UNIVERSITEIT • STELLENBOSCH • UNIVERSITY

Radial Basis Function Interpolation

by

Wilna du Toit

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science (Mathematical Sciences) at the University of Stellenbosch

Applied Mathematics, Department of Mathematical Sciences
University of Stellenbosch
Private Bag X1, 7602 Matieland, South Africa

Supervisors:

Prof B.M. Herbst   Dr K.M. Hunter

March 2008

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature: ....
Wilna du Toit

Date: ....

Abstract

Radial Basis Function Interpolation

Wilna du Toit
Applied Mathematics, Department of Mathematical Sciences
University of Stellenbosch
Private Bag X1, 7602 Matieland, South Africa

Thesis: MSc (Mathematical Sciences)
March 2008

A popular method for interpolating multidimensional scattered data is using radial basis functions. In this thesis we present the basic theory of radial basis function interpolation and also regard the solvability and stability of the method. Solving the interpolant directly has a high computational cost for large datasets, hence using numerical methods to approximate the interpolant is necessary. We consider some recent numerical algorithms. Software to implement radial basis function interpolation and to display the 3D interpolants obtained, is developed. We present results obtained from using our implementation for radial basis functions on GIS and 3D face data as well as an image warping application.

Uittreksel

Interpolasie met Radiale Basis Funksies

("Radial Basis Function Interpolation")

Wilna du Toit

Toegepaste Wiskunde, Departement Wiskundige Wetenskappe
Universiteit van Stellenbosch
Privaatsak X1, 7602 Matieland, Suid Afrika

Tesis: MSc (Wiskundige Wetenskappe)

Maart 2008

Interpolasie deur gebruik te maak van radiale basis funksies is 'n gewilde manier om ongeorganiseerde, multidimensionele data te modelleer. In hierdie tesis verduidelik ons die basiese teoretiese idees rondom die metode. Hiervolgens kyk ons ook na die oplosbaarheid en stabiliteit van die metode. Om die interpolant direk vir 'n groot datastel op te los, het 'n hoë berekeningskoste tot gevolg. Dus is dit noodsaaklik om numeriese metodes te gebruik om die interpolant te benader en ons bestudeer van hierdie onlangse algoritmes. Sagteware om radiale basis funksie interpolasie te implementeer en om die verkreë 3D interpolante uit te beeld, is ontwikkel. Ons gebruik hierdie sagteware om eksperimente uit te voer op GIS en 3D gesig data, asook 'n toepassing op beeldverwronging.

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations who contributed to making this work possible :

- the University of Stellenbosch and the Department of Applied Mathematics
- the National Research Foundation, which made a large contribution to the funding of my postgraduate studies
- Karin and Prof. Herbst, for their ideas, patience, support, enthusiasm and all the extra hours!
- my parents and friends for their support and comic relief

Contents

Chapter 1

Introduction

Applications in a wide range of fields such as mathematics, engineering, computer science, business studies and biology make extensive use of data interpolation. Examples of these applications are surface reconstruction, terrain modeling, image warping and the numerical solution of partial differential equations. In these examples, the data is often multidimensional, scattered and the datasets are large.

We therefore seek a method to solve scattered data interpolation in many dimensions. Amongst the methods developed for this are finite element methods, wavelets and splines. These methods however generally require the data to be organised in a mesh of some sort. As explained by Wendland in [Wen05], the most promising methods that are truely meshless include the moving least squares approximation, partition-of-unity methods and radial basis function interpolation. Franke's survey in 1982 [Fra82] on scattered data interpolation methods includes tests on inverse distance weighted methods, rectangle and triangle blending methods, finite element methods and radial basis function methods.

Radial basis function interpolation originated in the 1970's [Har71] in Hardy's cartography application and since that time has been successfully used in a variety of applications. Examples are measurements of the earth's temperature from meterological stations situated at scattered sites and learning applications by using neural networks.

We explain the theory of radial basis function interpolation and give theorems regarding unique solvability, but the emphasis of this thesis is on the algorithms rather than the proofs. Consequently, we provide software capable of performing interpolation on multidimensional data for different types of basis functions. An application to view 3-dimensional plots is provided to view the results. This application and its uses as well as the software that performs radial basis function interpolation is the subject of the appendices.

In Chapter 2 we give an overview of radial basis functions and how they are used in a simple example, the purpose of which is to give the reader a intuitive feel for the procedure. Furthermore some results that ensure the method generates a unique solution are quoted. The types of basis functions and their different historical origins are also explored, and we consider the issue regarding the so-called uncertainty principle in terms of the stability versus the accuracy of the interpolation.

The matrices involved are non-sparse and in the usual cases of a large number of datapoints, very large. Numerical methods to perform the computations more efficiently are discussed in Chapter 3. These include algorithms for fast matrix-vector multiplication as well as iterative methods to approximate the interpolant.

We illustrate some resulting interpolants in Chapter 4, and also include an image warping experiment, and conclude with Chapter 5 where we discuss our results as well as future work.

Chapter 2

Radial Basis Functions Theory

As mentioned in Chapter 1, in many practical applications where we need to approximate a function of many variables, we have scattered data. An example is the 3D data of the Stanford bunny [Lab07] in Figure 2.1. To approximate multidimensional scattered data the radial basis function method has been developed. It is suitable for this task since a distance function is calculated between every datapoint and its neighbours, also there are no restrictions upon the placement of datapoints except that they should all be different.

Franke's survey [Fra82] of 1982, as well as the work done by Micchelli in 1986 [Mic86] into proving the non-singularity of the interpolation matrix, have done much for the popularity of the method. We will briefly explain the basic ideas behind the method with the help of a simple example. Next we list popular basis functions used and a slight expansion of the basic ideas enables us to prove the unique solvability of the method. Lastly we look at the stability of the method with regards to the shape parameter, a parameter of the basis function which is very sensitive for certain types of radial basis functions.

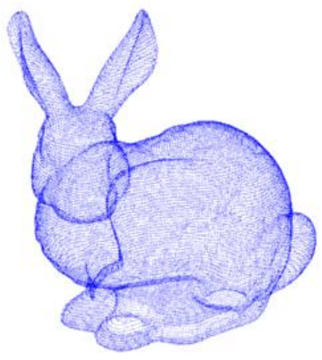


Figure 2.1: Point cloud of a bunny model [ARA07]

2.1 Basic Concepts

A function $g : \mathbb{R}^d \to \mathbb{R}$ for which its function value only depends on the magnitude of its argument, is called radial. For example $g(\mathbf{x}) = \phi(\|\mathbf{x}\|) = \phi(r)$, where $\phi : [0, \infty) \to \mathbb{R}$ and r is the length of $\mathbf{x}$. This means that ϕ is constant for input vectors of the same length. We call ϕ a *radial basis function* (RBF).

Suppose we are given data consisting of the form $(\mathbf{x}_i, f_i)$ for $i = 1, 2, \ldots, n$. Our goal is to find an interpolant $s(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^d$, satisfying

$$s(\mathbf{x}_i) = f_i, \quad i = 1, 2, \ldots, n. \tag{2.1.1}$$

For a radial basis function interpolant, we require that $s(\mathbf{x})$ be a linear combination of translates of $\phi(\mathbf{x})$, i.e.

$$s(\mathbf{x}) = \sum_{i=1}^{n} \lambda_i \phi(\|\mathbf{x} - \mathbf{x}_i\|), \ \mathbf{x} \in \mathbb{R}^d. \tag{2.1.2}$$

From the condition (2.1.1) it follows that

$$\sum_{i=1}^{n} \lambda_i \phi(\|\mathbf{x}_j - \mathbf{x}_i\|) = f_j, \ j = 1, 2, \ldots, n. \tag{2.1.3}$$

This can be written as

$$\begin{bmatrix} \phi(\|\mathbf{x}_1 - \mathbf{x}_1\|) & \phi(\|\mathbf{x}_2 - \mathbf{x}_1\|) & \ldots & \phi(\|\mathbf{x}_n - \mathbf{x}_1\|) \\ \phi(\|\mathbf{x}_1 - \mathbf{x}_2\|) & \phi(\|\mathbf{x}_2 - \mathbf{x}_2\|) & \ldots & \phi(\|\mathbf{x}_n - \mathbf{x}_2\|) \\ \vdots & \vdots & & \vdots \\ \phi(\|\mathbf{x}_1 - \mathbf{x}_n\|) & \phi(\|\mathbf{x}_2 - \mathbf{x}_n\|) & \ldots & \phi(\|\mathbf{x}_n - \mathbf{x}_n\|) \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} \tag{2.1.4}$$

in matrix form or

$$\Phi \boldsymbol{\lambda} = \mathbf{f}. \tag{2.1.5}$$

It is clear that the $n \times n$ matrix Φ is symmetric. For (2.1.4) to have a unique solution, we require that Φ be non-singular.

2.1.1 A Simple Example in 1 Dimension

Suppose we have 3 datapoints : $x_1 = 1$, $x_2 = 3$ and $x_3 = 3.5$. We are also given the corresponding function values : $f_1 = 1$, $f_2 = 0.2$ and $f_3 = 0.1$. We choose the radial basis function $\phi(r) = e^{-r^2}$. Firstly to calculate our interpolant $s(x)$ we need to calculate $\boldsymbol{\lambda}$ for the matrix as in (2.1.2). Using our dataset we have

$$\begin{bmatrix} \phi(0) & \phi(2) & \phi(2.5) \\ \phi(2) & \phi(0) & \phi(0.5) \\ \phi(2.5) & \phi(0.5) & \phi(0) \end{bmatrix} \boldsymbol{\lambda} = \begin{bmatrix} 1 & 0.02 & 0.002 \\ 0.02 & 1 & 0.78 \\ 0.002 & 0.78 & 1 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.2 \\ 0.1 \end{bmatrix}.$$

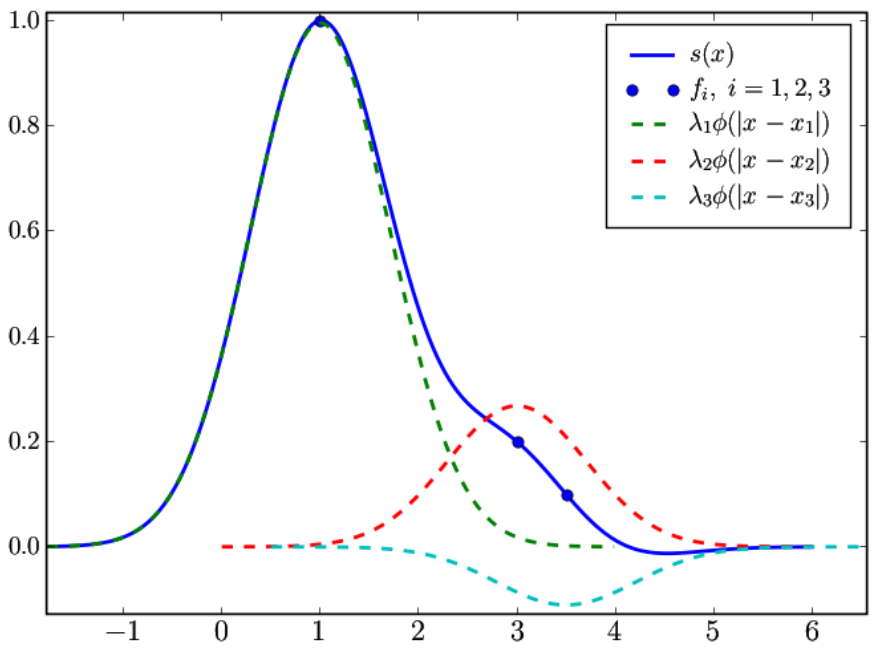


Figure 2.2: A simple example in 1 dimension.

Solving this equation directly we find that $\lambda_1 = 0.995$, $\lambda_2 = 0.268$ and $\lambda_3 = -0.111$. This means that our interpolant is

$$s(x) = 0.995\phi(|x-1|) + 0.268\phi(|x-3|) - 0.111\phi(|x-3.5|).$$

This interpolant is plotted in Figure 2.2, as shown by the solid line. Also the three functions plotted with dashed lines represent each datapoint's contribution to the interpolant. At any value of x, summing these functions gives the function value of the interpolant at that point.

2.1.2 Polynomial Terms

It is sometimes useful to add low order polynomials to our method of radial basis function interpolation. We let Π_{m-1}^d be the linear space of polynomials from $\mathbb{R}^d$ to $\mathbb{R}$ of degree at most $m-1$, and choose p_j, $j = 1, 2, \ldots, \hat{m}$ as a basis for this space. This means we let $s(\mathbf{x})$ have the form

$$s(\mathbf{x}) = \sum_{i=1}^{n} \lambda_i \phi(\|\mathbf{x} - \mathbf{x}_i\|) + \sum_{j=1}^{\hat{m}} \gamma_j p_j(\mathbf{x}),\ \ \mathbf{x} \in \mathbb{R}^d, \tag{2.1.6}$$

with the additional constraints

$$\sum_{i=1}^{n} \lambda_i p_j(\mathbf{x}_i) = 0, \; j = 1, 2, \ldots, \hat{m}.$$

Adding the extra constraints and the polynomial conditions to the interpolant, we find the system of linear equations,

$$\begin{bmatrix} \Phi & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \gamma \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{0} \end{bmatrix}, \tag{2.1.7}$$

or alternatively

$$\Phi\boldsymbol{\lambda} + P\boldsymbol{\gamma} = \mathbf{f}, \tag{2.1.8}$$

$$P^T\boldsymbol{\lambda} = \mathbf{0}. \tag{2.1.9}$$

The addition of polynomials of degree not more than $m-1$ guarantees polynomial precision, meaning that if the data comes from a polynomial of degree less than or equal to $m-1$ they are fitted by that polynomial. We call m the order of the radial basis function, to be further explained in Section 2.3.

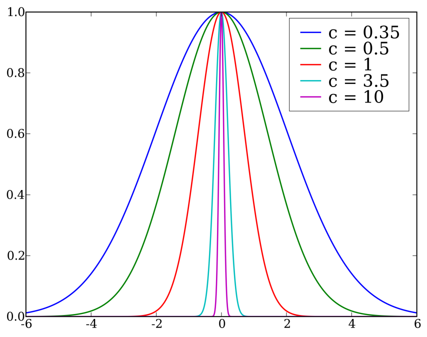
2.2 Types of Radial Functions

Some classical choices for radial basis functions can be seen in Table 2.1. Most of these functions have a parameter, c, to adjust the shape of the function. The Gaussian kernel, $\phi(r) = e^{-c^2r^2}$ is commonly used and can be seen for different values of the shape parameter in Figure 2.3(a). This function and the inverse multiquadric (see Figure 2.3(c)), $\phi(r) = \frac{1}{\sqrt{r^2+c^2}}$, has order $m = 0$. Another popular function is the multiquadric (see Figure 2.3(b)), $\phi(r) = \sqrt{r^2+c^2}$, for which the order is $m = 1$. Other radial functions can be seen in Figures 2.3(d) and 2.3(e).

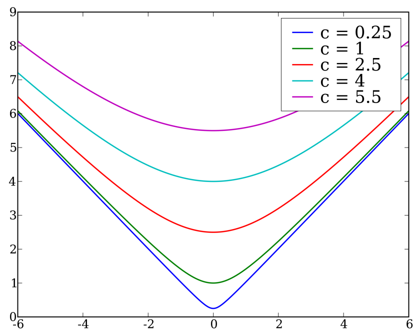
Radial Basis Function	$\phi(r)$	**parameters**	**order**
Gaussians	$e^{-(cr)^2}$	$c > 0$	0
Polyharmonic Splines	r^{2k-1}	$k \in \mathbb{N}$	$m = k$
	$r^{2k}\log(r)$	$k \in \mathbb{N}$	$m = k + 1$
Multiquadrics	$\sqrt{r^2+c^2}$	$c > 0$	1
Inverse Multiquadrics	$\frac{1}{\sqrt{r^2+c^2}}$	$c > 0$	0
Inverse Quadratics	$\frac{1}{r^2+c^2}$	$c > 0$	0

Table 2.1: Classic types of radial basis functions

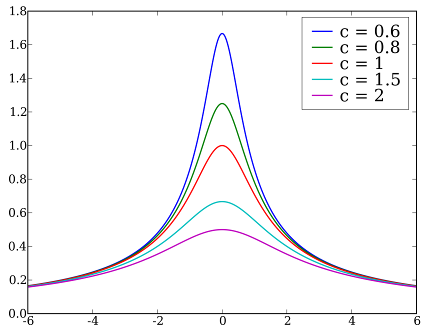
From Figure 2.3 it can be seen that the shape parameter usually adjusts the 'flatness' or 'steepness' of a function.



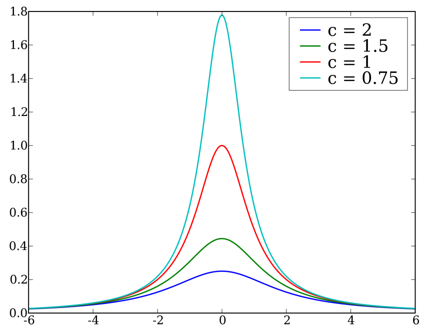
(a) Gaussian basis functions



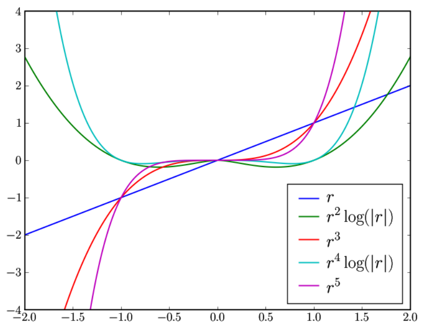
(b) Multiquadric basis functions



(c) Inverse Multiquadric basis functions



(d) Inverse Quadratic basis functions



(e) Polyharmonic splines

Figure 2.3: Radial basis functions

2.3 Uniqueness of the Interpolant

We would like to show that the linear system (2.1.7) has a unique solution for any choice of datapoints and function values. We can do this by showing that the homogeneous system of (2.1.7),

$$\Phi\boldsymbol{\lambda} + P\boldsymbol{\gamma} = \mathbf{0}, \tag{2.3.1}$$

$$P^T\boldsymbol{\lambda} = \mathbf{0}, \tag{2.3.2}$$

has the unique solution $\boldsymbol{\lambda} = \mathbf{0}$ and $\boldsymbol{\gamma} = \mathbf{0}$. If we multiply (2.3.1) by $\boldsymbol{\lambda}^T$ from the left and use (2.1.9) we obtain

$$\boldsymbol{\lambda}^T \Phi \boldsymbol{\lambda} = \mathbf{0}, \tag{2.3.3}$$

while still requiring $P^T \boldsymbol{\lambda} = 0$. This leads us to the following definition:

Definition 1. *A continuous radial function $\phi : \mathbb{R}^d \to \mathbb{R}$ is said to be conditionally positive definite of order m on $\mathbb{R}^d$ iff*

$$\boldsymbol{\lambda}^T \Phi \boldsymbol{\lambda} = \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j \phi(\|\mathbf{x}_i - \mathbf{x}_j\|) > 0$$

holds for all sets $X = \{\mathbf{x}_1, \dots, \mathbf{x}_N\} \subset \mathbb{R}^d$ with N distinct points and all nonzero vectors $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_N)^T \in \mathbb{R}^N$ that satisfy

$$\sum_{i=1}^{N} \lambda_i p(\mathbf{x}_i) = 0, \text{ for all } p \in \Pi_{m-1}^d.$$

A conditionally positive definite function of order 0 on $\mathbb{R}^d$ is called positive definite on $\mathbb{R}^d$.

The term 'conditionally' comes from the requirement that we do not need $\boldsymbol{\lambda}^T \Phi \boldsymbol{\lambda} > 0$ to hold for all nonzero vectors in $\mathbb{R}^N$, but only for the vectors in the null space of P^T.

Therefore, if ϕ is conditionally positive definite, (2.3.3) implies that $\boldsymbol{\lambda} = \mathbf{0}$ and this reduces to $P\boldsymbol{\gamma} = \mathbf{0}$ in (2.3.1). For this to imply $\boldsymbol{\gamma} = \mathbf{0}$ we require the columns of P to be linearly independent, which depends on the geometry of the interpolation points. It is sufficient to require that the point set X be Π_{m-1}^d-unisolvent, meaning any polynomial $p \in \Pi_{m-1}^d$ can be uniquely recovered from its function values on X.

Definition 2. *A set of points $X = \{\mathbf{x}_1, \dots, \mathbf{x}_N\} \subset \mathbb{R}^d$ is called Π_{m-1}^d-unisolvent if the only polynomial of total degree at most $m-1$ interpolating zero data on X is the zero polynomial,*

$$p(\mathbf{x}_j) = 0 \text{ for all } 1 \le j \le N \Longrightarrow p \equiv 0, \quad \text{for } p \in \Pi_{m-1}^d.$$

The requirement of unisolvency is a rather weak one, for example for $m = 2$ it means a set of points $X = \{\mathbf{x}_1, \dots, \mathbf{x}_N\} \subset \mathbb{R}^2$ cannot lie on a straight line. In conclusion, to ensure a unique solution to the interpolation problem, we require the radial basis function to be conditionally positive definite of a certain order m and the data points to be Π_{m-1}^d-unisolvent.

In [Mic86] Micchelli related conditionally positive radial functions to completely monotone functions.

k	$\psi^{(k)}(r)$	$(-1)^k\psi^{(k)}(r)$	σ
0	$r^{\frac{3}{2}}$	$r^{\frac{3}{2}}$	$+$
1	$\frac{3}{2}r^{\frac{1}{2}}$	$-\frac{3}{2}r^{\frac{1}{2}}$	$-$
2	$\frac{3}{4}r^{-\frac{1}{2}}$	$\frac{3}{4}r^{-\frac{1}{2}}$	$+$
3	$-\frac{3}{8}r^{-\frac{3}{2}}$	$\frac{3}{8}r^{-\frac{3}{2}}$	$+$
4	$\frac{9}{16}r^{-\frac{5}{2}}$	$\frac{9}{16}r^{-\frac{5}{2}}$	$+$

Table 2.2: Finding the integer m for $\phi = r^3$

Definition 3. *An infinitely differentiable function ψ is said to be completely monotone on $(0, \infty)$ iff*

$$(-1)^k\psi^{(k)}(r) \geq 0, \;\; k = 0, 1, 2, \ldots,$$

holds for all $r \in (0, \infty)$.

Guo, Hu and Sun [GHS93] proved a conjecture of Micchelli about a sufficient condition for radial functions to be classified as conditional positive definite.

Theorem 1. *Let $\phi(r)$ be a continuous radial function and $\psi(r) = \phi(\sqrt{r})$. Then $\phi(r)$ is conditionally positive definite of order m and radial on $\mathbb{R}^d$ iff $(-1)^m\psi^{(m)}$ is completely monotone on $(0, \infty)$.*

We can now easily verify that all the functions listed in Table 2.1 are conditionally positive definite with the listed order of m.

Here m as before refers to the order of $\phi(\mathbf{x})$, see Powell [Pow05]. Given a function $\phi(r)$ and defining $\psi(r) = \phi(\sqrt{r})$ where $r \geq 0$ and $\psi^{(k)}(r)$ denotes the kth derivative of ψ (for k a positive integer, and $\psi^{(0)}(r) = \psi(r)$) then m is the least nonnegative integer such that the sign of $(-1)^k\psi^{(k)}$ for $k = m, m+1, \ldots$ is independent of both r and k. We call the sign σ. For instance if we take $\phi(r) = r^3$ we find the order of ϕ to be 2, as seen in Table 2.2.

Note that the addition of polynomials of degree $m-1$ is a sufficient condition for the uniqueness of the interpolant, but practical experiments show that excluding the polynomial part can still lead to unique and solvable interpolants in many cases.

2.4 Stability

As observed by Schaback [Sch95] there is a trade-off between the accuracy of the interpolation and the condition number of the matrix Φ in (2.1.5). The spectral condition number of any matrix A is given by

$$\text{cond}(A) = \frac{\sigma_{max}}{\sigma_{min}}$$

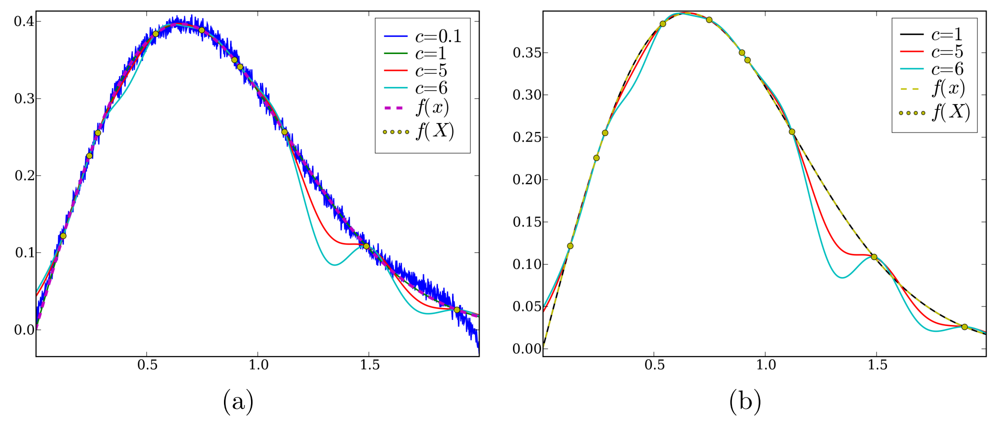


Figure 2.4: Gaussian RBF interpolants of data for different shape parameters

where σ_{min} is the smallest singular value and σ_{max} the largest singular value of A. In order to improve the accuracy of the interpolant we can change the shape parameter c to use 'flat' basis functions (or scale the data), which leads to a more ill-conditioned problem since the condition number of the matrix Φ increases. This trade-off has been called the uncertainty principle.

Alternatively we can increase the number of interpolation points to increase the accuracy, but this also leads to a larger condition number of Φ.

In Figure 2.4 the interpolants produced by different values of the shape parameter for the Gaussian kernel are illustrated. In this example the function $f(x) = e^{-x^2}\sin(x)$ is sampled at a random set of points X. In Table 2.3 the corresponding condition numbers and the errors at the interpolation points of these interpolants are given. For $c = 0.1$ the interpolant follows the trend of the function $f(x)$, but contains noise. This confirms the ill-conditioning which can be seen from the large condition number in Table 2.3. On the other hand for $c = 6$ the interpolant has good conditioning and high accuracy at the interpolation points, but deviates greatly from the function $f(x)$ between the datapoints.

c	condition number	ℓ_2-error at datapoints
0.1	8.13e17	0.016
1	3.22e12	9.79e-14
5	526	2.78e-17
6	195	6.22e-17

Table 2.3: Results for Gaussian RBF interpolants

In summary increasingly flat basis functions provide more accurate interpolants, but are unusable because of the numerical instabilities from ill-conditioning. This problem is addressed in [FW04], where algorithms for the stable computation of these interpolants are provided.

Chapter 3

Practical Implementation

Although the radial basis function method for interpolation is essentially very simple, difficulties arise when applications use a large number of datapoints. Directly solving the N by N system requires computational time of order N^3 as well as storage space of order N^2. Sparse methods cannot be used since the matrices arising are in most cases non-sparse. Additionally each function value evaluation will take order N computations. Clearly this approach is inadequate for systems with thousands of points. Hence in this chapter we will look at numerical methods to more efficiently solve our problem. Because the data we have to interpolate often contains noise or inaccuracies we will be satisfied with an approximation to the required interpolant, if that means a reduced computational cost.

In this chapter we briefly discuss multilevel and fast multipole methods. Next we explain the conjugate gradient method, leading to the Faul, Goodsell and Powell (FGP) algorithm, implemented in this thesis.

3.1 Multilevel Methods

An alternative way of dealing with a large number of interpolation points is a multiresolution technique. This means we wish to split our datapoints into several levels, forming a hierarchy of sets to be interpolated step-wise. This method was first implemented by Iske and Floater in [FI96].

Given a dataset $X = \{\mathbf{x}_1, \dots, \mathbf{x}_N\} \subset \mathbb{R}^d$, we wish to decompose X into nested subsets

$$X_1 \subset X_2 \subset \cdots \subset X_{K-1} \subset X_K = X.$$

How these subsets are chosen is very important with regards to the effectiveness of the method. We want to choose the subsets in such a way that the first subset highlights global features of the underlying function we wish to interpolate, and that subsequent subsets add finer or more local details. Also we would wish the density of each subset to be as uniform as possible. Thinning algorithms to compute these subsets using Delauney triangulations and priority queues were developed by Floater and Iske and more information can be found in [FI98].

From these subsets we recursively compute a sequence of interpolants $s_1, \dots, s_K$ as outlined in Algorithm 1.

```
$s_0 \equiv 0$
Fit the interpolant, $t_1(\mathbf{x}) = (f - s_0)(\mathbf{x})$, $x \in X_1$
$s_1(\mathbf{x}) = s_0(\mathbf{x}) + t_1(\mathbf{x})$
Fit the interpolant, $t_2(\mathbf{x}) = (f - s_1)(\mathbf{x})$, $x \in X_2$
$s_2(\mathbf{x}) = s_1(\mathbf{x}) + t_2(\mathbf{x})$
$\vdots$
Fit the interpolant, $t_K(\mathbf{x}) = (f - s_{K-1})(\mathbf{x})$, $x \in X_K$
$s_K(\mathbf{x}) = s_{K-1}(\mathbf{x}) + t_K(\mathbf{x})$
```

Algorithm 1: Multilevel method

It can be easily verified that for every subset X_j

$$s_j|_{X_j} = f|_{X_j}, \ j = 1, \dots, K.$$

A more thorough investigation about the approximation behaviour and computational costs of this method can be found in [Isk01]. A variation on this method involving domain decomposition can be seen in [IL05]. A multilevel method for evaluating radial basis function interpolants has been developed by Livne and Wright in [LW06].

3.2 Fast Multipole Methods

The fast multipole method was developed by Greengard and Rokhlin in 1987 [GR87] for the fast summation of potential fields as arising in particle simulations. Specifically it allows the matrix-vector multiplication of a matrix with a certain form to be calculated in $\mathcal{O}(n\log n)$ or even $\mathcal{O}(n)$ operations instead of the $\mathcal{O}(n^2)$ operations required for direct multiplication. The matrix-vector product required by this method is of the form

$$f_j = \sum_{i=1}^{n} \lambda_i \Phi(\mathbf{y}_j, \mathbf{x}_i), \ j = 1, \dots, m, \tag{3.2.1}$$

where $\mathbf{x}_i \in \mathbb{R}^d$ are called source points with corresponding source weights λ_i, $i = 1, \dots, n$ and $\mathbf{y}_j \in \mathbb{R}^d$ are called target points; Φ is the potential function.

In short, the method divides the source points into points ‘near’ and ‘far’ from the target point. The potential function Φ is written as a converging series expansion which is then truncated to desired accuracy. The series for (3.2.1) is constructed so that it separates into a matrix-vector product depending on the source points only and a function evaluation depending only on the target points.

To use this method for radial basis function interpolation, it is necessary to find a series expansion for each different basis function. In [BG97] more information is given as well as a discussion on using the fast multipole method with the multiquadric basis function.

3.3 Conjugate Gradient Method

The conjugate gradient method is an iterative technique for the solution of $\mathbf{x}$ from $A\mathbf{x} = \mathbf{b}$, for A symmetric and positive definite. This method is thus suitable for calculating the coefficients of our radial basis function interpolant. For details of theorems and proofs related to the method, see [GVL96] Section 10.2.

It can be shown that finding the solution $\mathbf{x} = A^{-1}\mathbf{b}$, A being symmetric and positive definite, is equivalent to minimizing the function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b},$$

where $\mathbf{b} \in \mathbb{R}^n$. A simple method for minimizing $f(\mathbf{x})$ is called steepest descent. Hereby we take a series of steps $\mathbf{x}_1$, $\mathbf{x}_2$,..., in the direction in which f decreases the most until we are sufficiently close to the solution $\mathbf{x}$. This direction is the negative gradient, $-\nabla f(\mathbf{x}_i) = \mathbf{b} - A\mathbf{x}_i$ for step i. We define $\mathbf{r}_i = \mathbf{b} - A\mathbf{x}_i$ as the residual of $\mathbf{x}_i$ and if this residual is nonzero it means that there exists a positive α for which $f(\mathbf{x}_i + \alpha\mathbf{r}_i) < f(\mathbf{x}_i)$. In this way if $\mathbf{r}_i$ is a direction of steepest descent, α is the distance in that direction. Since

$$f(\mathbf{x}_i + \alpha\mathbf{r}_i) = f(\mathbf{x}_i) - \alpha\mathbf{r}_i^T\mathbf{r}_i + \frac{1}{2}\alpha^2\mathbf{r}_i^T A\mathbf{r}_i,$$

differentiation with respect to α leads to the result that

$$\alpha_i = \frac{\mathbf{r}_i^T\mathbf{r}_i}{\mathbf{r}_i^T A\mathbf{r}_i}$$

minimizes $f(\mathbf{x}_i + \alpha_i\mathbf{r}_i)$. From this we calculate our next step

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i\mathbf{r}_i.$$

The rate of convergence of steepest descent is very slow. An improvement on the steepest descent method is to choose a set of linearly independent search directions $\{\mathbf{p}_1, \mathbf{p}_2, \ldots\}$. For given $\mathbf{p}_i$, $f(\mathbf{x}_i) = f(\mathbf{x}_{i-1} + \alpha_i\mathbf{p}_i)$ is minimized by

$$\alpha_i = \frac{\mathbf{p}_i^T\mathbf{r}_{i-1}}{\mathbf{p}_i^T A\mathbf{p}_i}.$$

To ensure that $f(\mathbf{x}_i)$ is getting smaller we require that the newest search direction not be orthogonal to the previous residual, i.e $\mathbf{p}_i^T\mathbf{r}_{i-1} \neq 0$. If for each step $\mathbf{x}_i$ we minimize $f(\mathbf{x})$ for $\mathbf{x} \in \{\mathbf{x}_0 + \gamma_1\mathbf{p}_1 + \cdots + \gamma_i\mathbf{p}_i\}$, then we are guaranteed of convergence in n steps for $\mathbf{x} \in \mathbb{R}^n$. For easy computation of $\mathbf{x}_{i+1}$ from $\mathbf{x}_i$ it is necessary for the search directions to be A-conjugate, meaning $\mathbf{p}_i^T A\mathbf{p}_j = 0$ for any two search directions $\mathbf{p}_i$ and $\mathbf{p}_j$. It can be shown that search directions satisfying these properties exist as long as $\mathbf{p}_i^T\mathbf{r}_{i-1} \neq 0$, see Lemma 10.2.1 in [GVL96].

To combine the method of choosing A-conjugate search directions with steepest descent we want to choose $\mathbf{p}_i$ to be the closest vector to the previous residual $\mathbf{r}_{i-1}$, thus minimizing $\|\mathbf{p} - \mathbf{r}_{i-1}\|$, that is A-conjugate to the previous search directions. Several steps are required to develop an efficient way to compute $\mathbf{p}_i$, as seen in Lemma 10.2.2, Theorem 10.2.3 and Corollary 10.2.4 in [GVL96]. The end result is that we can compute $\mathbf{p}_i$ by

$$\mathbf{p}_i = \mathbf{r}_{i-1} + \beta_i\mathbf{p}_{i-1}, \tag{3.3.1}$$

where

$$\beta_i = \frac{\mathbf{p}_{i-1}^T A\mathbf{r}_{i-1}}{\mathbf{p}_{i-1}^T A\mathbf{p}_{i-1}}.$$

In order to compute the variables for our iterations more efficiently, we can compute our residuals recursively,

$$\begin{aligned} \mathbf{r}_i &= \mathbf{b} - A\mathbf{x}_i \\ &= \mathbf{b} - A(\mathbf{x}_{i-1} + \alpha_i\mathbf{p}_i) \\ &= \mathbf{r}_{i-1} - \alpha_i A\mathbf{p}_i. \end{aligned}$$

Now

$$\begin{aligned} \mathbf{r}_{i-1}^T\mathbf{r}_{i-1} &= \mathbf{r}_{i-1}^T\mathbf{r}_{i-2} - \alpha_{i-1}A\mathbf{p}_{i-1} \\ &= -\alpha_{i-1}A\mathbf{p}_{i-1}, \end{aligned}$$

and since $\mathbf{r}_{i-2} = \mathbf{r}_{i-1} + \alpha_{i-1}A\mathbf{p}_{i-1}$,

$$\begin{aligned} \mathbf{r}_{i-2}^T\mathbf{r}_{i-2} &= \mathbf{r}_{i-2}^T\mathbf{r}_{i-1} + \alpha_{i-1}\mathbf{p}_{i-1}^T A\mathbf{p}_{i-1} \\ &= \alpha_{i-1}\mathbf{r}_{i-2}^T A\mathbf{p}_{i-1}. \end{aligned}$$

If we use (3.3.1) then because of A-conjugacy

$$\begin{aligned} \alpha_{i-1}\mathbf{r}_{i-2}^T A\mathbf{p}_{i-1} &= \alpha_{i-1}(\mathbf{p}_{i-1}^T - \beta_{i-1}\mathbf{p}_{i-2}^T)A\mathbf{p}_{i-1} \\ &= \alpha_{i-1}\mathbf{p}_{i-1}^T A\mathbf{p}_{i-1}. \end{aligned}$$

From the above β_i becomes

$$\beta_i = \frac{\mathbf{r}_{i-1}^T\mathbf{r}_{i-1}}{\mathbf{r}_{i-2}^T\mathbf{r}_{i-2}},$$

and we can also write α_i as

$$\alpha_i = \frac{\mathbf{r}_{i-1}^T \mathbf{r}_{i-1}}{\mathbf{p}_i^T A \mathbf{p}_i}.$$

The conjugate gradient algorithm is summarised in Algorithm 2 below.

```
**Input**: The matrix $A$, vector $b$ and $x_0$, an initial guess.
**Output**: The solution $x$.
$i = 0$
$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$
while $\mathbf{r}_i \neq 0$ do
    $i = i + 1$
    if $i = 1$ then
        $\mathbf{p}_1 = \mathbf{r}_0$
    else
        $\beta_i = \frac{\mathbf{r}_{i-1}^T \mathbf{r}_{i-1}}{\mathbf{r}_{i-2}^T \mathbf{r}_{i-2}}$
        $\mathbf{p}_i = \mathbf{r}_{i-1} + \beta_i \mathbf{p}_{i-1}$
    end
    $\alpha_i = \frac{\mathbf{r}_{i-1}^T \mathbf{r}_{i-1}}{\mathbf{p}_i^T A \mathbf{p}_i}$
    $\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i \mathbf{p}_i$
    $\mathbf{r}_i = \mathbf{r}_{i-1} - \alpha_i A \mathbf{p}_i$
end
$\mathbf{x} = \mathbf{x}_i$
```

Algorithm 2: Conjugate Gradient Algorithm

For easier computation we compute $v = A\mathbf{p}_i$ and $w = \mathbf{r}_i^T \mathbf{r}_i$ for each iteration, as well as stopping the iteration for a given error, ϵ, or a maximum iteration value i_{max}. This gives us a practical implementation of the algorithm, given in pseudocode in Algorithm 3.

```
Input: The matrix $A$, vector $b$, maximum allowed iterations $i_{max}$ and
        $x_0$, an initial guess.
Output: The solution $x$.
$i = 0$
$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$
$w_0 = \mathbf{r}^T\mathbf{r}$
while ($\sqrt{w_i} > \epsilon \|\mathbf{b}\|$) and ($i < i_{max}$) do
    $i = i + 1$
    if $i = 1$ then
        $\mathbf{p} = \mathbf{r}$
    else
        $\beta = \frac{w_{i-1}}{w_{i-2}}$
        $\mathbf{p} = \mathbf{r} + \beta\mathbf{p}$
    end
    $\mathbf{v} = A\mathbf{p}$
    $\alpha = \frac{w_{i-1}}{\mathbf{p}^T\mathbf{v}}$
    $\mathbf{x} = \mathbf{x} + \alpha\mathbf{p}$
    $\mathbf{r} = \mathbf{r} - \alpha\mathbf{v}$
    $w_i = \mathbf{r}^T\mathbf{r}$
end
```

Algorithm 3: Conjugate Gradient Algorithm : Practical Implementation

3.4 Algorithms Implemented: FGP

We have implemented an iterative method to calculate the RBF interpolant, developed by Faul, Goodsell and Powell [FGP05]. This implementation is specifically for the multiquadric basis function, but the main ideas, as seen in [Pow05], can be applied to a general radial basis function of order m. Note that this iterative procedure merely provides a faster method for large systems than the direct solution of the RBF interpolant, and was not developed for solving ill-conditioned systems. For current research into stable computation of ill-conditioned systems, see [FZ07, FP07, Pir07].

We first define a scalar product on the linear space S that contains the interpolant. Then we briefly explain how the steps of an iteration are calculated, using ideas from the conjugate gradient method. From this explanation the ideas of a linear operator A and preconditioning arise and we address these concepts. Lastly we consider the practical implementation, as well as giving a version of the algorithm in pseudo-code.

3.4.1 Semi-norms and scalar products

We define S as the N-dimensional linear space containing elements of the form

$$t(\mathbf{x}) = \sum_{i=1}^{N} \mu_i \phi(\|\mathbf{x} - \mathbf{y}_i\|) + \sum_{i=1}^{\hat{m}} \delta_i p_i(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^d, \tag{3.4.1}$$

where N is a finite positive integer, the data points $\mathbf{y}_i$ are Π_m^d-unisolvent for m the order of ϕ, a conditionally positive definite function, and the coefficients μ_i satisfy $\sum_{i=1}^{N} \mu_i p_j(\mathbf{y}_i) = 0$, $j = 1, \ldots, \hat{m}$. According to this definition the element (2.1.6) is also in S. Now we define the scalar product between two elements of S as

$$\langle s, t \rangle = \sigma \sum_{i=1}^{n} \sum_{j=1}^{N} \lambda_i \phi(\|\mathbf{x}_i - \mathbf{y}_j\|) \mu_j, \; s, t \in S, \tag{3.4.2}$$

where $\sigma = (-1)^m$. Hereby it is easy to show that $\langle s, t \rangle = \langle t, s \rangle$. Since for $s \in S$

$$\langle s, s \rangle = \sigma \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \phi(\|\mathbf{x}_i - \mathbf{x}_j\|) \lambda_j = \sigma \boldsymbol{\lambda}^T \Phi \boldsymbol{\lambda}, \tag{3.4.3}$$

where Φ is the matrix as in (2.1.5), $\langle s, s \rangle$ is nonnegative and we can define the semi-norm

$$\|s\| = \sqrt{\langle s, s \rangle}, \; s \in S. \tag{3.4.4}$$

Thus $\|s\| = 0$ if and only if $\boldsymbol{\lambda} = \mathbf{0}$. A useful identity for the scalar product can be derived by

$$\begin{aligned} \langle s, t \rangle &= \sigma \sum_{i=1}^{n} \sum_{j=1}^{N} \lambda_i \phi(\|\mathbf{x}_i - \mathbf{y}_j\|) \mu_j \\ &= \sigma \sum_{i=1}^{n} \lambda_i \sum_{j=1}^{N} \mu_j \phi(\|\mathbf{x}_i - \mathbf{y}_j\|) \\ &= \sigma \sum_{i=1}^{n} \lambda_i \left\{ \sum_{j=1}^{N} \mu_j \phi(\|\mathbf{x}_i - \mathbf{y}_j\|) + \sum_{i=1}^{\hat{m}} \delta_i p_i(\mathbf{x}_i) \right\} \\ &= \sigma \sum_{i=1}^{n} \lambda_i t(\mathbf{x}_i). \end{aligned}$$

In the same way we can show that

$$\langle s, t \rangle = \sigma \sum_{j=1}^{N} \mu_j s(\mathbf{y}_j). \tag{3.4.5}$$

This gives us the advantage that if s is a general element of S and s^* is the required interpolant, then

$$\begin{aligned}\langle s, s^* \rangle &= \sigma \sum_{i=1}^{n} \lambda_i s^*(\mathbf{x}_i) \\ &= \sigma \sum_{i=1}^{n} \lambda_i f_i.\end{aligned}$$

Note that this means that we can calculate $\langle s, s^* \rangle$ while the coefficients of s^* are still unknown. Also scalar products can be calculated without the computation of any matrix-vector products.

3.4.2 Calculating the interpolant

The procedure will iteratively construct $s_{k+1} \in S$ from $s_k \in S$, where $s^* \in S$ is the required interpolant satisfying $s^*(\mathbf{x}_i) = f_i$, $i = 1, \ldots, n$ and $s_1 \equiv 0$. This is done by applying a linear operator A to elements in S, where the operator is defined in Subsection 3.4.3. For each iteration $k \geq 1$, s_{k+1} is the element of S_k that minimizes $\|s_{k+1} - s^*\|$, where S_k is the linear subspace of S spanned by $A^j s^*$, $j = 1, \ldots, k$. We want to choose the operator A in such a way that the semi-norms $\|s^* - s_i\|$, $i = 1, \ldots, k$, decrease strictly monotonically. Also the iteration must terminate if $\|s^* - s_k\| = 0$.

This method is similar to the conjugate gradient using A as a preconditioner. For each iteration $k \geq 1$, we calculate s_{k+1} by

$$s_{k+1} = s_k + \gamma_k d_k, \tag{3.4.6}$$

where γ_k is the 'step-length' for which $\|s^* - s_{k+1}\|$ is a minimum, and $d_k \in S$ is a 'search direction' chosen from the Krylov subspace spanned by $A^j s^*$, $j = 1, \ldots, k$ such that $\langle d_k, d_j \rangle = 0$ for $j = 1, \ldots, k-1$, $k \geq 2$. Similar to the conjugate gradient iteration,

$$\gamma_k = \frac{\langle d_k, s_* - s_k \rangle}{\langle d_k, d_k \rangle}. \tag{3.4.7}$$

The search direction

$$d_k = A(s^* - s_k) - \beta_k d_{k-1}, \tag{3.4.8}$$

except for $k = 1$ in which case $d_1 = As^*$ since $s_1 \equiv 0$. If we define $t_k = A(s^* - s_k)$, the conditions $\langle d_k, d_j \rangle = 0$ for $j = 1, \ldots, k-1$ imply that

$$\beta_k = \frac{\langle t_k, d_{k-1} \rangle}{\langle d_{k-1}, d_{k-1} \rangle}. \tag{3.4.9}$$

The iteration stops when

$$|s_{k+1}(\mathbf{x}_i) - f_i| \leq \epsilon, \quad i = 1, \ldots, n, \tag{3.4.10}$$

where ϵ is a given tolerance.

3.4.3 The operator A

As mentioned in Subsection 3.4.2, we choose the operator A in order for the semi-norms $\|s^* - s_i\|$, $i = 1, \ldots, k$, to decrease strictly monotonically. For the procedure mentioned in Subsection 3.4.2 to give this result, it is shown in [FP00] that it is sufficient if the operator A from S to S has the properties

- $s \in S,\ s \neq 0\ \Rightarrow As \neq 0$ (nonsingularity)
- $s \in \Pi_{m-1}\ \Rightarrow As = s$ (polynomial reproduction)
- $s \in S, s \in \Pi_{m-1}\ \Rightarrow \langle s, As\rangle > 0$ (ellipticity)
- $s, t \in S\ \Rightarrow \langle t, As\rangle = \langle At, s\rangle$ (self-adjointness)

Recall from the definition of the semi-norm (3.4.4) that for $s \in S$, $\|s\| = 0$ implies $s \in \Pi_{m-1}$. The nonsingularity condition is required since we can only calculate elements of the set As, $s \in S$. Since the semi-norm ignores polynomial terms in s, the polynomial reproduction property is needed as a way to calculate the polynomial part of s^*. Ellipticity is required to ensure that $s_{k+1} \neq s_k$ and it can be shown by induction that the self-adjointness condition ensures that $\langle d_k, d_j\rangle = 0$ for $j = 1, \ldots, k-1$ for k in the interval $[2, k^*-1]$ and d_k as in (3.4.8).

The chosen operator satisfying the properties listed above is defined by the equation

$$As = \sum_{i=1}^{n} \frac{\langle z_i, s\rangle}{\langle z_i, z_i\rangle} z_i, \quad s \in S, \tag{3.4.11}$$

where z_i, $i = 1, \ldots n$ is a basis of S. If A were the identity operator then our iterative method would converge in one iteration, since $d_1 = As^* = s^*$ and from (3.4.6), $s_2 = \alpha_1 s^*$, where $\alpha = 1$ would give us the required interpolant. This is equivalent to requiring that the basis functions satisfy

$$\langle z_i, z_j\rangle = 0, \quad 1 \leq i < j \leq n. \tag{3.4.12}$$

3.4.4 Preconditioning using approximate Lagrange functions

For the rest of the discussion we assume that ϕ has order $m = 1$. The orthogonality conditions (3.4.12) lead us to choosing z_i, $i = 1, \ldots,\ n$ as Lagrange functions. Specifically for $\ell = 1, 2, \ldots, n$ we define $\hat{z}_\ell$ as the function

$$\hat{z}_\ell(\mathbf{x}) = \sum_{i=\ell}^{n} \hat{\zeta}_{i\ell}\phi(\|\mathbf{x} - \mathbf{x}_i\| + \alpha_\ell, \quad \mathbf{x} \in \mathbb{R}^d, \tag{3.4.13}$$

satisfying

$$\begin{aligned} \hat{z}_\ell(\mathbf{x}_i) &= \delta_{i\ell}, \quad i = \ell, \ell+1, \ldots, n. & (3.4.14) \\ \sum_{i=\ell}^{n} \hat{\zeta}_{\ell i} &= 0. & (3.4.15) \end{aligned}$$

Now the scalar product between two of these functions

$$\langle \hat{z}_k, \hat{z}_\ell \rangle = \sigma \sum_{i=k}^{n} \sum_{j=\ell}^{n} \hat{\zeta}_{ki} \hat{\zeta}_{\ell j} \phi(\|\mathbf{x}_i - \mathbf{x}_j\|) = \sigma \sum_{i=k}^{n} \hat{\zeta}_{ki} \hat{z}(\mathbf{x}_i), \tag{3.4.16}$$

is zero for $k > \ell$ since then $\hat{z}_\ell(\mathbf{x}_i)$ is zero by equation (3.4.14). So by choosing z_i, $i = 1, \ldots n$, in this way it satisfies the conditions $\langle \hat{z}_k, \hat{z}_\ell \rangle = 0$ for $1 \le k < \ell \le n$.

Unfortunately computing all the $\hat{\zeta}$ values is unfeasible since it requires as much work as solving the interpolation problem itself. Therefore Powell et al [FGP05] uses approximations to the $\hat{z}_\ell$ instead, where each approximation depends only on the positions of $q << n$ datapoints. Specifically, for each integer ℓ where $\ell = 1, 2, \ldots, n$, a set $\mathcal{L}_\ell$ is formed. For $\ell \le n - q$, $\mathcal{L}_\ell$ contains the indexes of the q points closest to $\mathbf{x}_\ell$, including ℓ itself. In other words $\mathcal{L}_\ell$ contains the q integers from $i = \ell, \ldots, n$ that minimize the distances $\|\mathbf{x}_i - \mathbf{x}_\ell\|$, $i \in \mathcal{L}_\ell$. For $\ell > n - q$, $\mathcal{L}_\ell$ contains the integers $\ell, \ell+1, \ldots, n$.

Now we define functions z_ℓ as

$$z_\ell(\mathbf{x}) = \sum_{i \in \mathcal{L}_\ell} \zeta_{\ell i} \phi(\|\mathbf{x} - \mathbf{x}_i\|), \ \mathbf{x} \in \mathbb{R}^d, \tag{3.4.17}$$

meaning $z_\ell \in S$ with a constant coefficient of zero. The coefficients $\zeta_{\ell i}$, $i \in \mathcal{L}_\ell$ are calculated to satisfy

$$\begin{aligned} z_\ell(\mathbf{x}_i) &= \delta_{i\ell}, \quad i \in \mathcal{L}_\ell & (3.4.18) \\ \sum_{i \in \mathcal{L}_\ell} \zeta_{\ell i} &= 0. & (3.4.19) \end{aligned}$$

3.4.5 The FGP Algorithm

In this section we give more details about how the method is implemented, for example by deriving efficient expressions that minimize matrix-vector multiplications.

We define the elements of S used in the iterations as

$$\begin{aligned} s^k(\mathbf{x}) &= \sum_{i=1}^{n} \lambda_{ki}\phi(\|\mathbf{x}-\mathbf{x}_i\|) + \alpha_k \\ d^k(\mathbf{x}) &= \sum_{i=1}^{n} \delta_{ki}\phi(\|\mathbf{x}-\mathbf{x}_i\|) + \text{constant} \\ t^k(\mathbf{x}) &= \sum_{i=1}^{n} \tau_{ki}\phi(\|\mathbf{x}-\mathbf{x}_i\|) + \text{constant} \\ z^k(\mathbf{x}) &= \sum_{i=1}^{n} \zeta_{ki}\phi(\|\mathbf{x}-\mathbf{x}_i\|) + \text{constant} \end{aligned}$$

where $\mathbf{x} \in \mathbb{R}^d$ and the constant terms are negligible except for s^k since the scalar product ignores polynomial terms. We also use a residual vector $\mathbf{r}_k$, defined per component as

$$r_{ki} = f_i - s_k(\mathbf{x}_i),\ i = 1, \ldots, n. \tag{3.4.20}$$

We define $t_k = A(s^* - s_k)$, as in the first term of (3.4.8). Then

$$t_k(\mathbf{x}) = \sum_{\ell=1}^{n} \frac{\langle z_\ell, s^* - s_k\rangle}{\langle z_\ell, z_\ell\rangle} z_\ell(\mathbf{x}) = \sum_{\ell=1}^{n} \mu_{k\ell} z_\ell(\mathbf{x}),\ \mathbf{x} \in \mathbb{R}^d, \tag{3.4.21}$$

and by (3.4.5)

$$\mu_{k\ell} = \frac{\langle z_\ell, s^* - s_k\rangle}{\langle z_\ell, z_\ell\rangle} = \frac{\sum_{i=1}^n \zeta_{\ell i} r_{ki}}{\sum_{i=1}^n \zeta_{\ell i} z_\ell(\mathbf{x}_i)},\quad \ell = 1, \ldots, n, \tag{3.4.22}$$

which because of (3.4.18) reduces to

$$\mu_{k\ell} = \frac{\sum_{i\in\mathcal{L}_\ell} \zeta_{\ell i} r_{ki}}{\zeta_{\ell\ell}},\ \ell = 1, \ldots, n. \tag{3.4.23}$$

Further by (3.4.5) we can write (3.4.7) as

$$\gamma_k = \frac{\sum_{i=1}^n \delta_{ki} r_{ki}}{\sum_{i=1}^n \delta_{ki} d_k(\mathbf{x}_i)}, \tag{3.4.24}$$

and also (3.4.9) becomes

$$\beta_k = \frac{\sum_{i=1}^n \tau_{ki} d_{k-1}(\mathbf{x}_i)}{\sum_{i=1}^n \delta_{ki} d_{k-1}(\mathbf{x}_i)}. \tag{3.4.25}$$

It is more efficient in the algorithm to do calculations on the coefficients of elements in the space S instead of computing the function values by a matrix vector multiplication. The only such multiplication in the iteration is the calculation of the function d_k. In a more efficient implementation of the FGP algorithm, this computational bottleneck can be replaced by a Fast Multipole method.

It can be seen from (3.4.6) that the coefficients of s_{k+1} can be written as

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \gamma_k \boldsymbol{\delta}_k, \tag{3.4.26}$$

and also from (3.4.8) we can compute the coefficients of d_k by

$$\boldsymbol{\delta}_k = \boldsymbol{\tau}_k - \beta_k \boldsymbol{\delta}_{k-1}. \tag{3.4.27}$$

The constant α_k is taken at each iteration as the average of the largest and smallest component of the current residual vector, and this value is then added to the previous constant, α_{k-1}. The algorithm is given in pseudo-code in Algorithm 4.

3.5 Remarks

When applying RBF interpolation to large datasets, three major issues arise. Firstly, loading the full dataset into memory is unfeasible once the dataset is larger than 10000 points. Loading the data as needed from a file is a possible solution.

Also, we need to evaluate the interpolant at a large number of points. This means we need to efficiently compute matrix-vector products. The fast multipole method is successful in reducing the computational cost to $\mathcal{O}(n\log n)$.

Lastly, we need algorithms to compute the RBF coefficients efficiently. Multilevel and FGP methods are improvements we considered. The FGP method however needs to be able to efficiently compute matrix-vector products. This again needs efficient matrix-vector multiplication, solved by fast multipole methods.

We implemented the FGP algorithm for any RBF with a constant polynomial part. Further improvements include using fast multipole methods for the matrix-vector products, as well as loading data on demand.

```
Input: For $i = 1, \ldots, n$ the values $f_i$ and datapoints $\mathbf{x}_i$ as well as the
       integer $q$, the multiquadric parameter $c$, error tolerance $\epsilon$ and
       maximum number of iterations $N_{max}$.
Output: The coefficients $\boldsymbol{\lambda}$ as well as the constant $\alpha$.
Initialization :
$\boldsymbol{\lambda} = \mathbf{0}$
$\alpha = \frac{\min(f_1,\ldots,f_n)+\max(f_1,\ldots,f_n)}{2}$
$\mathbf{r}$ contains $n$ components $r_i = f_i - \alpha$ for $i = 1, \ldots, n$
A random ordering, $\boldsymbol{\Omega}$, of the indices $\{1, \ldots, n\}$ is generated
Preprocessing :
for $m = 1, \ldots, n$ do
    Find $\mathcal{L}_{\Omega_m}$
    $\ell = \Omega_m$
    Calculate the coefficients $\boldsymbol{\zeta}_\ell$ by solving
    $\sum_{i \in \mathcal{L}_\ell} \zeta_{\ell i} \phi(\|\mathbf{x} - \mathbf{x}_i\|) = \delta_{i\ell}, \quad i \in \mathcal{L}_\ell$
end
Iteration :
error $= \max(|r_1|, \ldots, |r_n|)$
$k = 0$
while error $> \epsilon$ and $k < N_{max}$ do
    $k = k + 1$
    $\boldsymbol{\tau} = 0$
    for $m = 1, \ldots, n$ do
        $\ell = \Omega_m$
        $\boldsymbol{\tau}_m = \mathbf{0}$
        $\mu = \frac{\sum_{i \in \mathcal{L}_\ell} \zeta_{\ell i} r_i}{\zeta_{\ell\ell}}$
        $\tau_{mj} = \mu \zeta_{\ell j}, \; j \in \mathcal{L}_\ell$
        $\boldsymbol{\tau} = \boldsymbol{\tau} + \boldsymbol{\tau}_m$
    end
    if $k = 1$ then
        $\boldsymbol{\delta} = \boldsymbol{\tau}$
    else
        $\beta = \frac{\boldsymbol{\tau}^T \mathbf{d}}{\boldsymbol{\delta}^T \mathbf{d}}$
        $\boldsymbol{\delta} = \boldsymbol{\tau} - \beta \boldsymbol{\delta}$
    end
    $\mathbf{d} = \Phi \boldsymbol{\delta}$
    $\gamma = \frac{\boldsymbol{\delta}^T \mathbf{r}}{\boldsymbol{\delta}^T \mathbf{d}}$
    $\mathbf{r} = \mathbf{r} - \gamma \mathbf{d}$
    error $= \max(|r_1|, \ldots, |r_n|)$
    $\alpha_k = \frac{\min(r_1,\ldots,r_n)+\max(r_1,\ldots,r_n)}{2}$
    $\alpha = \alpha + \alpha_k$
    $\mathbf{r} = \mathbf{r} - \alpha_k \mathbf{1}$
    $\boldsymbol{\lambda} = \boldsymbol{\lambda} + \gamma \boldsymbol{\delta}$
end
```

Algorithm 4: FGP Algorithm

Chapter 4

Applications and Experiments

In this chapter we focus on some applications of radial basis function interpolation. We apply RBF interpolation to data on a half-ellipsoid, and experiment with the local effect on the interpolant when modifying some points. We also apply RBF interpolation to topological (GIS) data and 3D face data as well as an application in image warping.

All three dimensional rendering in this chapter is done using the Python class *Mat3d*, developed for this thesis and discussed in Appendix A.1. The RBF interpolants were calculated using the Python software developed for this purpose, see Appendix B.

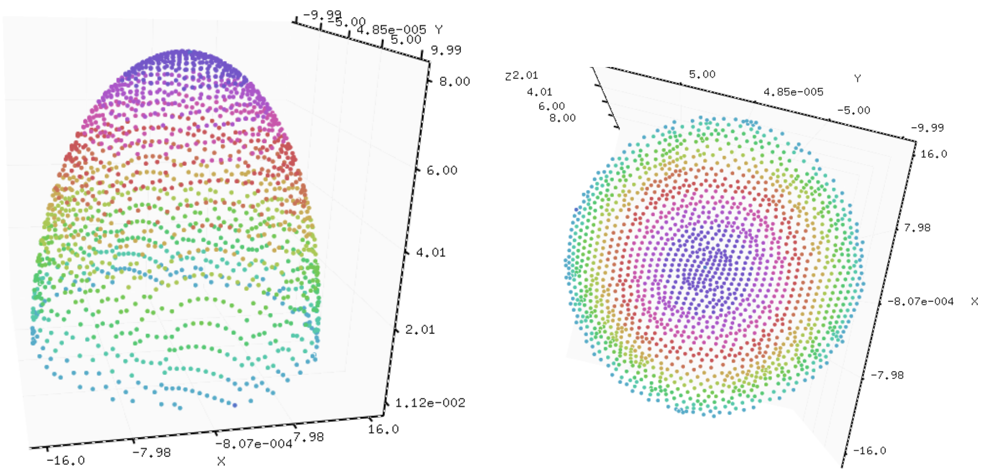
4.1 3D Experiments

4.1.1 Half-ellipsoid

Here we reconstruct a half-ellipsoid in three dimensions. This is a function $f : \mathbb{R}^2 \to \mathbb{R}$, directly amenable to RBF interpolation. We use the data shown in Figure 4.1, consisting of 1722 x,y data points with corresponding function values (heights). It can be seen from Figure 4.1 that the dataset is not on a regular grid in the x, y plane.

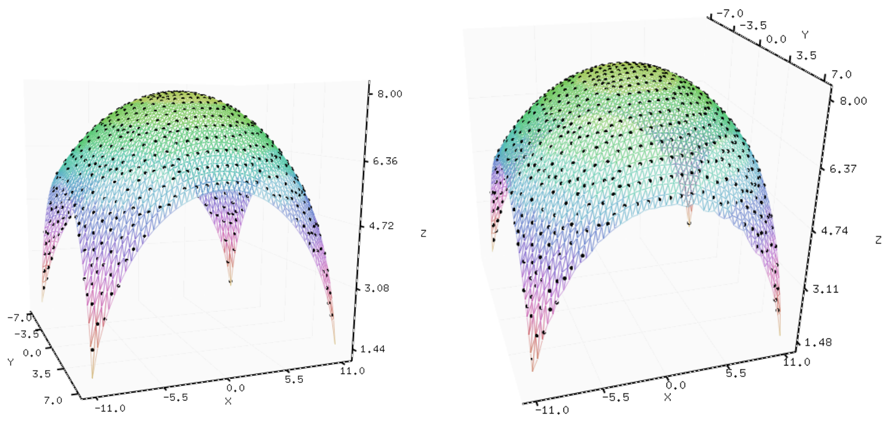
Our goal is to interpolate this dataset with the different radial basis functions at our disposal, given in Table 2.1. In the case of the polyharmonic splines, we only use the linear, thin plate spline and cubic functions. Those are the functions $\phi(r) = r$, $\phi(r) = r^2 \log r$ and $\phi(r) = r^3$ respectively. We select a rectangular grid of points in the x, y plane lying within the dataset. In this way all points lie inside the area for which we have data, avoiding strange behaviour due to extrapolation. The shape parameter used for the interpolants was $c = 1$, except for the Gaussian for which different shape parameters were used.

The results for all the radial basis functions tested, can be seen in Figures 4.2 to 4.7. The black dots indicate the original dataset and the coloured mesh



(a) Half-ellipsoid data points (b) Positions of data points

Figure 4.1: Half-ellipsoid data

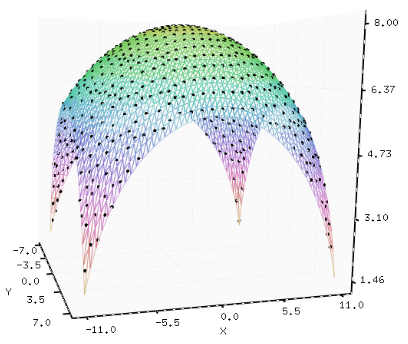


(a) Linear (b) Multiquadric

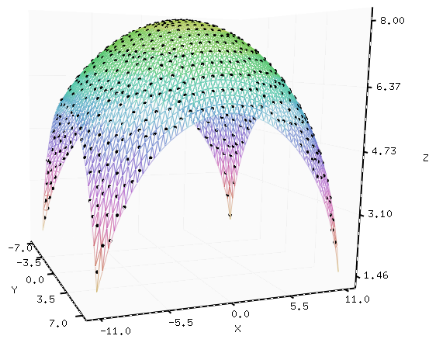
Figure 4.2: Half-ellipsoid interpolants

is the constructed interpolant. It can be seen from the figures that smooth interpolants are constructed that fit the given data. Even varying the shape parameter does not degrade the quality of the interpolants.

The Gaussian on the other hand is very sensitive to the value of the shape parameter. In Figures 4.5, 4.6 and 4.7 the resulting interpolants can be seen for shape parameters $c = 1$, 1.1, 1.2, 1.3, 1.4, 1.5. The 'best' interpolant seems to be $c = 1.3$, smaller c's suffer from poor conditioning while larger c's lead to interpolants with larger variation.

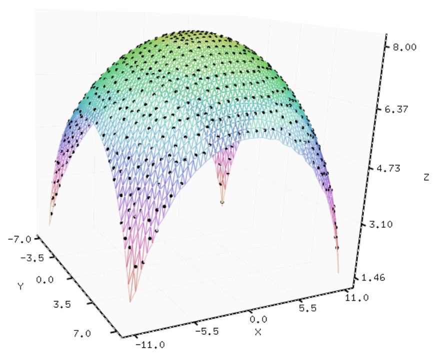


(a) Thin plate spline

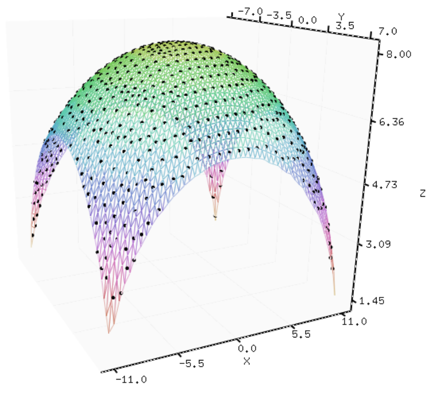


(b) Cubic

Figure 4.3: Half-ellipsoid interpolants 2



(a) Inverse Multiquadric



(b) Inverse Quadratic

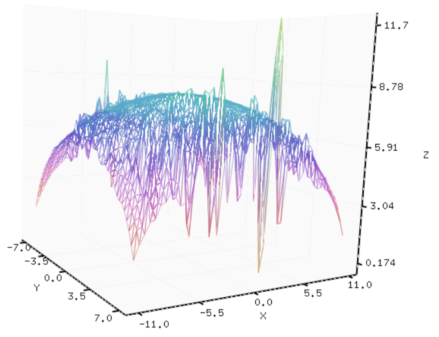
Figure 4.4: Half-ellipsoid interpolants 3

4.1.2 Modified half-ellipsoid

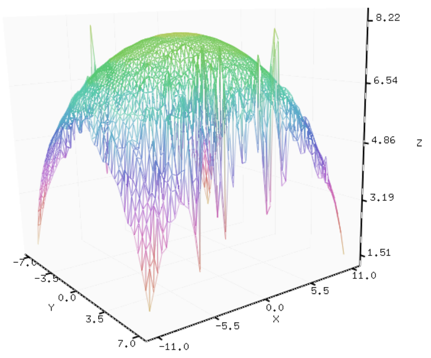
Here we test the effect on the interpolant if a select few data points are moved. Two clusters of points are ‘pulled out’ of the surface of the old ellipsoid data, forming two ‘peaks’ on the surface, as shown in Figure 4.8. We test the same radial basis functions as before and evaluate the interpolants on the same set of data as the original ellipsoid. The plots are shown approximately from the same angle.

In Figures 4.9, 4.10 and 4.11 the resulting interpolants for the polyharmonic splines tested are shown. The linear interpolant shows only a local effect in the regions of the ‘peaks’. The thin plate spline and even more so the cubic interpolant shows oscillations where the ‘peaks’ and the normal surface of the ellipsoid meet, meaning the moved points have a more global effect on the interpolant.

Much like the cubic interpolant, the multiquadric interpolant (Figure 4.12)

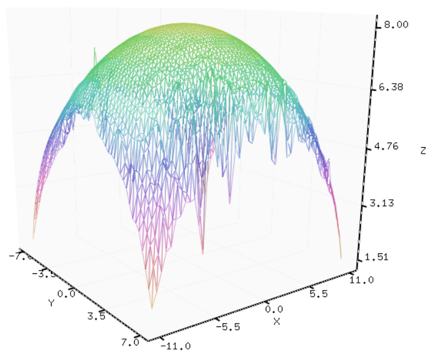


(a) $c = 1$

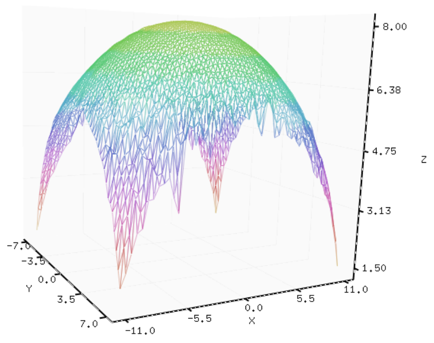


(b) $c = 1.1$

Figure 4.5: Gaussian interpolants



(a) $c = 1.2$



(b) $c = 1.3$

Figure 4.6: Gaussian interpolants 2

shows waves where the ‘peaks’ and the ellipsoid surface meet. These oscillations are less in the cases of the inverse quadratic and inverse multiquadric interpolants as seen in Figures 4.14 and 4.13. A series of Gaussian interpolants for different shape parameters was calculated, the ‘best’ looking one is shown in Figure 4.15 for which $c = 1.4$.

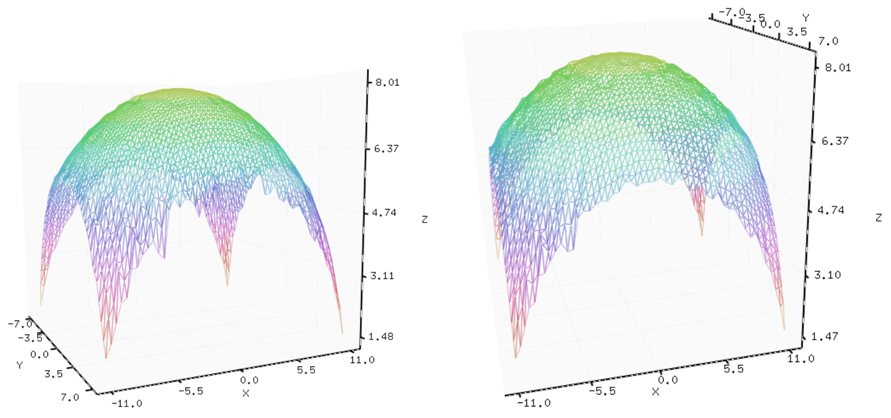

(a) $c = 1.4$ (b) $c = 1.5$

Figure 4.7: Gaussian interpolants 3

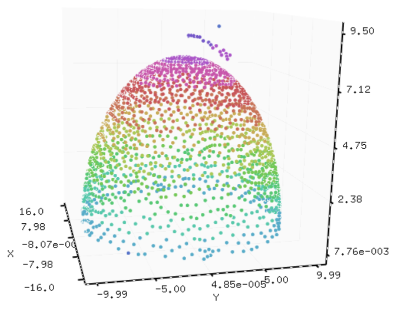


Figure 4.8: Modified half-ellipsoid data

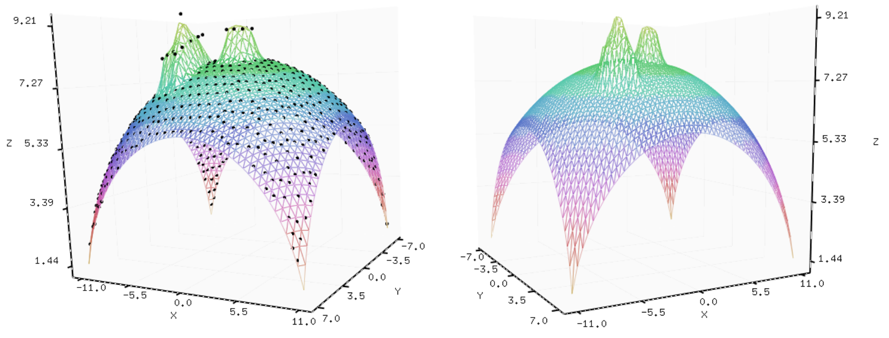
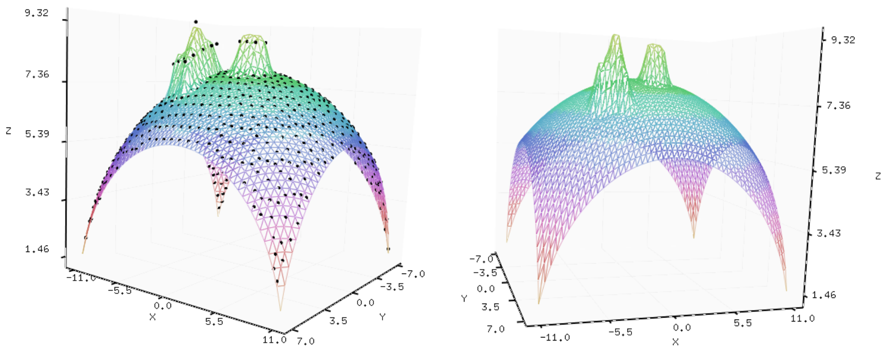
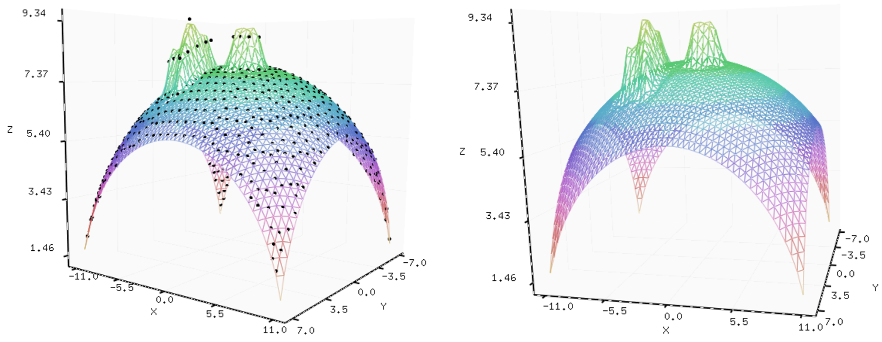

(a) Given data and interpolant (b) Interpolant

Figure 4.9: Linear RBF interpolant



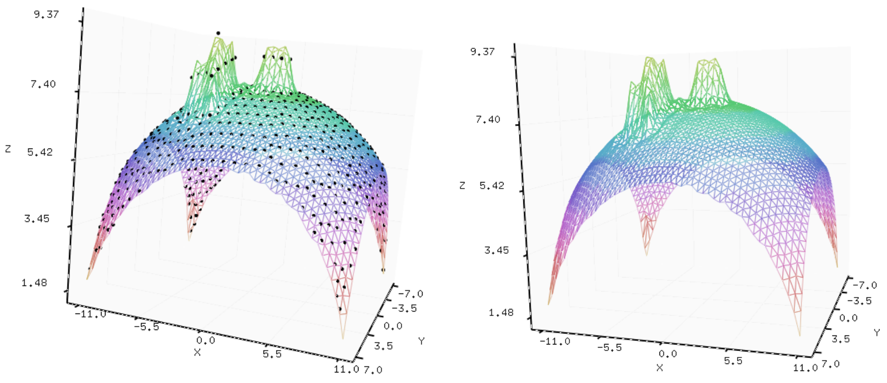
(a) Given data and interpolant (b) Interpolant

Figure 4.10: Thin Plate Spline RBF interpolant



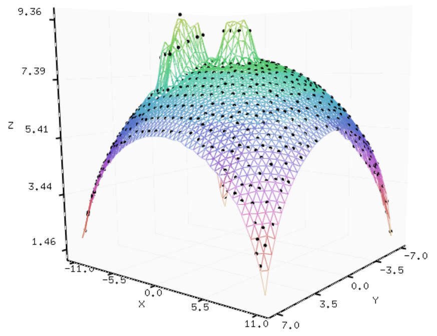
(a) Given data and interpolant (b) Interpolant

Figure 4.11: Cubic RBF interpolant

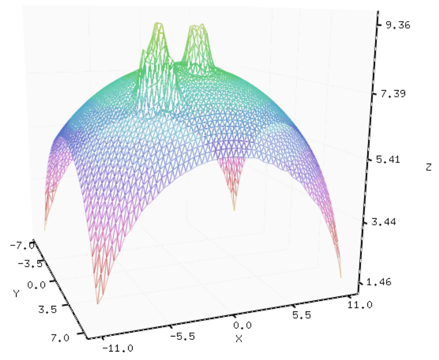


(a) Given data and interpolant (b) Interpolant

Figure 4.12: Multiquadric RBF interpolant

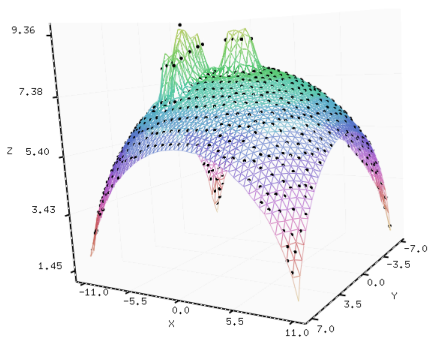


(a) Given data and interpolant

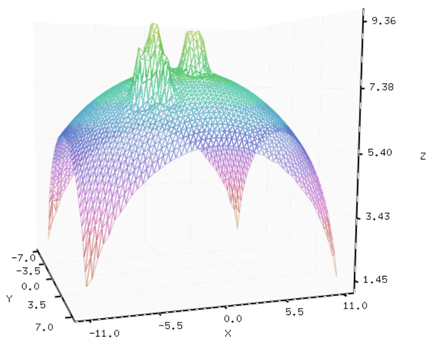


(b) Interpolant

Figure 4.13: Inverse multiquadric RBF interpolant

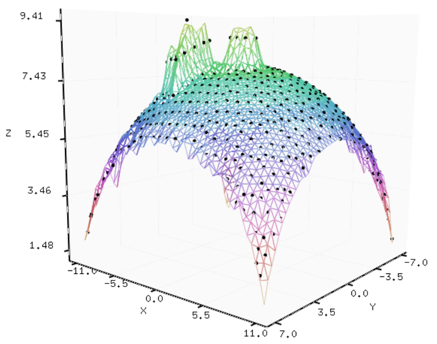


(a) Given data and interpolant

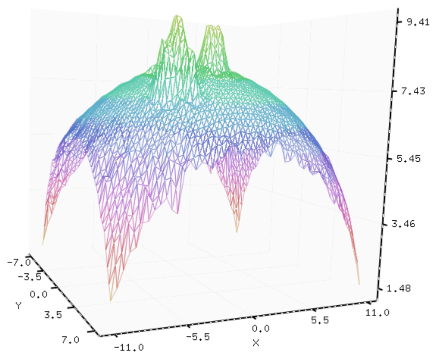


(b) Interpolant

Figure 4.14: Inverse Quadratic RBF interpolant



(a) Given data and interpolant



(b) Interpolant

Figure 4.15: Gaussian RBF interpolant ($c = 1.4$)

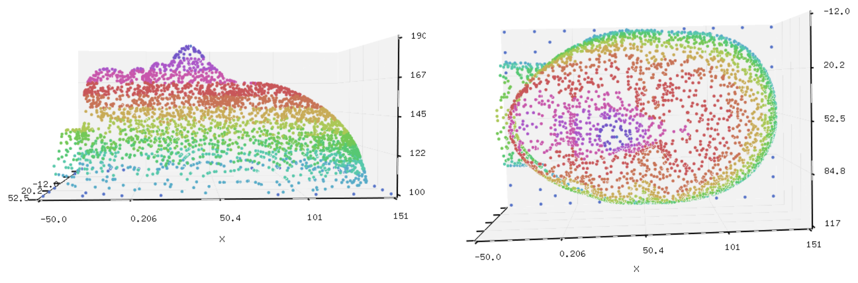

(a) (b)

Figure 4.16: Face data

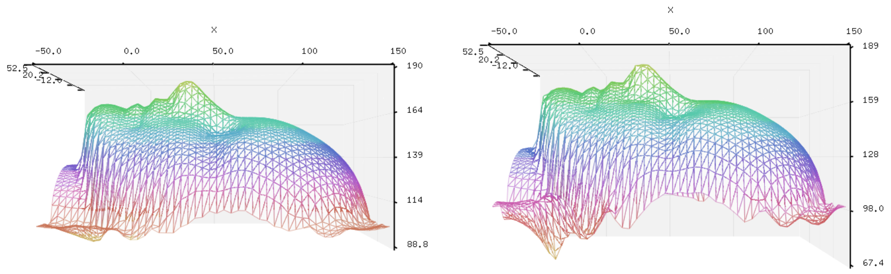

(a) Linear interpolant (b) Thin plate spline interpolant

Figure 4.17: Some interpolants on the human face data

4.1.3 Interpolants on human face data

A more visually interesting test is done on data from a human face. The dataset contains 3292 points, including extra points added on a rectangle framing the face. This is done in order for an evaluation space to be chosen on a rectangle with little effect from extrapolation. The dataset used can be seen in Figure 4.16 and the added points can clearly be seen in Figure 4.16(b).

Some of the resulting interpolants can be seen in Figure 4.17. These two figures are the best results we calculated for this dataset. The linear interpolant best fits our idea of what the interpolant should look like while the thin plate spline interpolant shows more oscillations on the part of the grid where few datapoints are available, e.g. the area away from the face.

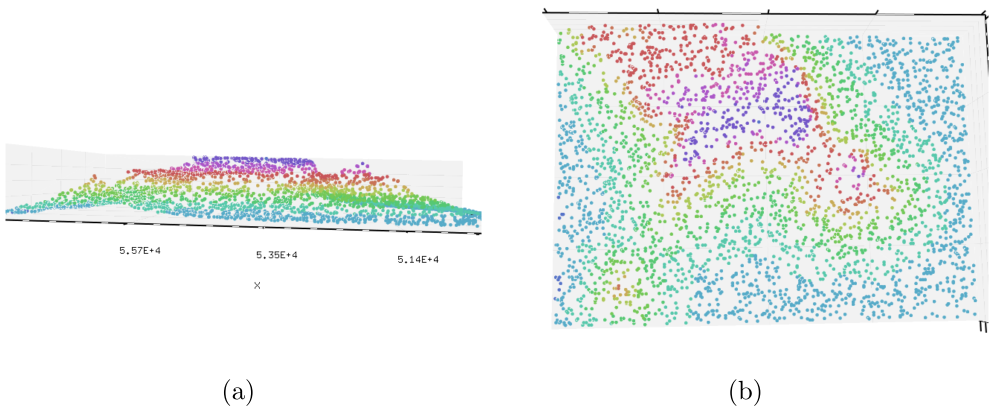


Figure 4.18: Dataset for Table Mountain used

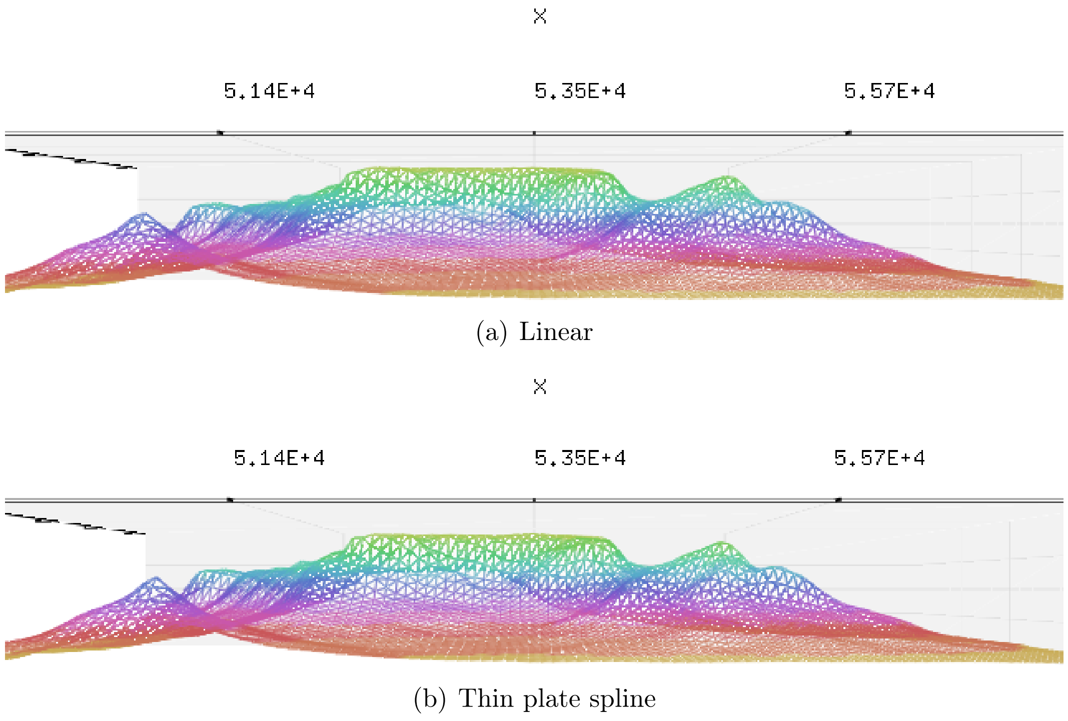
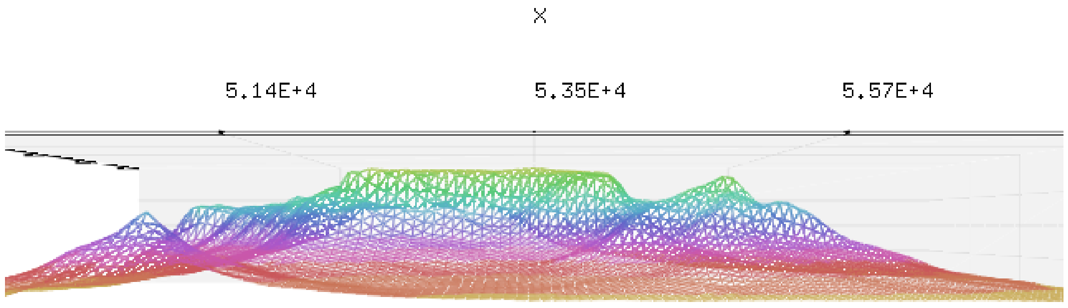


Figure 4.19: Some RBF interpolants on the dataset

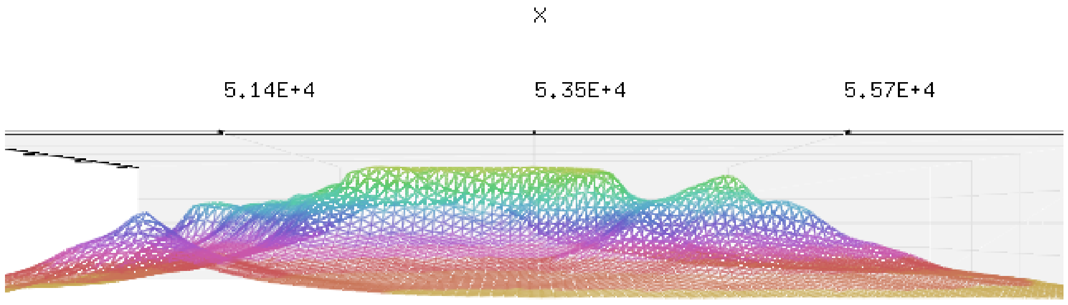
4.2 Application to GIS Data

We have used the GIS data of Table Mountain in our experiments, as this is a well known landmark in South Africa. The original data was converted to x,y,z values using the software package Global Mapper. The dataset used consists of 3388 scattered datapoints and can be seen in Figure 4.18.

We calculated the interpolants for all the radial basis functions previously



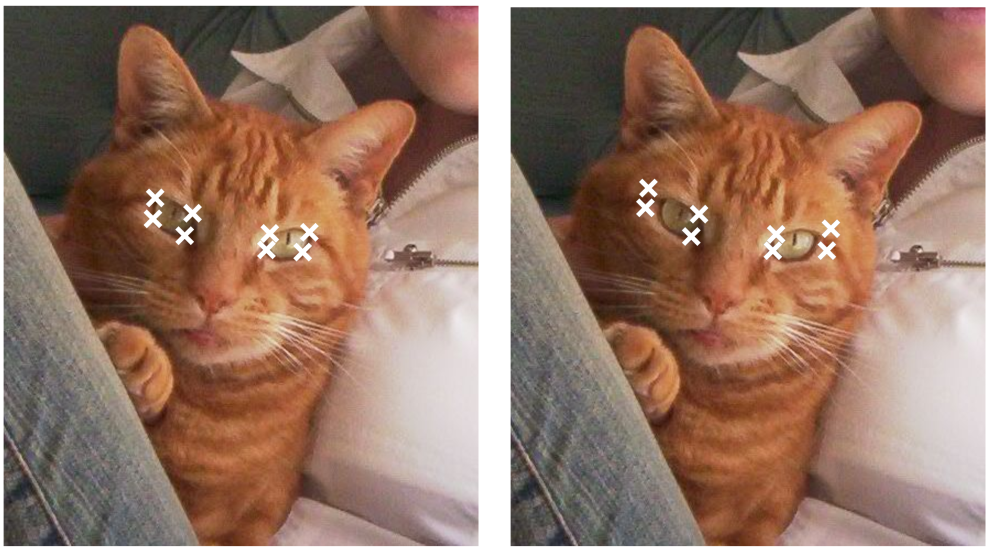
(a) Cubic



(b) Multiquadric

Figure 4.20: More RBF interpolants on the dataset

investigated. The best results obtained can be seen in Figure 4.20. These interpolants all look very similar, and sufficiently match the landmark's data.



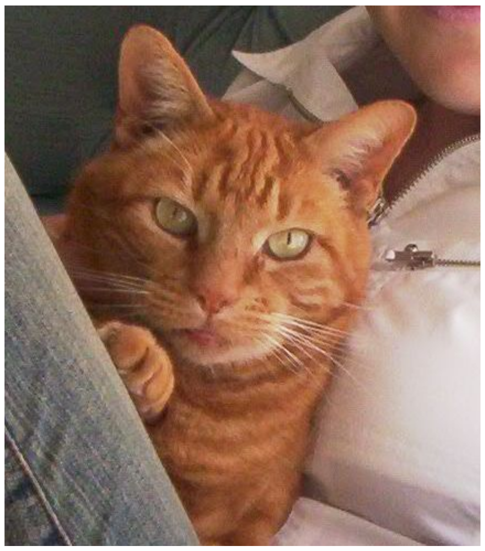
(a) Source Points (b) Destination Points

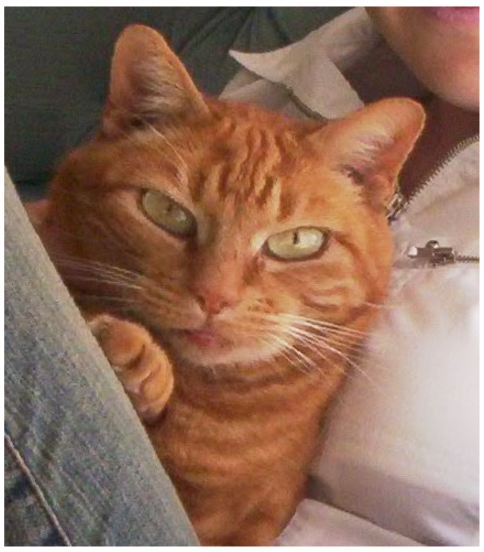
Figure 4.21: Image with source and destination points marked.

4.3 Image Warping

As seen in [ADRY94], RBF interpolation can be used in an image warping application. Since constructing a warped image from an original one is a mapping from $\mathbb{R}^2 \to \mathbb{R}^2$, we construct a pair of interpolants $\mathbb{R}^2 \to \mathbb{R}$. This means that given two sets of 2-dimensional data, mapping points of the form (x, y) to (i, j), we form interpolants on this data of the form $f(x, y) = i$ and $g(x, y) = j$. For example in the image in Figure 4.21, we chose the corners of the cat's eyes and the four corners of the image as data. We want to move the eye-corner points to the new locations as shown in 4.21(b), while keeping the corners of the image in the same positions. We perform RBF interpolation only on these 12 points, meaning the interpolant is a measure of how pixels should move to new locations in the warped image.

A smoothing factor is added to the method to provide a trade-off between the interpolation error and the warping on the image. We refer the reader to [ADRY94] for further details. In this application, a warping effect close to the datapoints, with as little change to the rest of the image as possible, is desirable. Using the given mapping in Figure 4.21, we warped the original image using thin plate spline and Gaussian RBF's. We chose a relatively 'thin' Gaussian shape parameter in order to have only local changes in the image. The difference between the warped image for this choice and the thin plate spline can be seen in Figure 4.22. The Gaussian interpolant moves only the pixels close to the eyes whereas in the thin plate spline interpolant, the entire image is warped. This can be seen in the way that the cat is stretched


(a) Original Picture


(b) Warped Image: thin plate spline

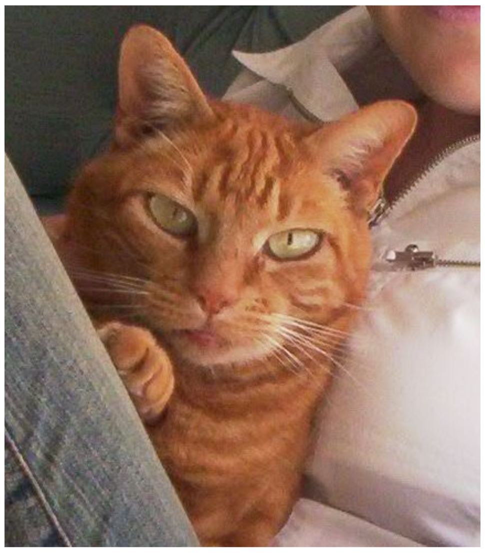

(c) Warped Image: Gaussian ($c = 0.05$)

Figure 4.22: Image warping

compared to the original picture.

The sensitivity of the Gaussian to the shape parameter, as well as its property that for 'thin' basis functions the interpolation between datapoints tends to zero, is undesirable for the surface reconstruction applications seen in this work. However it proves to be ideal for this image warping application, where the global effect exhibited by the thin plate spline and all the polyharmonic spline basis functions is in this case undesirable.

Chapter 5

Conclusion

RBF interpolation, the topic of this thesis, is one of the few ways to interpolate scattered data in many dimensions. The theory of RBF interpolation is well developed in the literature [Buh03, Wen05, IA04]. The complications of RBF interpolation lie in the implementation of the algorithms: large, non-sparse matrices; ill-conditioning and sensitivity to shape parameters. Also, the best choice of a basis function for each specific application is unclear.

In our experiments we found that Gaussian interpolation is extremely sensitive to the choice of the shape parameter. An optimal shape parameter is dependent on how close points are to each other in the given dataset. Stability and finding the optimal shape parameter is indeed one of the main research areas in the literature, e.g. [FZ07, FP07, Pir07]. In our surface fitting implementation, we found that the polyharmonic splines and multiquadric radial basis functions usually give good results. Also polyharmonic splines do not have a shape parameter, so can be used successfully with much less effort. In the image warping application, the Gaussian is preferred since it allows a localised warp.

The problem of large non-sparse matrices has been addressed by developing fast methods both for finding the coefficients of the interpolant and evaluating new points. We have considered and implemented some of these methods, although a full application using efficient methods for every stage of the interpolation process is necessary. Expanding our current RBF code to use fast methods for matrix vector multiplications like the fast multipole method is a future goal.

Finally, RBF interpolants are calculated in the same way for all dimensions. However, there are difficulties when reconstructing a solid model in three dimensions, since we cannot represent such an object as an explicit function. As seen in [CBC$^+$01] we can model such a surface implicitly with a function $f(x, y, z)$. In this way, $f(x, y, z) = 0$ for points lying on the surface, the function is negative for points lying inside the surface and positive for points lying outside the surface. This means that in order to do RBF interpolation we insert additional points for each datapoint, lying inside and outside the surface at a normal direction to the underlying surface. Methods such as marching cubes are then used to recover the implicit surface. Implementing these methods in order to build software capable of reconstructing solid objects in three dimensions is another future extension to our software.

Appendices

Appendix A

Mat3D Application

Mat3d is a Python class developed for plotting three dimensional data, allowing interactive rotation,translation and zooming in on the plot by the user. The *mat3d* application was developed since the Python packages for plotting, mainly *matplotlib*, has limited support for three dimensional plots. In this section we briefly explain the features of the program, how to use it and show a simple example including the expected output.

A.1 Required software

We developed the application using the Python programming language, hence a working installation of Python 1.4 is a requirement. The following packages and/or modules in your Python installation are required :

- NumPy
- Python's Imaging Library (PIL)
- Tk
- PyOpenGL

For more information regarding installation on different systems, see the website for the application at http://www.scipy.org/WilnaDuToit.

A.2 Features

The application is meant to have to some of the three dimensional plotting functionality of Matlab. There are four different types of plots available, each having variables such as the width of the lines, colourmaps, the precision of the numbers on the tickmarks to use and more to be set by the user. All of these plots can be moved, rotated and zoomed in or out by using the mouse, as well as saved in a variety of different image formats.

The different types of plots available are :

- *mesh* - Forms a mesh of triangulated three dimensional data. The triangles can be either filled or be wireframed.
- *plot3* - Plots lines in three dimensions for each column of the input data.
- *plot3_points* - Plots three dimensional data as a pointcloud.
- *plotrbf* - Plot developed specifically for plotting data obtained from rbf interpolation. Plots a *mesh* as well as plotting additional datapoints as black points in the graph. The purpose if this is to be able to plot the interpolant obtained as well as the original input data used on the same graph.

The mouse commands for interactive manipulation of the viewing of the plot are :

- left button - Hold down and move the mouse for translations of the plot.
- middle button - Hold down and move for rotations of the plot.
- right button - Hold down and move for scaling (zooming) of the plot.

Images can be saved by clicking on the 'Save' button in the gui window. This opens a file dialog into which a filename can be entered. Images can be saved in the following formats :

- Windows Bitmap (*.bmp)
- Enhanced Windows Metafile (*.emf)
- Encapsulated PostScript (*.eps)
- CompuServe GIF (*.gif)
- JPEG (*.jpg)
- Zsoft Paintbrush (*.pcx)
- Portable Network Graphics (*.png)
- Portable Pixelmap (*.ppm)
- Tagged Image File Format (*.tif)

A.3 User Guide

Here we explain the input variables used in the application and how to use them to generate plots. Some of these variables have default values and as such do not need to be specified when plotting data. Others, like the data to be plotted itself, have to be specified and in a certain format. The input data should be given as :

- *X,Y,Z* - This is the format used in *mesh*, *plot3* and *plotrbf.* The x,y and z-dimensional data is split into three matrices. The triangulations are formed by regarding the matrices row and column numbers as the underlying grid for the mesh. Meaning adjacent entries in the rows of the matrices plus an entry in either one entries' adjacent column form a triangle.

- *points* - This format is used in *plot3_points* and *plotrbf.* This data is just a list of vectors in three dimensions, since for the plots it is used in no connectivity information between these points is needed.

The following variables used are mainly for tweaking the look of the plots and as such have default values.

- *colors* - Specifies the colourmap used. The default colours can be seen in all the three dimensional plots in this thesis. This variable can be customized by any array consisting of twelve three dimensional vectors containing r, g, b values between zero and one.

- *linewidth* - Specifies an OpenGL variable regarding the width of lines drawn. The default is 0.1.

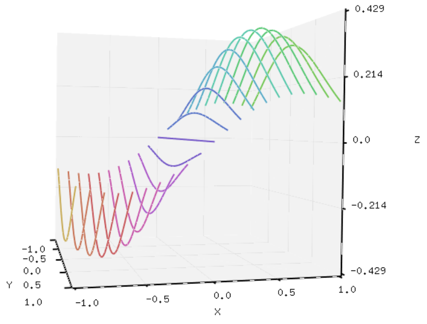
- *precision* - Specifies the number of digits precision to be used when displaying data values on the axes. The default is 3.

- *num_ticks* - Specifies the number of tickmarks on the axes. The default is 5.

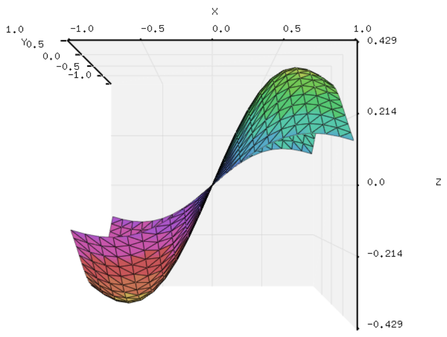
- *axiseq* - Specifies whether data is scaled proportionally. The default is 'True', meaning all data is rescaled and drawn in order for all axes to appear to have the same length.

- *fill_mesh* - Specifies whether triangles are filled in our only drawn in wireframe. The default is 0, meaning triangles are drawn in wireframe.

- *pointsize* - Specifies an OpenGL variable regarding the size of points drawn. The default is 5.



(a) Example using *plot3*



(b) Example using *mesh*

Figure A.1: Resulting plots

A.3.1 Code Examples

We show two examples, one using *plot3* and the other *mesh*. The first script results in the plot seen in Figure A.1(a). The second script generates the plot in Figure A.1(b).

```
import numpy as N
import mat3d as M

x0,x1,npts_x = -1.0,1.1,11
y0,y1,npts_y = -1.0,1.1,11

X,Y = N.mgrid[x0:x1:0.1,y0:y1:0.1]
Z = X * N.exp(-X**2 - Y**2)

M.plot3(X,Y,Z,linewidth =2.0)
```

Python Script A: Example using *plot3*

```
import numpy as N
import mat3d as M

x0,x1,npts_x = -1.0,1.1,11
y0,y1,npts_y = -1.0,1.1,11

x_delta = (x1-x0)/float(npts_x)
y_delta = (y1-y0)/float(npts_y)

X,Y = N.mgrid[x0:x1:0.1,y0:y1:0.1]
Z = X * N.exp(-X**2 - Y**2)

x = M.mesh(X,Y,Z, fill_mesh=1)
```

Python Script B: Example using *mesh*

Appendix B

RBF code

We developed Python code for the practical implementation of RBF interpolation. In this chapter we explain how the code is structured and the numerical methods implemented. Also its limitations in terms of memory and computational time is discussed as well as giving some code examples to help the user in using this code for their own experiments.

B.1 Modules and Dependencies

Two Python files are of concern here, *rbf.py* and *fgp_ iterator.py.* The former contains the *rbf* class, capable of finding the coefficients for interpolation as well as evaluating data on given coefficients for any of the radial basis functions mentioned in this work. Data of any number of dimensions is allowed, also different instances of the *rbf* class can be invoked which gives the ability to work with different RBF functions and shape parameters simultaneously.

The file *fgp_ iterator.py* contains our implementation of the FGP algorithm as discussed in Section 3.4. The iteration is split into steps by using functions used internally in the code. The only function to be used by the user is the function containing the main iteration, namely *def fgp_ iterator*, which gives as output the required coefficients. Consequently this file only performs the fitting of the RBF, in order to do evaluations the *rbf* class is needed. Note that any of the RBF functions mentioned in this work can be chosen for the iteration but a constant polynomial part will be added to the coefficient matrix regardless of the order of the chosen RBF.

In order to successfully use the above Python code, a system with a working Python 2.4 installation is required. The numerical packages Numpy and Scipy are also needed since matrix data types as well as matrix calculations are handled by functions in these packages.

B.2 Numerical Methods used

We consider only fast methods for the fitting or evaluation of RBF interpolants used in our code here. The *rbf* class solves the equation (2.1.7) for $\boldsymbol{\lambda}$ and $\boldsymbol{\gamma}$ by using a direct method, namely the *solve* function in the *scipy.linalg* (sub)package. Evaluations, meaning the matrix-vector multiplication of the left side of equation (2.1.7) is also done directly.

The iterative method in *fgp_iterator.py* was implemented as an alternative to the direct solution of the coefficients used in the *rbf* class. As the iteration performs a matrix-vector product for each iteration to calculate d_k as explained in Section 3.4.5, a fast method would be a huge improvement. The current implementation however uses a direct method for the matrix-vector product.

B.3 Capabilities and Limitations

Both *rbf.py* and *fpg_iterator.py* hold the entire matrix Φ in memory. This means that using a too large dataset or evaluating on a large set of points could lead to the program running out of memory. In both cases we have as a rule not used datasets containing more than 4000 vectors. Because of this limitation the problems regarding to computational time have not been a problem for datasets of this size.

Reading the entries of Φ as needed from a file as well as implementing fast multipole methods for matrix-vector multiplication are all improvements that can be made to the current code. Also alternative methods for the solution of the coefficients for the interpolant are possibilities to be implemented.

B.4 Examples

A simple example of using the *rbf* class to calculate multiquadric and thin plate spline interpolants to given data consisting of three points is shown in Python Script A. The interpolants are evaluated at a set of points in the region of the input data and the resulting plot is shown in Figure B.4.

Another simple example showing the usage of *fgp_iterator* can be seen in Python Script B. It chooses a random dataset consisting of 100 vectors in two dimensions with corresponding function values. The RBF coefficients for this dataset is then calculated iteratively. The resulting coefficients, and a vector containing the mean error after each iteration are among the output data from the iteration.

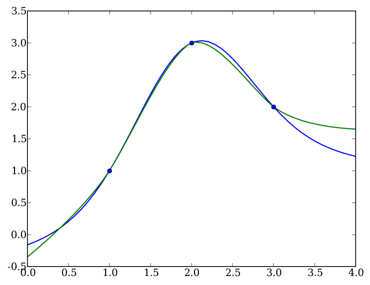


Figure B.1: Resulting graph from running Python Script A

```
import numpy as N
import rbf as R
import pylab as P                #extra package for plots

M = R.rbf('mqu',1.1)                  #define a multiquadric rbf
T = R.rbf('tps')                 #define a thin plate spline rbf

data = N.array([[1,1],[2,3],[3,2]])                #3 points input data
Mcoeff,A,f = M.getcoeff(data)                # calculate mqu coefficients
Tcoeff,A,f = T.getcoeff(data)                # calculate tps coefficients

eval_points = N.linspace(0,4,50)                #choose points to evaluate the
interpolant on
Y_M = M.evaluate(eval_points,data,Mcoeff)                #function values for mqu
Y_T = T.evaluate(eval_points,data,Tcoeff)                #function values for tps

P.plot(eval_points,Y_M)
P.plot(eval_points,Y_T)
P.plot(data[:,0],data[:,1],'ob')
P.show()                #commands to plot the interpolants and input data
```

Python Script A: Example using *rbf* class

```
import numpy as N
import fgp05_iterator as F

data = N.random.rand(2,100)*100
fdata = N.random.rand(100)

c = 1
q = 30
tol = 1e-16
Nitermax = 100
printscreen = True

lambdas,alpha,err,k,errs = F.fgp05_iterator(c,data,fdata,q,tol,Nitermax,printscreen)
```

Python Script B: Example using *fgp_ iterator*

List of References

[ADRY94] Nur Arad, Nira Dyn, Daniel Reisfeld, and Yehezkel Yeshurun. Image warping by radial basis functions: Application to facial expressions. *Computer Vision, Graphics, and Image Processing. Graphical Models and Image Processing*, 56(2):161–172, 1994.

[ARA07] ARANZ. Aranz: Fitting point clouds, 2007. [`http://www.aranz.com/research/modelling/surface/applications/pointcloud.html`; accessed 13-December-2007].

[BG97] Rick Beatson and Leslie Greengard. A short course on fast multipole methods. In *Wavelets, multilevel methods and elliptic PDEs (Leicester, 1996)*, Numer. Math. Sci. Comput., pages 1–37. Oxford Univ. Press, New York, 1997.

[Buh03] M. D. Buhmann. *Radial basis functions: theory and implementations*, volume 12 of *Cambridge Monographs on Applied and Computational Mathematics.* Cambridge University Press, Cambridge, 2003.

[CBC$^+$01] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76, New York, NY, USA, 2001. ACM.

[FGP05] A. C. Faul, G. Goodsell, and M. J. D. Powell. A Krylov subspace algorithm for multiquadric interpolation in many dimensions. *IMA J. Numer. Anal.*, 25(1):1–24, 2005.

[FI96] Michael S. Floater and Armin Iske. Multistep scattered data interpolation using compactly supported radial basis functions. *J. Comput. Appl. Math.*, 73(1-2):65–78, 1996.

[FI98] Michael S. Floater and Armin Iske. Thinning algorithms for scattered data interpolation. *BIT*, 38(4):705–720, 1998.

[FP00] A. C. Faul and M. J. D. Powell. Krylov subspace methods for radial basis function interpolation. In *Numerical analysis 1999 (Dundee)*, volume 420 of *Chapman & Hall/CRC Res. Notes Math.*, pages 115–141. Chapman & Hall/CRC, Boca Raton, FL, 2000.

[FP07] Bengt Fornberg and Cecile Piret. A stable algorithm for flat radial basis functions on a sphere. *SIAM Journal on Scientific Computing*, 30(1):60–80, 2007.

[Fra82] Richard Franke. Scattered data interpolation: tests of some methods. *Math. Comp.*, 38(157):181–200, 1982.

[FW04] B. Fornberg and G. Wright. Stable computation of multiquadric interpolants for all values of the shape parameter. *Comput. Math. Appl.*, 48(5-6):853–867, 2004.

[FZ07] Bengt Fornberg and Julia Zuev. The Runge phenomenon and spatially variable shape parameters in RBF interpolation. *Comput. Math. Appl.*, 54(3):379–398, 2007.

[GHS93] K. Guo, S. Hu, and X. Sun. Conditionally positive definite functions and Laplace-Stieltjes integrals. *J. Approx. Theory*, 74(3):249–265, 1993.

[GR87] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73(2):325–348, 1987.

[GVL96] Gene H. Golub and Charles F. Van Loan. *Matrix computations.* Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[Har71] R. L. Hardy. Multiquadric equations of topography and other irregular surfaces. *J. Geophys. Res.*, (76):1905–1915, 1971.

[IA04] Armin Iske and V. I. Arnold. *Multiresolution Methods in Scattered Data Modelling.* SpringerVerlag, 2004.

[IL05] Armin Iske and Jeremy Levesley. Multilevel scattered data approximation by adaptive domain decomposition. *Numer. Algorithms*, 39(1-3):187–198, 2005.

[Isk01] Armin Iske. Hierarchical scattered data filtering for multilevel interpolation schemes. In *Mathematical methods for curves and surfaces (Oslo, 2000)*, Innov. Appl. Math., pages 211–221. Vanderbilt Univ. Press, Nashville, TN, 2001.

[Lab07] Stanford Computer Graphics Laboratory. The stanford 3d scanning repository, 2007. [`http://www-graphics.stanford.edu/data/3Dscanrep/`; accessed 13-December-2007].

[LW06] Oren E. Livne and Grady B. Wright. Fast multilevel evaluation of smooth radial basis function expansions. *Electron. Trans. Numer. Anal.*, 23:263–287 (electronic), 2006.

[Mic86] Charles A. Micchelli. Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constr. Approx.*, 2(1):11–22, 1986.

[Pir07] Cecile Piret. *Analytical and Numerical Advances in Radial Basis Functions*. PhD thesis, University of Colorado, 2007.

[Pow05] M.J. Powell. Five lectures on radial basis functions. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2005.

[Sch95] Robert Schaback. Error estimates and condition numbers for radial basis function interpolation. *Adv. Comput. Math.*, 3(3):251–264, 1995.

[Wen05] Holger Wendland. *Scattered data approximation*, volume 17 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2005.