x, y)}{\sum_{k=1}^{n} w_k(x, y)} \tag{2.7}$$

where the relative weights w_k are proportional to the inverse of the Euclidean distance function

$$w_k(x, y) = \left(\frac{(R - \|(x_k, y_k) - (x, y)\|_2)_+}{R\,\|(x_k, y_k) - (x, y)\|_2} \right)^2 \tag{2.8}$$

where

$$(a)_+ = \begin{cases} a & if\, a > 0 \\ 0 & otherwise \end{cases}$$

Here, R denotes the radius of influence around the point (x_k, y_k). Spth shows in [36], that for any $R > 0$

$$W_i(x_j, y_j) = \begin{cases} 1 & if\, i = j \\ 0 & otherwise \end{cases} \tag{2.9}$$

and therefore $S_{mod}(x, y)$ satisfies the interpolating condition

$$S_m(x_i, y_i) = Q_i(x_i, y_i) = F_i \tag{2.10}$$

Spth also shows in [36], that S_{mod} has quadratic precision, that is, if F is quadratic, $S_{mod}(x, y) = F(x, y)$ for any point (x, y).

ACM algorithm 790 (CSHEP2D) developed by Renka and Brown [30], is a variation of the MSM. This is one of the two methods we use to compare the performance of the technique we introduce in Chapter 3.

2.2 Multiquadrics

Another important method of interpolation is known as *Hardy's Multiquadrics.* In this global method, we consider an interpolating function $H(x, y)$ of n mesh points (x_k, y_k) that is a combination of basis functions B_k, defined by

$$H(x, y) = \sum_{i=1}^{n} a_i B_i(x, y) \tag{2.11}$$

where the a_k coefficients are determined from the interpolation requirements (1.2). There are many choices for the basis B_k, a popular choice are the radial basis functions:

$$B_k(x, y) = B_k(r_k),\ r_k = \|(x, y) - (x_k, y_k)\|_2 \tag{2.12}$$

Examples of such functions are

$$\begin{gathered} B_k(r_k) = (r_k^2 + R^2)^q\,,\ q, R \in \Re \\ B_k(r_k) = ln\,(r_k^2 + R^2) \end{gathered} \tag{2.13}$$

where R is a parameter (chosen by the user) that should not be too large. To find the corresponding interpolant, we must solve the system

$$B\underline{a} = \underline{f} \tag{2.14}$$

where the matrix B and the vector$\underline{f} = [F_1, F_2, ..., F_n]^T$ are known.

Franke, in [13], reports the superior performance of multiquadrics compared with other methods. For further details on multiquadrics, we refer the reader to Hardy's excellent survey [16].

2.3 Divide and Conquer: Cell Subdivision

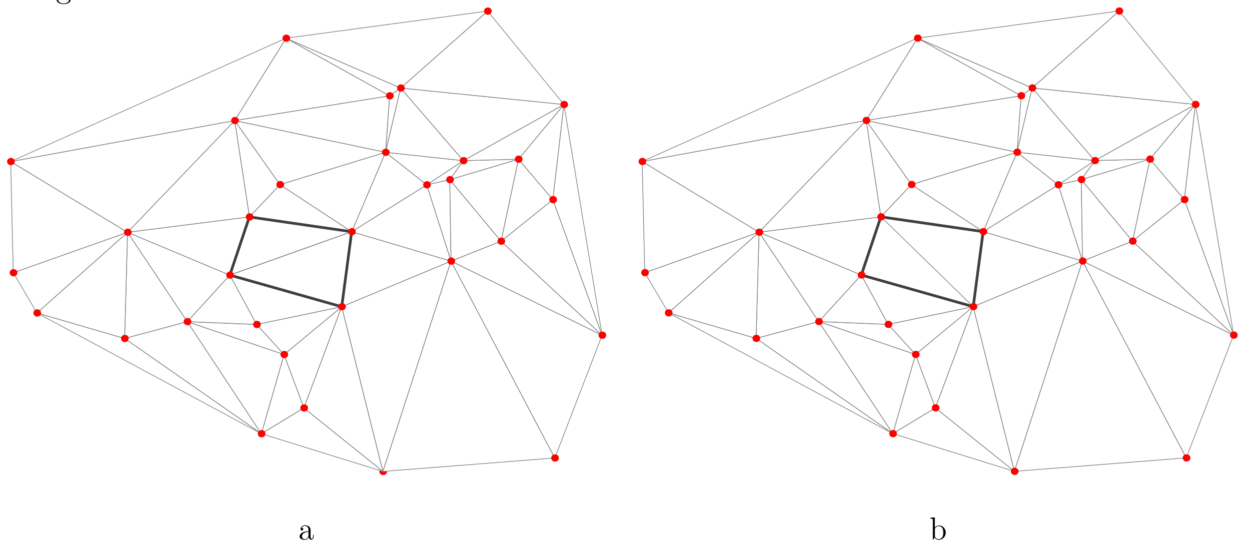
A popular way to construct an interpolant, is to consider it as a piecewise union of patches joined with certain continuity. This approach subdivides the domain into cells and, for each cell, determines a local interpolating function. This is the basis of finite element methods, where each cell is called an 'element' and a patch is a low degree multivariate polynomial. When the points (x_k, y_k) are regularly spaced over the domain, it is natural to define the cells as rectangles, but for scattered data is more common to subdivide the domain into triangles. The subdivision process is called triangularization, and the set of triangles produced is called a triangulation.

We define a triangulation over the plane as a set $T = \{T_k\}$, $T_k = (P_{\alpha_k}, P_{\beta_k}, P_{\gamma_k})$ where

- $P_i = (x_i, y_i)$ are the data points in $\Re^2$
- $\alpha_k, \beta_k, \gamma_k$ are distinct integers in the range $[1, n]$
- Each triangle T_k contains only 3 data points: the vertices of T_k
- The intersection between any two triangles T_k and T_j $(k \neq j)$ has area zero.
- The union of all the triangles T_k is equal to the convex hull of the points P_i .

Given a set of n points, there could be more than one triangulation that can be constructed. This is easy to see with four points defining a convex quadrilateral: Two triangulations are possible depending which diagonal is considered. There are several criterion we can use to choose between two possible triangulations. In the simple case of a quadrilateral, an easy test would be to choose the triangulation with the shortest diagonal. However this approach can produce long thin triangles, a feature that in some applications should be avoided (In the case of interpolation, long thin triangles may span a region where there is considerable variation in $F(x, y)$).

Figure 2.1: Two triangulations for the same set of points. In the center quadrilateral formed by the shaded lines, choosing any of the two possible diagonals defines a different triangulation.



A concept that helps to avoid long thin triangles is the max-min angle criterion: Let be $a(T_k)$ = minimum angle in the triangle T_k, and given a triangulation T, let $ma(T) = min\{a(T_k) : T_k \in T\}$. According to the max-min criterion, a triangulation T is better than another T' if $ma(T) > ma(T')$. A triangulation that is better than all other triangulations, according to this criteria, is called a Delaunay (or Thiessen) triangulation.

The classic Delaunay triangulations algorithms can be characterized as follows.

- Post Optimization Algorithm: This method starts with an arbitrary triangulation T, and considers all the convex quadrilaterals. It changes the corresponding diagonal if that leads to a better triangulation. This method runs in $O(n^2)$.
- Incremental Algorithm: This method starts with a single triangle and then adds one point at a time, updating the triangulation in such a way that it is still locally optimal. This algorithm runs in the worst case in $O(n^2)$.
- Divide and Conquer Algorithm : This method divides the data, finds the optimal triangulation for each of the two pieces and then merges the results. It is shown in

[37] that this algorithm runs in $O\left(n\,log\left(n\right)\right)$.

It is worth noticing that the computation of a Delaunay triangulation also finds the convex hull of a given set of points, a well known problem that has a minimal complexity of $O\left(n\,log\left(n\right)\right)$. For a more detailed review on Delaunay triangulations, the reader can consult [37].

2.4 Bezier Patches

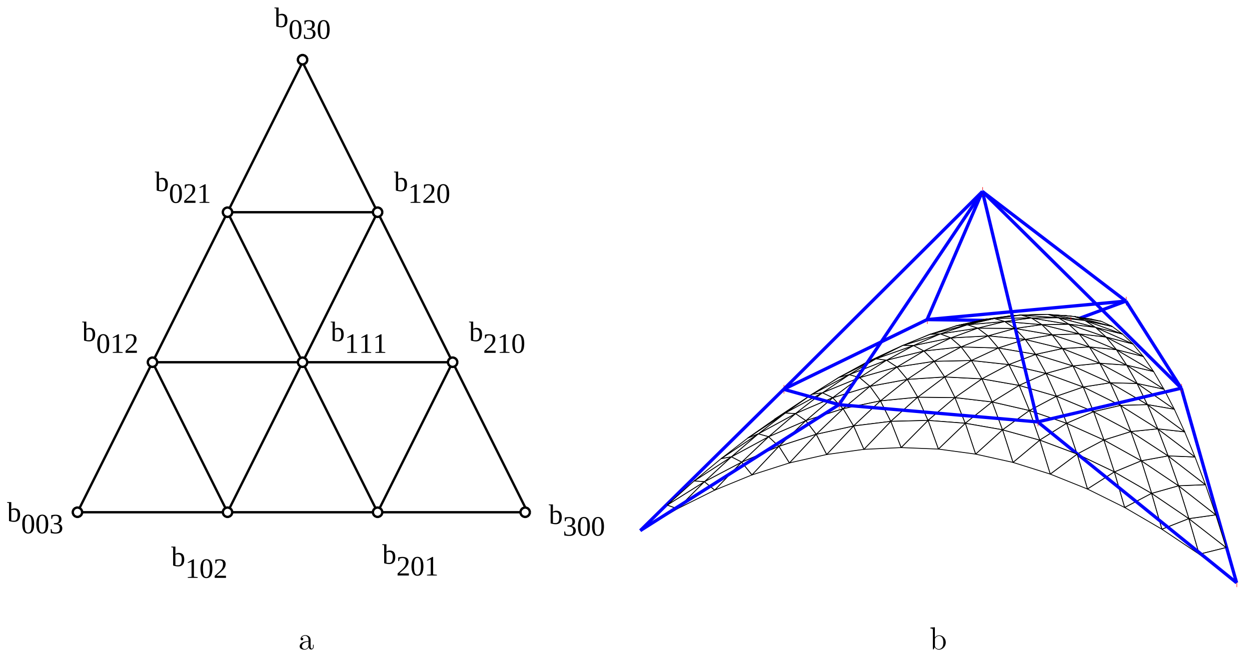
The one-dimensional interpolation problem is often addressed using a spline, which is a function composed of pieces of simpler functions defined on subintervals, connected at their endpoints with a certain level of continuity. There are many kinds of simpler functions that can be used to construct a spline, and in the case of polynomials (such as the piecewise cubic used by acm752), it is possible to express the spline using a Bezier formulation [3][10].

Bezier Triangles, or *Patches*, are a natural extension of the concept of Bezier curves, which are the result of the pioneering work of P. de Casteljau and P. Bezier. Bezier triangles are extensively used in the area of geometric design, and they are a tool that gives the user the freedom to model and change the shape of an object. An excellent description of Bezier curves and surfaces can be found in [10].

A Bezier triangle is defined by a set of control points that defines a net in 3D space. For the trivial case of a planar patch element (corresponding to a piecewise linear interpolant) the net consists of 3 control points, where for the cubic case we need 10 control points. In general, if we want to obtain a triangular surface patch of degree k, we need $\frac{1}{2}(k+1)(k+2)$ control points. This set of control points b_i is usually referred as the control net or mesh, and the projection of this mesh into the x-y plane is a triangle called the domain triangle.

Bezier Triangles are defined in terms of Bernstein polynomials B_I^k,

Figure 2.2: a) A triangular Bezier domain triangle for a patch of degree 3. b) A Triangular Bezier patch associated with the mesh in a).



$$b^k(U) = \sum_{|I|=k} b_I B_I^k(U) \tag{2.15}$$

where $U = (u, v, w)$ are the barycentric coordinates[1] of the patch with respect to the vertices of the Bezier triangle, and $I = (i, j, k)$ is a multindex, where $|I| = i + j + k$ (Figure 2.2).

For our interpolating problem, only the three vertices of the domain triangle are provided: we must determine the inner control points of the control mesh. This can be done if information about the gradient, or normal on the control points is given, leading to a formulation also known as Hermite interpolation, and it is described by Farin in [10]. When that information is not available, it is necessary to use estimation techniques to approximate the derivatives at the control points. Spth describes [36] several ways to estimate the gradient of the underlying function for a set of scattered mesh points.

[1]Given three different points A, B, and C we can write any point p in the plane defined by A, B, and C as $p = uA + vB + wC$ where $u + v + w = 1$

Some important properties of Bezier objects include:

- Endpoint interpolation: Bezier triangles 'touch' the three outermost vertices of the control mesh. This can be verified directly from the properties of Bernstein polynomials and (2.15).

- Convex Hull: From (2.15) also follows that the object is contained in the convex hull defined by the control points. This follows since the Bernstein polynomials are positive and they form a partition of unity.

- The boundaries of Bezier objects are also Bezier objects of lower dimension. In particular the boundaries of a Bezier triangle are three Bezier curves, each of which is determined by the control points over the boundaries. This make Bezier interpolants C^0 continuous.

Farin shows in [10] that two triangular patches are C^1 if the adjacent sub-triangles in the control meshes are coplanar and an affine map of the domain triangles, as shown in Figure 2.3.

Two triangular patches can be made C^1 continuous by adjusting their inner control points, so they make the adjacent sub-triangles coplanar. The vertex consistency problem, however, shows that it is not always possible to obtain a smooth global (C^1 or G^1) interpolant [27] (Figure 2.4).

Split-triangle schemes are one way to overcome this problem. Since there are more continuity restrictions than control points we can arbitrarily change, one solution is to subdivide the triangular patch into smaller triangles (mini-triangles) that will introduce more degrees of freedom (control points). As a result, the triangular patch will be a piecewise patch composed of sub-patches defined over the mini-triangles. We will describe one of those subdivision techniques in the next section.

Figure 2.3: Two adjacent Bezier patches with C^1 continuity. The highlighted sub-triangles from both control meshes must be coplanar, and an affine map of the domain triangle they belong to.

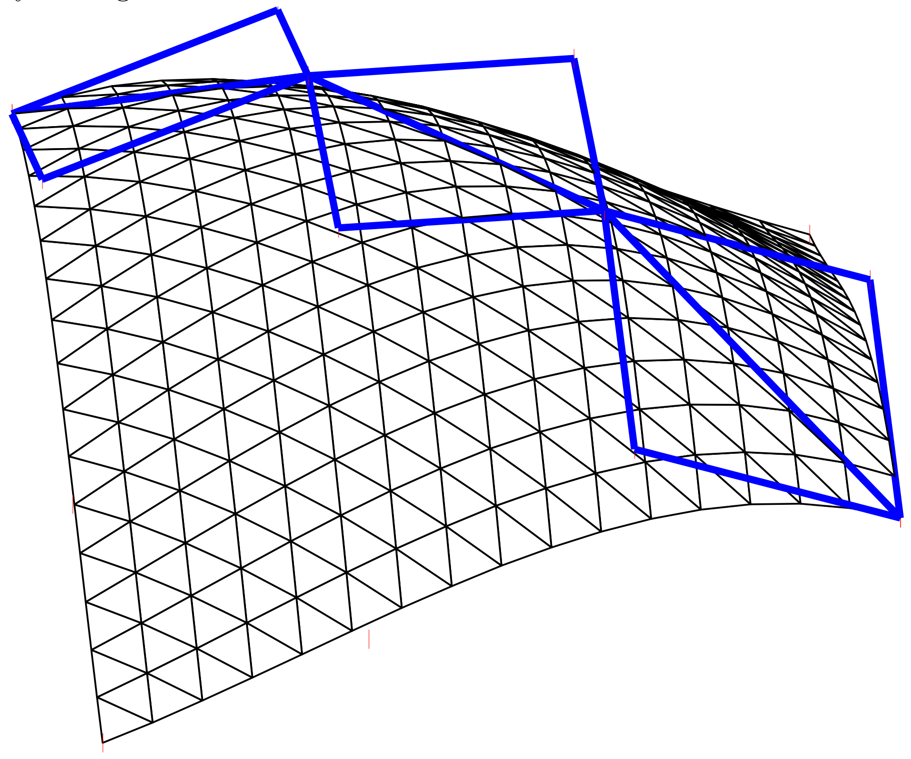


Figure 2.4: The Vertex Consistency Problem: for the center and rightmost triangles to be C^1, p must be fixed. But that breaks the continuity between the center and left triangles.

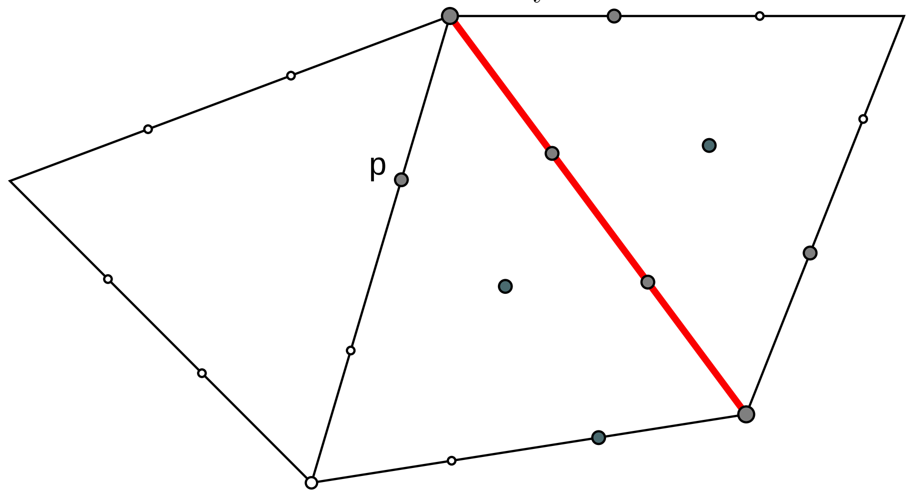
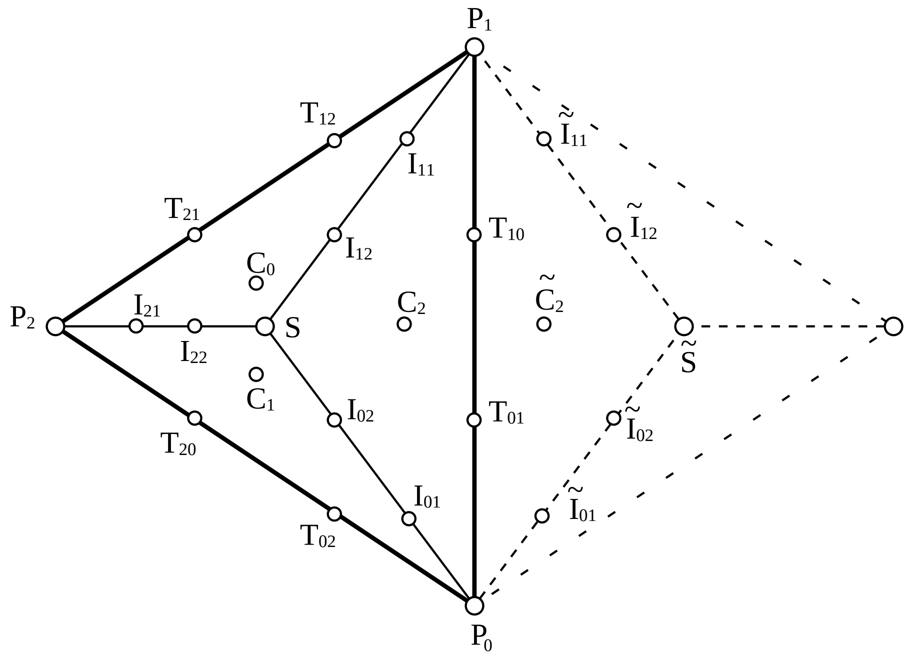


Figure 2.5: Two triangles subdivided using the Clough-Tocher criteria. Both triangles share the edge P_0P_1.



2.4.1 The Clough-Tocher Interpolant.

The Clough-Tocher interpolant was developed initially in the area of finite element analysis, and it is considered one of the simplest of the split-triangle interpolation techniques. It subdivides a macro-triangle T, into three mini-triangles by adding an internal extra mesh point, S, that refines the mesh, as it is shown for triangle $P_0P_1P_2$ in Figure 2.5. The method assumes that for every point (x_k, y_k) in the triangulation with corresponding data values F_k, we also have the corresponding value (or estimate) of the normal to the underlying function N_k .

The Clough-Tocher interpolant will produce a cubic patch for each mini-triangle, that will be connected to their internal and external neighbors with C^1 continuity. An algorithm described by Mann [21], that computes the values for the control points in Figure 2.5[2] is presented in Algorithm 1. The interpolant obtained in this way has quadratic

[2]Since the control points are in 3D space we will denote them as (P, z) where P is a 2D domain point and z is a scalar.

precision. Mann also discusses how to construct a Clough-Tocher interpolant with cubic precision.

Algorithm 1 Algorithm to compute the control points for the Clough-Tocher subdivision.

```
1: for values $ijk \in \{012, 120, 201\}$ do
2:     $V_i = (P_i, F_i)$
3:     Let $a_{ij} = \frac{2}{3}P_i + \frac{1}{3}P_j$ and $z_i(P)$ denote the equation of the tangent plane at $V_i$ .
       Then $T_{ij} = (a_{ij}, z_i(a_{ij}))$
4:     $I_{i1} = \frac{1}{3}(V_i + T_{ij} + T_{ik})$. After this stage two out of the three pairs of triangles (see
       Figure 2.5) are made coplanar.
5:     Set $C_i$ to be coplanar with $T_{jk}$, $T_{kj}$ and the opposite $C$ on the other side of the
       $V_jV_k$ edge. This is achieved prescribing some cross boundary derivative at each
       edge in the subdivision. This derivative could be in any direction with exception
       of the parallel to its corresponding edge.
6:     Choose $I_{i2}$ to lie in the plane defined by $C_j$, $C_k$ and $I_{i1}$
7: end for
8: $S = \frac{1}{3}(I_{02} + I_{12} + I_{22})$
```

ACM Algorithm 752, by Renka and Cline [32], uses the Clough-Tocher subdivision to construct a cubic C^1 interpolant over a triangulation. This algorithm is also used to compare the performance of the new approach described in the next chapter.

Chapter 3

DEI: The Differential Equation Interpolant.

3.1 Description

For most Partial Differential Equations (PDE) it is not possible to obtain a closed form expression for the solution, so it is common to apply a numerical method that produces an approximate solution at certain points P_i in the domain of interest. It is also generally the case that the points, where the solution is approximated, are distributed over a non-uniform mesh that adapts itself to the local behavior of the solution. In [9], Enright develops an approach, called Differential Equation Interpolant (DEI), that efficiently approximates the values of the solution of a PDE at off-mesh points. The method is such that the precision at the computed off-mesh points has the same order of accuracy as the solution obtained at the mesh points P_i. The interpolation scheme assumes that the underlying solution u satisfies a known two-dimensional, second order PDE of the form

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

where L is a differential operator defined by

$$L = a_1(x,y)u_{xx} + a_2(x,y)u_{yy} + a_3(x,y)u_{xy} \tag{3.2}$$

The approach assumes that an accurate approximation to the gradient of u at the points $P_i = (x_i, y_i)$ is also known. The interpolant is defined as a piecewise surface, where each triangle element e in the domain is associated with a bivariate polynomial

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

The variables s and t correspond to a mapping of x and y into a unit square where the domain triangle is contained, as illustrated in Figure 3.1. The mapping is defined by $s = \frac{(x-x_0)}{D_1}$, $t = \frac{(y-y_0)}{D_2}$, where the vertex $P_0 = (x_0, y_0)$ denotes the origin of the unit square $s = 0$, $t = 0$, and D_1, D_2 are appropriate scaling factors that depend on the dimensions of e. To characterize $p_{d,e}$ we need to find the coefficients c_{ij}. For the case we are interested in, where the patch defined over e is cubic ($d = 3$), we need to find 16 coefficients. We do this by specifying 16 linear equations associated with the coefficients c_{ij}. We determine the first nine equations by imposing the following linear constraints at each of the three points (x_k, y_k) defining e:

$$\sum_{i=0}^{3}\sum_{j=0}^{3} c_{ij}s_k^i t_k^j = F_k \tag{3.4}$$

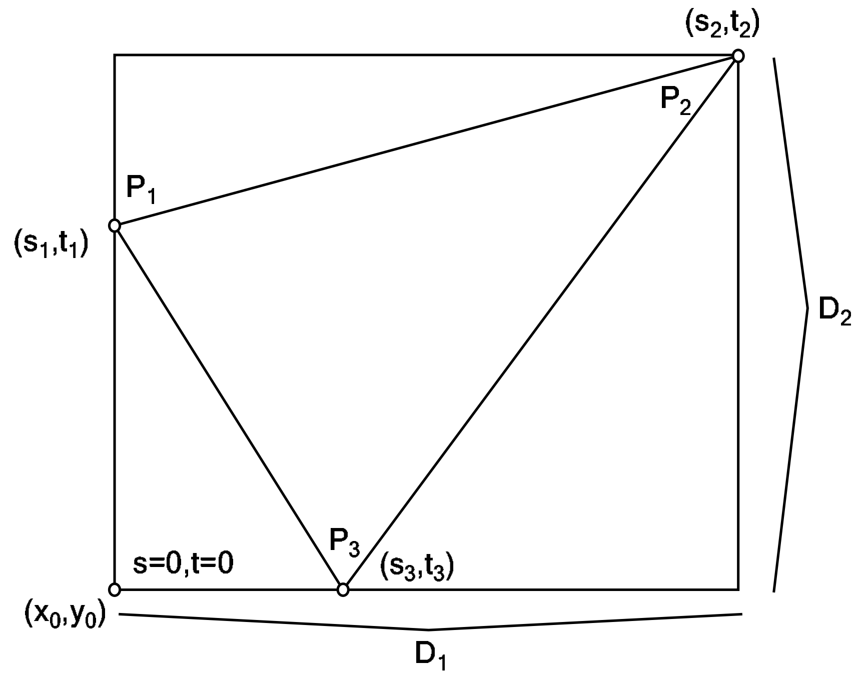
$$\sum_{i=0}^{3}\sum_{j=0}^{3} ic_{ij}s_k^{i-1} t_k^j = D_1\frac{\partial F_k}{\partial x} \tag{3.5}$$

$$\sum_{i=0}^{3}\sum_{j=0}^{3} jc_{ij}s_k^{i} t_k^{j-1} = D_2\frac{\partial F_k}{\partial y} \tag{3.6}$$

where $s_k = \frac{(x_k-x_0)}{D_1}$, $t_k = \frac{(y_k-y_0)}{D_2}$. We construct the remaining seven linear equations by choosing seven 'collocation points' inside the triangle and imposing a condition that the interpolant 'almost' satisfy Equation (3.1) at the chosen points.

From (3.3) we have

Figure 3.1: A triangle cell inscribed in a rectangle of dimensions D_1 by D_2. Mapping the triangle into the square makes s and t defined on $[0, 1]$.



$$\frac{\partial^2 p_{3,e}(x,y)}{\partial x^2} = \frac{1}{D_1^2}\sum_{i=0}^{3}\sum_{j=0}^{3} i(i-1)c_{ij}\left(\frac{x-x_0}{D_1}\right)^{i-2}\left(\frac{y-y_0}{D_2}\right)^{j} \tag{3.7}$$

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

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

and therefore, for a fixed value of x and y, $Lp_{3,e}(x, y)$ is a linear combination of the coefficients c_{ij}. Then, for a collocation point $(\widehat{x}, \widehat{y})$ we impose a linear constraint of the form

$$Lp_{3,e}(\widehat{x}, \widehat{y}) = g(\widehat{x}, \widehat{y}, \widehat{F}, \widehat{F_x}, \widehat{F_y}) \tag{3.10}$$

where $\widehat{F}, \widehat{F_x}$ and $\widehat{F_y}$ are approximations (actually, the piecewise linear approximation associated with the data from P_1, P_2, P_3) to the actual values of F, $\frac{\partial F(x,y)}{\partial x}$ and

$\frac{\partial F(x,y)}{\partial y}$respectively. To determine the coefficients c_{ij}, we use the system of linear equations formed by nine linear interpolation constraints, (3.4-3.6), and linear constraints corresponding to seven collocation points, (3.10). This system can be expressed as

$$Wc = b \tag{3.11}$$

where $c = \{c_{ij}\}$, and W is a matrix that depends only on the mesh points, the collocation points and the definition of L. The vector b depends on the mesh data and the approximations $\widehat{F}, \widehat{F_x}$ and $\widehat{F_y}$ associated with the collocation points. To ensure that there is a solution the collocation points must be chosen such that W is nonsingular.

Although the approach has an accuracy comparable with the numerical method that provided the data, it does not generate necessarily a continuous piecewise interpolant. In particular, for some choice of collocation points, this can be observed in the form of oscillations in the approximation near the boundaries of the triangles e, due to the fact that on the boundaries, the function $p_{d,e}$ may well be a high degree polynomial. On the edge between mesh points (s_i, t_i) and (s_j, t_j), the interpolating polynomial can be expressed as

$$p_{d,e}(q) = p_{d,e}((s_i, t_i) + q(ds, dt)) \tag{3.12}$$

where $ds = s_j - s_i$ and $dt = t_j - t_i$. From (3.3), we see that (3.12) is a polynomial of degree six in q, however, in the case where the boundaries are parallel to the x or y axes, (3.12) is reduced to a cubic curve uniquely defined by the mesh data on the edges , and this makes the interpolant C^0. For the purposes of visualization, it is desirable to attenuate the discontinuities as much as possible, because that will produce smoother surfaces. For that, we develop a test that rejects choices of collocation points that produce noticeable discontinuities among patches. Analyzing several examples, we observe that patches with non-negligible discontinuities have associated coefficients, c_{ij}, which are larger than expected in magnitude. This behavior is reflected in the norm of the vector

c (3.11) associated with the patch. The magnitude of the norm is reflected by the value $norm(c)$:

$$norm(c) = \sum_{i,j \in [0,3]} (c_{ij})^2/16 \qquad (3.13)$$

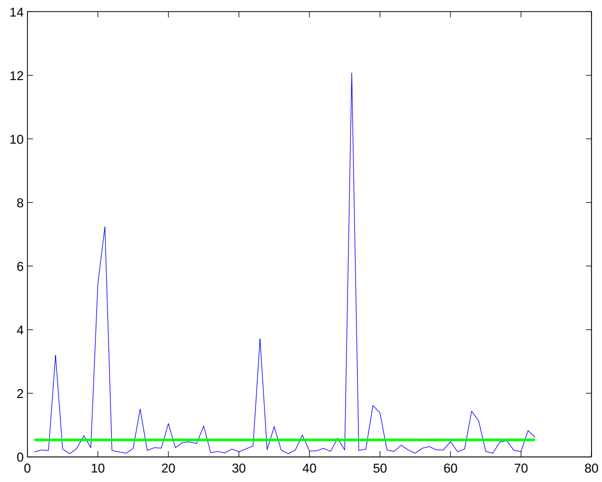
We observed that the noticeable discontinuities correspond to patches with large values of $norm(c)$, and this is illustrated in Figure 3.2-a. To attenuate the discontinuities, we developed an algorithm that detects the offending patches (those with large $norm(c)$), and decreases the associated $norm(c)$ by choosing alternate collocation points, and therefore a different vector c.

the algorithm computes in its main iteration a threshold using the norms of the vectors c from all patches [1]. This threshold, scaled by a coefficient $\alpha > 1$, is used to identify those elements (patches) where we must consider alternate collocation points. The algorithm continues until all the norms are below the threshold or no more improvements can be made, this last criteria is controlled by a parameter, $MAXTRIES$, that states how many choices of collocation points per patch the algorithm can try, before it decides it can't decrease the norm. The pseudo-code in Algorithm 2 describes this process.

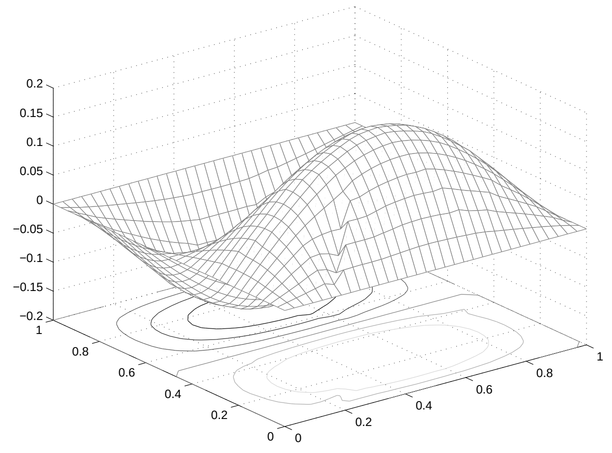
The choice of the values α and $MAXTRIES$ affects the performance of the algorithm, both in how long it takes to find a suitable set of collocation points, and how how much the discontinuities are attenuated. Although the best choices for these values will depend on the function, F, and the location of the scattered data, we observed that overall good results were obtained for $\alpha = 1.5$ and $MAXTRIES = 100$. We call the interpolant defined with this set of alternate collocation points, the Alternate Differential Equation Interpolant (ADEI).

[1]For our calculations, we actually discard the 10% largest norms to avoid the influence of the spikes.

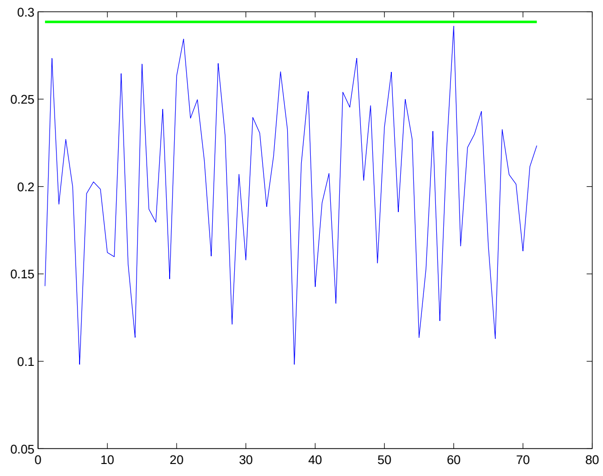
Figure 3.2: Magnitude of the norm for the coefficients c_{ij} for a 6x6 regular mesh (72 triangles); a) shows the norms for the initial choice of collocation points, the horizontal line is the threshold value used to decide when a patch must be re-calculated; b) the interpolant obtained with the initial choice, where discontinuities can be seen; c) the result after the application of the algorithm from Table 2 (notice the change of scale). In this last case all patches have their norm below the threshold. The final interpolant is shown in d).



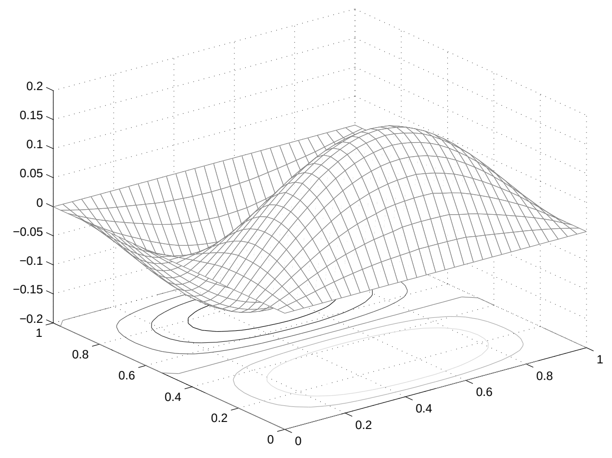
a



b



c



d

Algorithm 2 Algorithm to find a set of coefficients c_{ij} that will attenuate the oscillations in the boundaries of the triangles e.

```
for every $e$ do
  Compute an initial set $c = \{c_{ij}\}$
end for
$success = true$
loop
  Compute the norm vector: $nc(e) = norm(c)$
  Compute threshold: $thresh = \alpha \cdot mean_{90\%}(nc)$
  Compute set of offending triangles: $T = \{e : nc(e) > thresh\}$
  if $T = \emptyset$ or $\neg success$ then
    FINISH
  end if
  $success = false$
  for each $e \in T$ do
    $times = 0$
    $PartialSuccess = false$
    repeat
      $times = times + 1$
      Choose alternate set of collocation points for $e$
      Compute $\widetilde{c} = \{\widetilde{c_{ij}}\}$
      if $norm(\widetilde{c}) < nc(e)$ then
        Update coefficient with better values: $c = \widetilde{c}$
        $success = true$
        $PartialSuccess = true$
      end if
    until $PartialSuccess$ or $times = MAXTRIES$
  end for
end loop
```

3.2 The Test: Functions and Data.

The purpose of this section is to describe the procedure used to measure the performance of ADEI, and how effective it can be relative to other widely used methods. For the comparison, we chose two ACM algorithms for the interpolation of scattered data, algorithms 752 (acm752) and 790 (acm790). We believe that the accuracy tests and comparisons presented by Renka in [26], introduce a common framework or benchmark to perform evaluations of new interpolation methods. The tests were performed on three functions: F_1 from [9], were it is used to perform a preliminary accuracy test, F_2 from the test suite used by Renka in [26], and F_3 which is a function with less variation than the previous two, that serves to evaluate the performance of the methods on functions without significant features. Surface plots of each test functions are shown in Figure 3.3. Although Algorithm 752 can perform an estimation of the gradients at the mesh points, we make the comparison as fair as possible, by giving Algorithm 752 the gradients in the same way that they were provided to ADEI. Because the routines in Algorithm 790 do not allow a direct input of the gradients, they were not used with this algorithm.

The test functions are

$$F_1(x,y) = cos(10y) + sin(10(x-y))$$

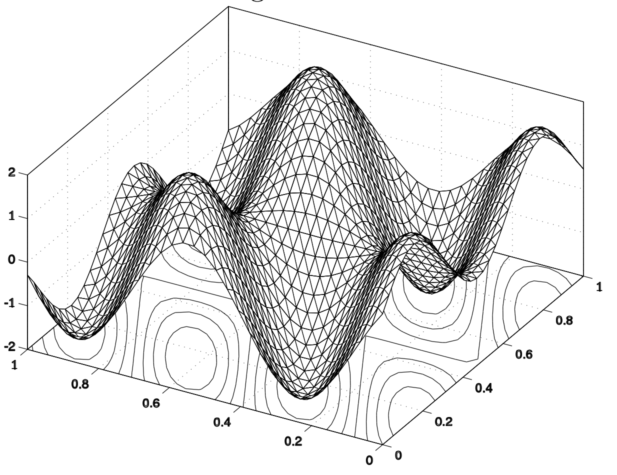
$$F_2(x,y) = e^{(-\frac{(5-10x)^2}{2})} + 0.75e^{(-\frac{(5-10y)^2}{2})} + 0.75e^{(-\frac{(5-10x)^2}{2})}e^{(-\frac{(5-10y)^2}{2})}$$

$$F_3(x,y) = sin(2\pi y) * sin(\pi x)$$

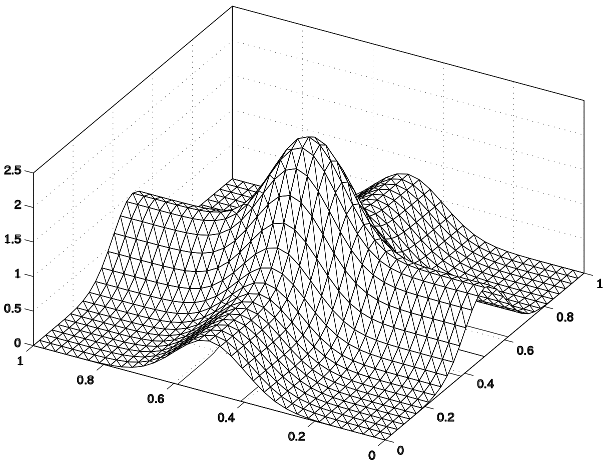
For ADEI, we must introduce a differential operator, L, for each test function. We used $Lu = u_{xx}+u_{yy}$, and the corresponding functions g_i such that $LF_i = g_i(x,y,u,u_x,u_y)$. The corresponding function g_i[2] are:

[2]For g_1, we choose f to ensure that the true solution agrees with the specified PDE boundary conditions.

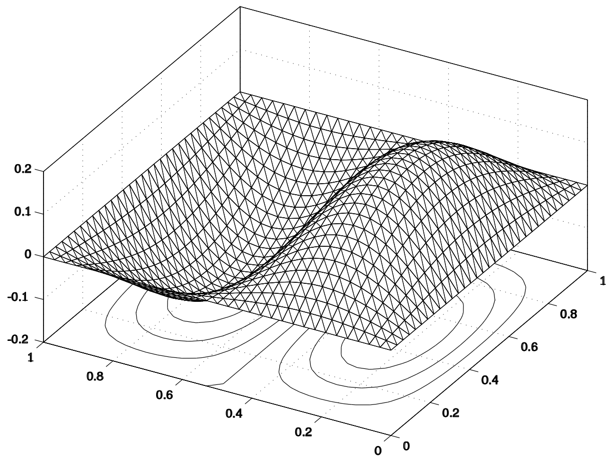
Figure 3.3: Test functions F_1 (a), F_2 (b), and F_3 (c).



a



b



c

$$g_1(x,y,u,u_x,u_y) = cos(10y)u - (1+sin(10x))u_x + f$$

$$g_2(x,y,u,u_x,u_y) = -100e^{(-\frac{(5-10x)^2}{2})} + (50-100x)u_x - 150e^{(-\frac{(5-10x)^2}{2})}e^{(-\frac{(5-10y)^2}{2})}$$
$$-75e^{(-\frac{(5-10y)^2}{2})} + (50-100y)u_y$$

$$g_3(x,y,u,u_x,u_y) = (-\pi^2 - 4\pi^2)u$$

The data is divided into three classes or node sets: truly random samples, adaptive random samples, and regular samples. The truly random class (Node Set II) consists of three sets of 100 randomly chosen points referred to in this paper as Data Sets III, IV, and V. The adaptively random class (Node Set I) has for every test function, two data sets of 100 points, referred as Data Sets I and II. The purpose of this class is to consider the more realistic case where the points are still randomly generated but with a distribution that adapts to the behavior of the function. The third class (Node Set III) is a regular grid that divides the unit square into an 8 by 8 subdivision (81 mesh points), referred as Data Set VI. Also, to summarize the results, we group the functions into two classes: Those with many features (F_1 and F_2), and without many features (F_3).

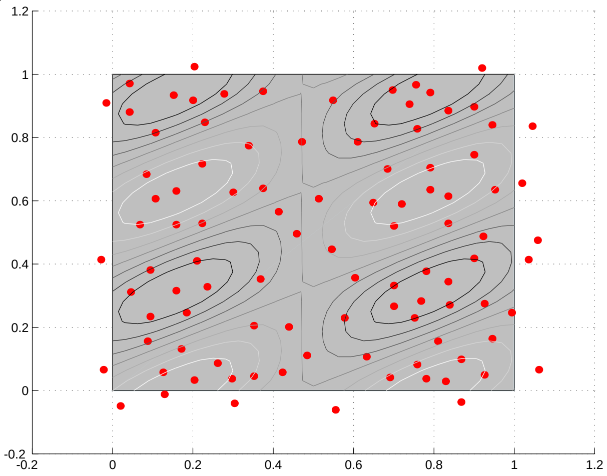
All random points are generated in the square [-0.1,1.1]×[-0.1,1.1], making most of the thin triangles of the associated triangulation lie outside the unit square [0,1]×[0,1], where all the error measurements are taken. Some representative data sets are shown in Figure 3.4.

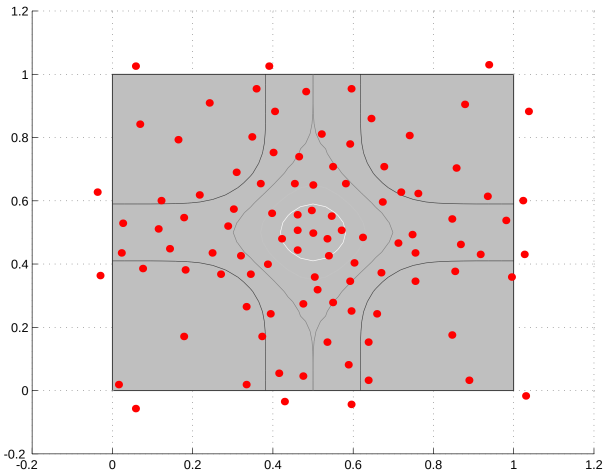
We apply the error measure used by Renka and Brown in [26] to compare a function u and its approximation $\widetilde{u}$. It is defined by

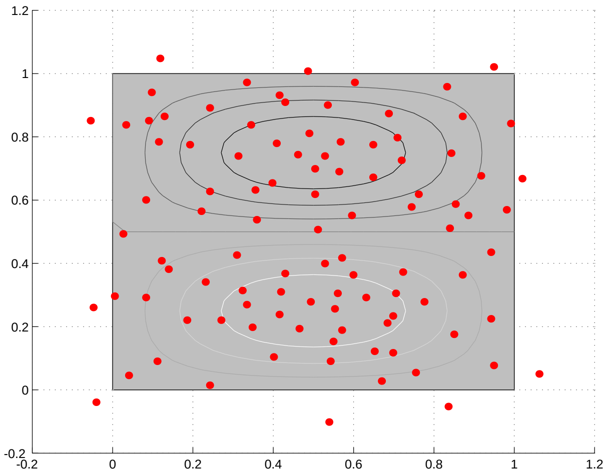
$$error(u,\widetilde{u}) = \frac{\sum_{p \in NodeSet}(u(p) - \widetilde{u}(p))^2}{\sum_{p \in NodeSet}(u(p) - \overline{u})^2} = \frac{SSE}{SSM} \tag{3.14}$$

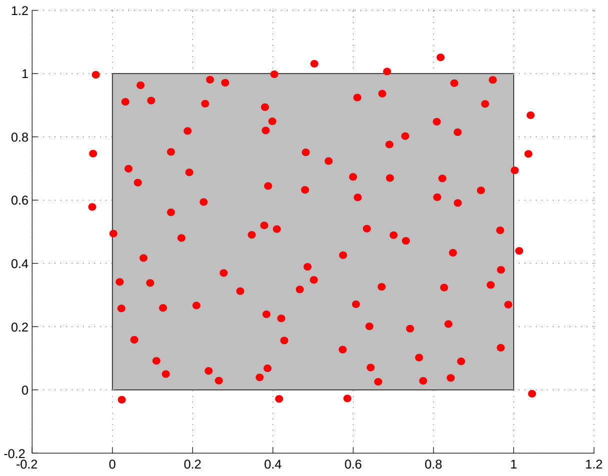
where $NodeSet$ are the points from a 33 by 33 uniform grid defined on the unit square, that lie in the convex hull (CH) defined by a Node Set (I, II, or III). In other words, we

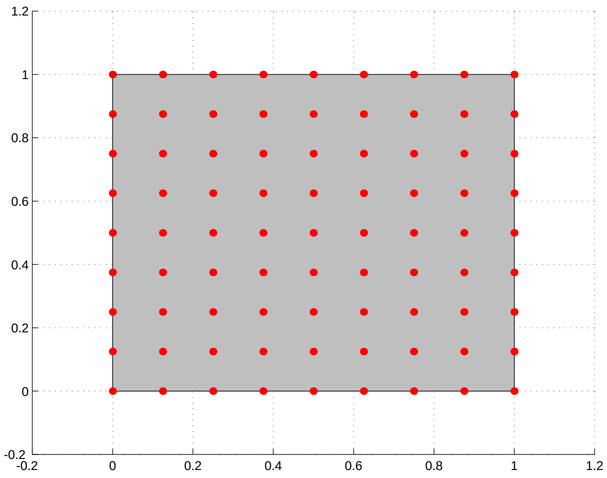
Figure 3.4: Some representative Data Sets. From Node Set I: a) 100 adaptive random points for F_1, b) 100 adaptive random points for F_2, c) 100 adaptive random points for F_3. From Data Set II: d) 100 truly random points . From Data Set III: e) 8 by 8 uniform grid.


a


b


c


d


e

are explicitly not considering extrapolation in assessing the methods. The error defined in this way is the quotient of the sum of squared errors (SSE) and the sum of squares of deviations of u from its mean $\overline{u}$ (SSM).

To summarize the results given by the error measures, and present them in a more compact form we compute the values $r^2 = 1 - Avg(error)$. In statistics, the magnitude r^2 is called the *coefficient of determination*. Its interpretation, as discussed in [26], is summarized in Table 4.1.

Chapter 4

Results

Tables 4.3 through 4.8 show the error norms for the different Node Sets[1]. In each table, the first column shows the performance of DEI, i.e., with the first selection of collocation points. Because this choice is random, the results shown in that column may vary if the test is performed again. The second column shows the error with the set of collocation point adjusted using the algorithm described in Chapter 3: the improvement over the non-optimized version is noticeable, and illustrates the sensitivity of the method to the choice of collocation points. The tables contain a row labeled Avg., with the error averaged over functions F_1 and F_2 (we don't include F_3 in the average because of the difference in features with respect to F_1 and F_2).

We also compute for a given a set of values of r^2 the expected fit for a particular interpolation method. Table 4.2 shows how we can categorize by numbers the magnitude r^2 by finding in which sub-interval of $(0, 1)$ it lies in. Then for a given set of values r^2 it is possible to count how many of them belong to each fitness level (interval), and by doing so we can obtain the relative frequency $rf(i)$ for a particular fitness level i. We define the expected fit as

[1]Note that for each function, the distribution of points in Node Set I and II is different, because it adapts to the shape of the function.

Table 4.1: The parameter r^2 can be seen as being obtained from a least-squares fit from a constant function to the data. The interpretation associated with its value is summarized in this table.

r^2	Interpretation
0	no accuracy
0.9	fair fit
0.95	good fit
0.99	very good fit
0.999	excellent fit
0.9999	almost perfect fit (negligible error in empirical data)

Table 4.2: Fitness levels intervals. A value r^2 belongs to one of these 5 categories

Interval	Fitness Level	Fit
0-0.9	0	No
0.9-0.95	1	Fair
0.95-0.99	2	Good
0.99-0.999	3	Very Good
0.999-0.9999	4	Excellent
0.9999-1	5	Almost Perfect

$$ExpectedFit = \sum_{i=0,5} rf(i)i \tag{4.1}$$

We observe that for the functions with many features (F_1 and F_2), ADEI has better accuracy that acm752 (even with the supplied gradients) and acm790 for Node Sets I,II,IV, and V. For Node Sets III and VI, ADEI is comparable to acm752, both showing excellent to almost perfect fit. Figure 4.1 shows the expected fit of the methods for the functions with many features.

For function F_3, all the methods exhibit an excellent to almost perfect fit to this simple function, 100 scattered mesh points appear to be more than enough for the interpolation task. Again, the two methods with higher accuracy in this case are ADEI and acm752. Figure 4.2 shows the expected fit of the methods for F_3.

Tables 4.9 through 4.11 show the data grouped by method instead of Node Set. In this case, the columns labeled Avg. have the error averaged over each function, with distinction among the Data Sets. This measures how good a Data Set is for a particular

Table 4.3: Error norms for Algorithms on Node Set I - Adaptive random

Function	DEI	ADEI	ACM 752	ACM 790	#points
F_1	0.4589022399	0.0145220916	0.0380224025	0.0237643860	1069
F_2	0.0526024338	0.0023761940	0.0010418790	0.0085815570	1071
Avg.	—	0.0084491428	0.0195321407	0.0161729715	—
F_3	0.0019640597	0.0000400020	0.0000300440	0.0002530472	1072

Table 4.4: Error norms for Algorithms on Node Set II - Adaptive random

Function	DEI	ADEI	ACM 752	ACM 790	#points
F_1	0.0050865181	0.0003796216	0.0084194107	0.0074587876	1082
F_2	0.0712914085	0.0009033367	0.0002178276	0.0050332888	1061
Avg.	—	0.0006414791	0.0043186192	0.0062460382	—
F_3	0.0729033447	0.0000087406	0.0000749690	0.0001282769	1062

method/function combination.

ADEI has the better accuracy when interpolating F_1, where it always generates an excellent fit for adaptive, truly random, and uniform meshes. On the other hand, for F_2 the best method is acm752, with ADEI being comparable for the random sets only. For F_3 all the methods exhibit excellent to almost perfect fitting characteristics, again, when ADEI is not the best, it is comparable to acm752. Figures 4.3, 4.4, and 4.5, show the expected fit over Data Sets I-III, for functions F_1, F_2, and F_3 respectively.

It is clear that acm790 didn't perform comparably to the other two methods because no gradient data was supplied to it. We also observed that acm752 improved its performance using true gradient information instead of gradient estimation techniques. This kind of information when accessible proves to be important when constructing inter-

Table 4.5: Error norms over 1085 grid points for Algorithms on Node Set III - Truly random

Function	DEI	ADEI	ACM 752	ACM 790
F_1	0.0693021328	0.0015187244	0.0045488267	0.0195790931
F_2	0.4895293851	0.0059274472	0.0029519236	0.0204049517
Avg.	—	0.0037230858	0.0037503752	0.0199920224
F_3	0.0092975213	0.0004102222	0.0002288046	0.0004842083

Table 4.6: Error norms over 1052 grid points for Algorithms on Node Set IV - Truly random

Function	DEI	ADEI	ACM 752	ACM 790
F_1	0.4291720175	0.0012573059	0.0417162181	0.0447948512
F_2	0.5381581199	0.0053618095	0.0047226946	0.0154283581
Avg.	—	0.0033095577	0.0232194563	0.0301116047
F_3	0.0001214650	0.0000416803	0.0003432168	0.0004287497

Table 4.7: Error norms over 1076 grid points for Algorithms on Node Set V - Truly random

Function	DEI	ADEI	ACM 752	ACM 790
F_1	0.4444841239	0.0006805438	0.0172977341	0.0134447750
F_2	0.9600867117	0.0014902458	0.0041406490	0.0085123544
Avg.	—	0.0010853948	0.0107191915	0.0109785647
F_3	0.0000733812	0.0000101496	0.0000239087	0.0000693743

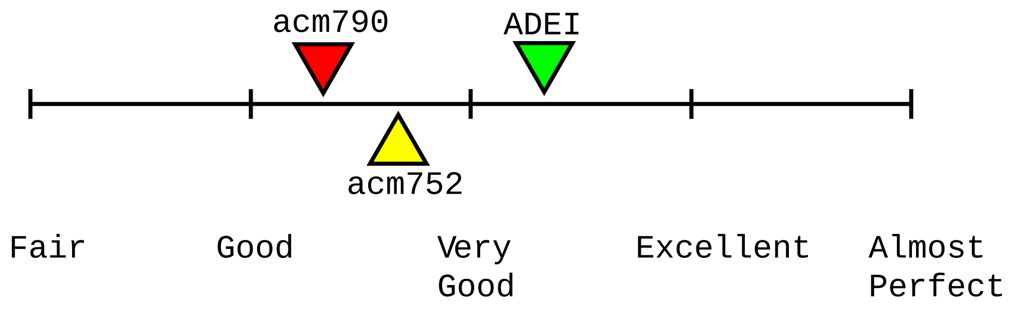
Table 4.8: Error norms over 1089 grid points for Algorithms on Node Set VI - Uniform 8 by 8 grid

Function	DEI	ADEI	ACM 752	ACM 790
F_1	0.0377498821	0.0004255222	0.0006966450	0.0018688849
F_2	0.0323203149	0.0007703701	0.0001219202	0.0018598887
Avg.	—	0.0005979462	0.0004092826	0.0018643868
F_3	0.0021250237	0.0000059197	0.0000039042	0.0000460145

Table 4.9: Error norm for ADEI on functions F_1 through F_3.

	F_1	F_2	F_3
Node Set I	0.0145220916	0.0023761940	0.0000400020
Node Set II	0.0003796216	0.0009033367	0.0000087406
Avg.	0.0074508566	0.0016397653	0.0000243713
Node Set III	0.0015187244	0.0059274472	0.0004102222
Node Set IV	0.0012573059	0.0053618095	0.0000416803
Node Set V	0.0006805438	0.0014902458	0.0000101496
Avg.	0.0011521913	0.0042598341	0.0001540173
Node Set VI	0.0004255222	0.0007703701	0.0000059197

Figure 4.1: Expected fit over functions F_1, F_2 on node sets I-VI.



polants.

The cost of evaluating the interpolant in acm790 is different than in ADEI and acm752, although all methods share an initial stage where they must find the cell (or triangle[2]) that contains the point to be evaluated. The final evaluation on ADEI and on acm752 requires the evaluation of a cubic polynomial, however, the final evaluation in acm790 involves several evaluations of cubic polynomials, one for each mesh point in a disc centered in the evaluation point, making the evaluation not as fast as the methods based on triangulations. The optimization algorithm described in Table 2 makes the estimation of the overall cost of ADEI difficult, because we can't tell in advance how many times alternate collocation points have to be chosen, or how many patches will present non-negligible discontinuities.

[2]The expected cost of locating a point in a triangulation is $O(log(n))$ [8], whereas for the cell method on acm790 is $O(1)$ with a $O(n)$ worst case [29].

Table 4.10: Error norm for acm752 on functions F_1 through F_3.

	F_1	F_2	F_3
Node Set I	0.0380224025	0.0010418790	0.0000300440
Node Set II	0.0084194107	0.0002178276	0.0000749690
Avg.	0.0232209066	0.0006298533	0.0000525065
Node Set III	0.0045488267	0.0029519236	0.0002288046
Node Set IV	0.0417162181	0.0047226946	0.0003432168
Node Set V	0.0172977341	0.0041406490	0.0000239087
Avg.	0.0198080581	0.0039384224	0.0001986433
Node Set VI	0.0006966450	0.0001219202	0.0000039042

Table 4.11: Error norm for acm790 on functions F_1 through F_3.

	F_1	F_2	F_3
Node Set I	0.0237643860	0.0085815570	0.0002530472
Node Set II	0.0074587876	0.0050332888	0.0001282769
Avg.	0.0156115868	0.0068074229	0.0026431680
Node Set III	0.0195790931	0.0204049517	0.0004842083
Node Set IV	0.0447948512	0.0154283581	0.0004287497
Node Set V	0.0134447750	0.0085123544	0.0000693743
Avg.	0.0259395731	0.0245707191	0.0003274441
Node Set VI	0.0018688849	0.0018598887	0.0000460145

Figure 4.2: Expected fit over function F_3 on node sets I-VI.

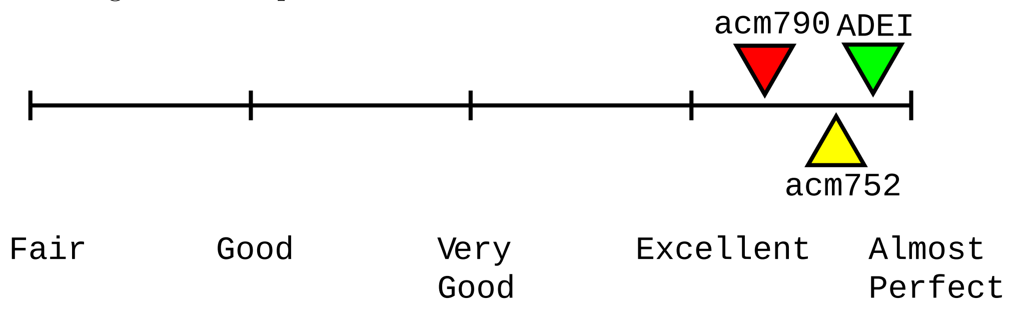


Figure 4.3: Expected fit over Data Sets I-III, on function F_1.

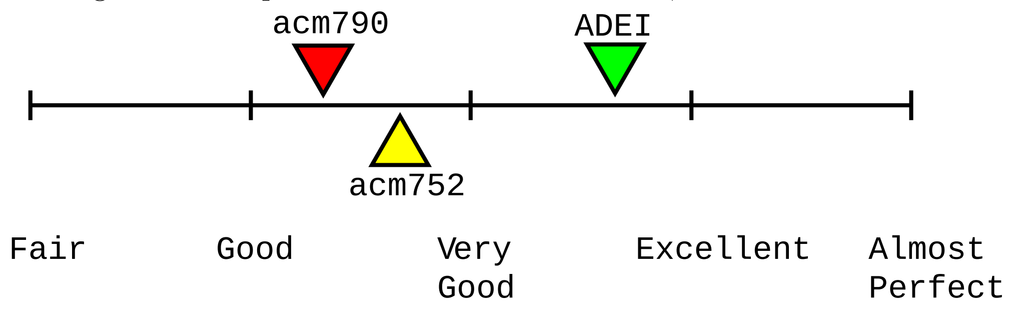


Figure 4.4: Expected fit over Data Sets I-III, on function F_2.

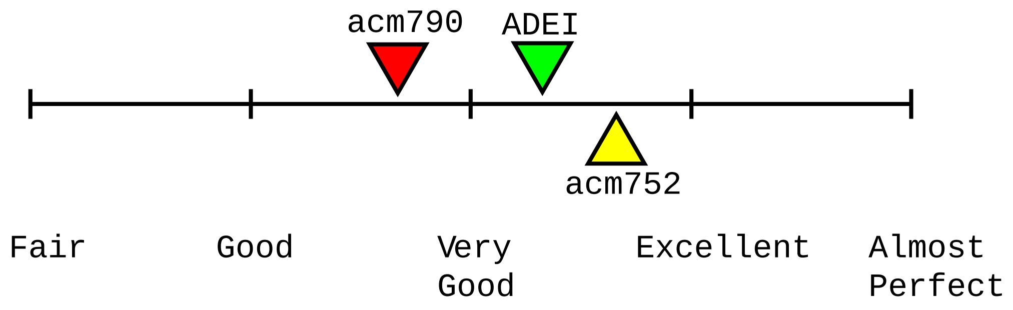
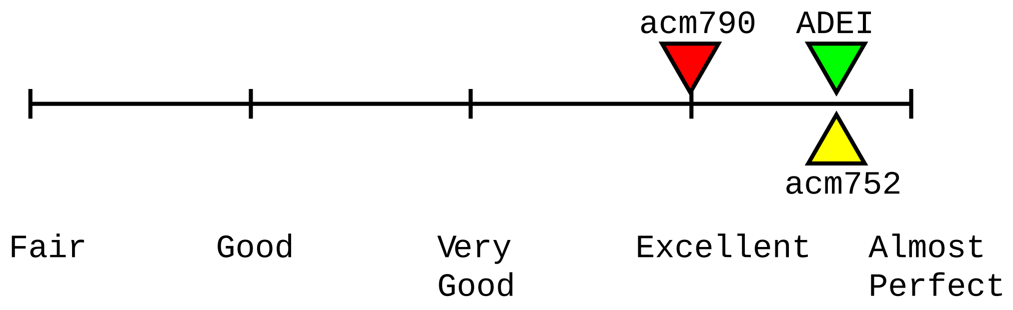


Figure 4.5: Expected fit over Data Sets I-III, on function F_3.



4.1 Conclusions, Limitations and Future Work

We studied and quantified the performance of a general technique, presented in [9], that produces a piecewise interpolant for the numerical solution of a PDE over a scattered data set, and found that noticeable discontinuities may arise because of a poor choice of random collocation points. To attenuate this problem, we developed an algorithm that chooses alternate collocation points. This process adds an overhead to the interpolation scheme, but we believe that the accuracy obtained justifies its use.

The ADEI approach was evaluated using several data sets and test functions, and the results were compared with two algorithms from the ACM repository: acm752, and acm790. We observed that for the given data sets and functions, ADEI performed an excellent to near perfect fit, with better, if not comparable, accuracy results than the other algorithms, in particular acm752. The best two overall methods are ADEI and acm752, which, in our tests use gradient information. It remains to be seen how to include this information into acm790, to improve its accuracy.

Our work only begins an investigation of the sensitivity of the choice of collocation points of the interpolant generated by ADEI. Although there are known cases where the choice will generate a bad solution, i.e., with noticeable discontinuities, we still cannot see a clear way to generate good collocation points. Future work in this area will address this issue as well as the issue of how ADEI and the optimization scale to higher dimensions.

The obtained results raise the question of accuracy versus continuity: although for visualization purposes one may be inclined to prefer continuity (because we don't want to 'see' gaps in the surface), for data with high accuracy the discontinuities may be invisible to the eye. Of course, that is not the case if the interpolant is subject to a zooming operation. On the other hand, in the case of numerical queries (without visualization), the accurate method should be preferred. In either circumstance, both methods can benefit from each other using a two stage interpolation process, which first generates an approximation to the function on a higher resolution mesh, and later interpolates that approximation. In our case the density of the grid could be augmented by the accurate method, and later interpolated by a more continuous scheme.

Since the error measure used does not take into consideration the human visual system, it is fair to ask how relevant are the results obtained, with respect to the performance of a method in scientific visualization. Other metrics should be introduced and applied to the evaluation of approximation and interpolation techniques.

Bibliography

[1] H. AKIMA. Algorithm 761: Scattered-data surface fitting that has the accuracy of a cubic polynomial. *ACM Transactions on Mathematical Software*, 22(3):362 371, September 1996.

[2] R. E. BARNHILL. Representation and approximation of surfaces. *Mathematical Software III*, pages 69–120, 1977.

[3] R. BARTLES. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann Publishers, Inc, 1987.

[4] K. BASSO, P.R. ZINGANO, and C.M. FREITAS. Interpolation of scattered data: Investigating alternatives for the modified shepard method. *IEEE Symposium on Computer Graphics and Image Processing*, 1999.

[5] S.A. COONS. Surfaces for computed-aided design of space forms. *MIT, MAC-TR-41*, June 1967.

[6] C. DeBOOR. *A Practical Guide to Splines*. Springer-Verlag, 1978.

[7] J. DUCHON. Fonctons splines du type plaque mince en dimension 2. *Seminarire d'analyse numerique, Grenoble*, (231), 1975.

[8] M. De Berg (Editor). *Computational Geometry: Algorithms and Applications*. Springer Verlag, May 2000.

[9] W.H. ENRIGHT. The efficient computation of accurate approximate solutions of partial different equations at off-mesh points. 1999.

[10] G. FARIN. *Curves and Surfaces for Computer Aided Geometric Design - A Practical Guide. Fourth Edition*. Academic Press, 1997.

[11] G. FARIN and D. HANSFORD. Discrete coons patches. *Computer Aided Geometric Design*, 16(7):691–700, 1999.

[12] FOLEY, vanDAM, FEINER, and HUGHES. *Computer Graphics: Principles and Practice, Second Edition*. Addison-Wesley, 1990.

[13] R. FRANKE. Scattered data interpolation: Test of some methods. *Mathematics of Computation*, 38(175):181–199, 1982.

[14] R. FRANKE and G. NIELSON. Smooth interpolations of large data sets of scattered data. *International Journal for Numerical Methods in Engineering*, 15:1691–1704, 1980.

[15] H. HAGEN, G. NIELSON, and Y. NAKAJIMA. Surface design using triangular patches. *Computer Aided Geometric Design*, 13(9):789–957, December 1996.

[16] R.L. HARDY. Theory and applications of the multiquadric-biharmonic method. *Computers Math. Applic.*, 19(8-9):163–208, 1990.

[17] P. KASHYAP. Improving clough-tocher interpolants. *Computer Aided Geometric Design*, 13(7):583–671, October 1996.

[18] M. LOUNSBERY, C. LOOP, S. MANN, J. PAINTER D. MEYERS, T. DeROSE, and K. SLOAN. A testbed for the comparison of parametric surface methods. *Proceedings of the 1990 SPIE Conference on Curves in Computer Vision and Graphics*, pages 94–105, 1990.

[19] M. LOUNSBERY, C. LOOP, S. MANN, J. PAINTER D. MEYERS, T. DeROSE, and K. SLOANM. A survey of parametric scattered data fitting using triangular interpolant. *Curve and Surface Design*, 1992.

[20] H. MA, K. W. JONES, and E. STERN. Scientific visualization and data modeling of scattered sediment contaminant data in new york/new jersey estuaries. *Proc. of the Conference on Visualization '98, Research Triangle Park, NC*, pages 467–471, October 1998.

[21] S. MANN. Cubic precision clough-tocher interpolation. *Technical Report CS-98-15*, 1998.

[22] S. MANN. Continuity adjustments to triangular bezier patches that retain polynomial precision. *Research Report CS-2000-01*, 2000.

[23] G.M. NIELSON. Scattered data modeling. *Computer Graphics and Applications*, January 1993.

[24] G.M. NIELSON, P. BRUNET, M. GROSS, and S.V. KLIMENKO H.HAGEN. Research issues in data modeling for scientific visualization. *IEEE Computer Graphics and Applications*, March 1994.

[25] G.M. NIELSON, D. HOLLIDAY, and T. ROXBOROUGH. Cracking the cracking problem with coons patches. *Proceedings of the 1999 IEEE Visualization Conference, San Francisco, CA*, 1999.

[26] R. BROWN R. J. RENKA. Algorithm 792: Accuracy tests of acm algorithms for interpolation of scattered data in the plane. *ACM Transactions on Mathematical Software*, 25(1):78–94, March 1999.

[27] J.C. REITER. Texturated surface modeling using bezier patches. Master's thesis, University of Toronto, Graduate Department of Computer Science, 1996.

[28] R. J. RENKA. Algorithm 661 qshep3d: Quadratic shepard method for trivariate interpolation of scattered data. *ACM Transactions on Mathematical Software*, 14(2):151–152, June 1988.

[29] R. J. RENKA. Multivariate interpolation of large sets of scattered data. *ACM Transactions on Mathematical Software*, 14(2):139–148, June 1988.

[30] R. J. RENKA. Algorithm 790: Cshep2d: Cubic shepard method for bivariate interpolation of scattered data. *ACM Transactions on Mathematical Software*, 25(1):70–73, March 1999.

[31] R. J. RENKA. Algorithm 791: Tshep2d: Cosine series shepard method for bivariate interpolation of scattered data. *ACM Transactions on Mathematical Software*, 25(1):74–77, March 1999.

[32] R. J. RENKA and A.K. CLINE. A triangle based c1 interpolation method. *Rocky Mt. J Math*, 14(1):223–237, 1984.

[33] L. L. SCHUMAKER. Fitting surfaces to scattered data. *Approximation Theory II*, pages 203–268, 1976.

[34] L. L. SCHUMAKER. Triangulation methods. *Topics in Multivariate Approximation*, pages 219–232, 1987.

[35] H. SPTH. *One Dimensional Spline Interpolation Algorithms.* AK Peters, Ltd, 1995.

[36] H. SPTH. *Two Dimensional Spline Interpolation Algorithms.* AK Peters, Ltd, 1995.

[37] E. WELZL, P. SU, and R. DRYSDALE. A comparison of sequential delaunay triangulation algorithms. *Computational Geometry*, 7(5-6):263–406, 1997.

Chapter 5

Appendix: Relevant Matlab Code.

```
function [x,y,u,ux,uy]=gen_sd(n,strF)

%------------------------------------------------------------
% GEN_SD: Generate Trully Random Scattered Data Mesh Points
%
% Input:
%       n -> No. of mesh points to generate
%       strF -> name of function (F) being sampled
%
% Output:
%       x,y -> mesh point coordinates
%       u,ux,uy -> F, dF/dx, dF/dy values at the mesh points
%------------------------------------------------------------

nc=128;

zz=zeros(nc+1,nc+1);

gx=zeros(size(zz));

gy=zeros(size(zz));

x0=-0.1; xF=1.1;

dx=xF-x0;

xx=x0:dx/nc:xF;

for i=1:nc+1
```

```
   px=(i-1)/nc*xF+(1-(i-1)/nc)*x0;

   for j=1:nc+1

      py=(j-1)/nc*xF+(1-(j-1)/nc)*x0;

      eval('z, zx, zy]=' strF '(px,py);']);

      zz(i,j)=z;

      gx(i,j)=zx;

      gy(i,j)=zy;

   end

end

m=nc;

xi=rand(1,n);

xi=ceil(m*xi+1);

yi=rand(1,n);

yi=ceil(m*yi+1);

for i=1:n

   j=xi(i);

   k=yi(i);

   x(i)=xx(j);

   y(i)=xx(k);

   u(i)=zz(j,k);

   ux(i)=gx(j,k);

   uy(i)=gy(j,k);
```

```
end


function [c,cpv,cpw]=make_p(x,y,u,ux,uy,strG)

%-----------------------------------------------------------------
% MAKE_P:Make Polynomials (Interpolant).
% Input:
%       x,y -> mesh points
%       u,ux,uy -> values of F, Fx and Fy at the mesh points
%       strG -> function g_i
%
% Output:
%       c -> matrix of size 16 by k, where k is the number of
%            triangles (patches) defined by the mesh points x,y. It
%            Defines the piecewise ADAI interpolant.
%       cpv,cpw -> set of collocation points produced by the method.
%                  Each is a vector of length k.
%
% Uses:
%      st(s,t) = sum(i=0,3;sum(j=0,3;s^i*t^j))
%      st_x(...) = d st/ds
%      st_y(...) = d st/dt
% -----------------------------------------------------------------



e=3;

k1 = 1; % ---

k2 = 1; % --- > Uxx + Uyy + 0 Uxy

k3 = 0; % ---



% \# of triangles...

tri=delaunay(x,y);

n = size(tri); n= n(1);
```

```
c = zeros(n,(e+1)^2);

cpv=zeros(n,7);

cpw=zeros(n,7);


% for every triangle...

for k=1:n

   %given a triangle p1,p2,p3 with values

   % u1,u1x,u1y ; u2,u2x,u2y ; u3,u3x,u3y

   T=tri(k,:);


   p1 = [x(T(1)) y(T(1))]; p1x=p1(1); p1y=p1(2);

   u1 = u(T(1));

   u1x = ux(T(1));

   u1y = uy(T(1));


   p2 = [x(T(2)) y(T(2))]; p2x=p2(1); p2y=p2(2);

   u2 = u(T(2));

   u2x = ux(T(2));

   u2y = uy(T(2));


   p3 = [x(T(3)) y(T(3))]; p3x=p3(1); p3y=p3(2);

   u3 = u(T(3));

   u3x = ux(T(3));
```

```
u3y = uy(T(3));


% compute normalization factors D1, D2 (k)

xmn=min([p1x p2x p3x]);

xmx=max([p1x p2x p3x]);

D1=xmx-xmn;


ymn=min([p1y p2y p3y]);

ymx=max([p1y p2y p3y]);

D2=ymx-ymn;


% W = [a;b;c ; ...d...];


% 1. P(px,py) = u(px,py)

s1=(p1x-xmn)/D1;

t1=(p1y-ymn)/D2;


s2=(p2x-xmn)/D1;

t2=(p2y-ymn)/D2;


s3=(p3x-xmn)/D1;

t3=(p3y-ymn)/D2;
```

```
a1 = st(s1,t1,e);

a2 = st(s2,t2,e);

a3 = st(s3,t3,e);


% 2. Px(px,py) = ux(px,py)


b1 = 1/D1*st_x(s1,t1,e);

b2 = 1/D1*st_x(s2,t2,e);

b3 = 1/D1*st_x(s3,t3,e);


% 3. Py(px,py) = uy(px,py)


c1 = 1/D2*st_y(s1,t1,e);

c2 = 1/D2*st_y(s2,t2,e);

c3 = 1/D2*st_y(s3,t3,e);


% choose 7 collocation points...

% cp1 ... cp7


for i=1:7

   v=rand;

   w=rand;
   h=rand;

   nr=v+w+h;

   v=v/nr; w=w/nr; h=h/nr;

   cp = v*p1+w*p2+h*p3;

   cpx(i)=cp(1); cpy(i)=cp(2);

   cu(i)=(v*u1+w*u2+h*u3);

   cux(i)=(v*u1x+w*u2x+h*u3x);

   cuy(i)=(v*u1y+w*u2y+h*u3y);


   cpv(k,i)=cpx(i);

   cpw(k,i)=cpy(i);

end


h=max([D1 D2]);

h2=h*h;


%for each collocation point...

for i=1:7

   s=(cpx(i)-xmn)/D1;

   t=(cpy(i)-ymn)/D2;


   % 4. L[u(px,py)] = g(px,py,u,ux,uy)
```

```
    % L[u] = k1(cx,cy)*uxx(cx,cy)+k2(cx,cy)*uyy(cx,cy)


    d =k1/D1^2* ...

    [0 0 0 0 0 0 0 0 2 2*t 2*t^2 2*t^3 6*s 6*s*t 6*s*t^2 6*s*t^3]+ ...

     k2/D2^2* ...

     [0 0 2 6*t 0 0 2*s 6*s*t 0 0 2*s^2 6*s^2*t 0 0 2*s^3 6*s^3*t];


    % di = d

    eval(['d' num2str(i) '= d;']);


    % gi = strG(cpx(i), ...);

    eval(['g(i) = ' strG '(cpx(i),cpy(i),cu(i),cux(i),cuy(i));']);

    cpg(k,i)=g(i);

end


W = [a1;a2;a3; b1;b2;b3; c1;c2;c3; ...

h2*d1;h2*d2;h2*d3;h2*d4;h2*d5;h2*d6;h2*d7];


% Wc=b

b = [u1;u2;u3; u1x;u2x;u3x; u1y;u2y;u3y; ...

h2*g(1);h2*g(2);h2*g(3);h2*g(4);h2*g(5);h2*g(6);h2*g(7)];


cij=W^-1*b;
```

```
   c(k,:)=cij';

end


% 2nd stage: Get rid of the 'bad' c coefficients...

% =======================================================================

tfactor=1.5;

change=1;

while 1


   nc=[];

   for k=1:n

      nc(k)=norm(c(k,:));

   end

   ppl=sort(nc); ppl=ppl(1:length(ppl)-floor(0.1*length(ppl)));

   mc = mean(ppl); % mean(nc);

   cthresh = tfactor*mc;


   % check how many 'bad' cs are there...

   nbad=0;


   figure(2);

   meshc([0 1],[0 1],zeros(2,2)-max(abs(u))); view(2);
```

```
for k=1:n
   if nc(k)>cthresh
      cbad(nbad+1)=k;
      nbad=nbad+1;

      plotT(x,y,u,cpv,cpw,k);
   end
end

figure(1)
plot(nc);
hold on;
plot (zeros(size(nc))+cthresh,'g');
hold off;

% Exit conditions: all triangles satisfy requirements
% or some can't be improved
if nbad==0
   break;
end

if change==0
```

```
   break;
end


% for every bad triangle... (actually its collocation points)
change=0;
for nb=1:nbad
   k = cbad(nb);
   nck=norm(c(k,:));


   %given a triangle p1,p2,p3 with values u1,u1x,u1y ; u2,u2x,u2y ;
   % u3,u3x,u3y
   T=tri(k,:);


   p1 = [x(T(1)) y(T(1))]; p1x=p1(1); p1y=p1(2);
   u1 = u(T(1));
   u1x = ux(T(1));
   u1y = uy(T(1));


   p2 = [x(T(2)) y(T(2))]; p2x=p2(1); p2y=p2(2);
   u2 = u(T(2));
   u2x = ux(T(2));
   u2y = uy(T(2));
```

```
p3 = [x(T(3)) y(T(3))]; p3x=p3(1); p3y=p3(2);

u3 = u(T(3));

u3x = ux(T(3));

u3y = uy(T(3));


% compute normalization factors D1, D2 (k)

xmn=min([p1x p2x p3x]);

xmx=max([p1x p2x p3x]);

D1=xmx-xmn;


ymn=min([p1y p2y p3y]);

ymx=max([p1y p2y p3y]);

D2=ymx-ymn;


% W = [a;b;c ; ...d...];


% 1. P(px,py) = u(px,py)

s1=(p1x-xmn)/D1;

t1=(p1y-ymn)/D2;


s2=(p2x-xmn)/D1;

t2=(p2y-ymn)/D2;
```

```
s3=(p3x-xmn)/D1;

t3=(p3y-ymn)/D2;


a1 = st(s1,t1,e);

a2 = st(s2,t2,e);

a3 = st(s3,t3,e);


% 2. Px(px,py) = ux(px,py)

b1 = 1/D1*st_x(s1,t1,e);

b2 = 1/D1*st_x(s2,t2,e);

b3 = 1/D1*st_x(s3,t3,e);


% 3. Py(px,py) = uy(px,py)

c1 = 1/D2*st_y(s1,t1,e);

c2 = 1/D2*st_y(s2,t2,e);

c3 = 1/D2*st_y(s3,t3,e);


% choose 7 new collocation points (cp1 ... cp7) that

% hopefully will make the patch k well behaved...


cij=c(k,:);
```

```
tries=0; % \# of times generates points for k

maxtries=100;

while 1

   % generate cps...

   for i=1:7

      v=rand; w=rand; h=rand;

      nr=v+w+h;

      v=v/nr; w=w/nr; h=h/nr;

      cp = v*p1+w*p2+h*p3;

      cpx(i)=cp(1); cpy(i)=cp(2);

      cu(i)=(v*u1+w*u2+h*u3);

      cux(i)=(v*u1x+w*u2x+h*u3x);

      cuy(i)=(v*u1y+w*u2y+h*u3y);


      cpv(k,i)=cpx(i);

      cpw(k,i)=cpy(i);

   end


   h=max([D1 D2]);  h2=h*h;


   %for each collocation point...

   for i=1:7

      s=(cpx(i)-xmn)/D1;  t=(cpy(i)-ymn)/D2;
      % 4. L[u(px,py)] = g(px,py,u,ux,uy)

      % L[u] = k1(cx,cy)*uxx(cx,cy)+k2(cx,cy)*uyy(cx,cy)


      d=k1/D1^2* ...

        [0 0 0 0  0 0 0 0 2 2*t 2*t^2 2*t^3 6*s 6*s*t 6*s*t^2 6*s*t^3]+...

         k2/D2^2* ...

        [0 0 2 6*t 0 0 2*s 6*s*t 0 0 2*s^2 6*s^2*t 0 0 2*s^3 6*s^3*t];


      % di = d

      eval(['d' num2str(i) '= d;']);


      % gi = strG(cpx(i), ...);

      eval(['g(i) = ' strG '(cpx(i),cpy(i),cu(i),cux(i),cuy(i));']);

      cpg(k,i)=g(i);

   end


   W = [a1;a2;a3; b1;b2;b3; c1;c2;c3;...

   h2*d1;h2*d2;h2*d3;h2*d4;h2*d5;h2*d6;h2*d7];

   b = [u1;u2;u3; u1x;u2x;u3x; u1y;u2y;u3y;...

       h2*g(1);h2*g(2);h2*g(3);h2*g(4);h2*g(5);h2*g(6);h2*g(7)];

   cij=W^-1*b;
```

```
   %if choice was good...

   if norm(cij)<=nck

      break;

   end


   if tries > maxtries

      break;

   end


   tries=tries+1;

end


if norm(cij)<norm(c(k,:))

   c(k,:)=cij';

   cn(k)=norm(cij);

   change=1;

end


figure(1)

plot(nc);

hold on;

plot ([0 k k k k n],[cthresh cthresh max(nc) 0 cthresh cthresh],'g');
```

```
      l=tries/maxtries; led=max(nc)*l+(1-l)*cthresh;

      plot([k k],[cthresh led],'r');

      hold off;



   end

end



function [iz,izx,izy]=imesh(n,x,y,c)

%------------------------------------------------------------------------
% IMESH: Generate Interpolating Surface Sampling Over a Uniform Mesh
%
% Input:
%       n -> dimension of sampling mesh will be n by n
%       x,y -> scattered mesh points
%       c -> polynomial coefficients matrix obtained from make_p(...)
%
% Output:
%       iz -> n by n matrix with the sampled interpolated solution
%       izx, izy -> n by n matrix with the sampled interpolated partials
%
% Uses:
%      pdev(...) = sum(i=0,3;sum(j=0,3;c_ij*s^i*t^j))
%      pdev_x(...) = d pdev/dx
%      pdev_y(...) = d pdev/dy
%------------------------------------------------------------------------

e=3;


iz=zeros(n+1,n+1)+NaN;

t=delaunay(x,y);

px=0;
```

```
for i=1:n+1

   px=(i-1)/(n);

   for j=1:n+1

      py=(j-1)/(n);

      kt=tsearch(x,y,t,px,py);k = kt(1);


      if isnan(k)

         iz(j,i)=0;

      else

         cij=c(k,:)';


         % compute normalization factors D1, D2 (k)

         tr=t(k,:);

         p1x=x(tr(1));

         p2x=x(tr(2));

         p3x=x(tr(3));

         xmn=min([p1x p2x p3x]);

         xmx=max([p1x p2x p3x]);

         D1=xmx-xmn;


         p1y=y(tr(1));

         p2y=y(tr(2));
         p3y=y(tr(3));

         ymn=min([p1y p2y p3y]);

         ymx=max([p1y p2y p3y]);

         D2=ymx-ymn;



         iz(j,i)=pdev(xmn,ymn,cij,D1,D2, px,py);

         izx(j,i)=pdev_x(xmn,ymn,cij,D1,D2, px,py);

         izy(j,i)=pdev_y(xmn,ymn,cij,D1,D2, px,py);

      end

   end

end
```