An Application of the Randomized Singular Value Decomposition (SVD) to Polynomial and Radial Basis Function (RBF) Approximations

Hannan Bruns
University of Utah

UUCS-20-015

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

4 December 2020

Abstract

Matrix decompositions have widespread application in scientific computing and numerical analysis. The Singular Value Decomposition (SVD) is of particular interest, due to its applicability in solving rank-deficient and ill-conditioned linear systems, and in solving linear least squares problems. Traditional algorithms for computing the SVD are computationally expensive, sometimes prohibiting their use in numerical approximation. We propose to apply a modern randomized technique for computing the SVD to deterministic approximation problems. Our target approximation problems involve linear approximation with approximants comprised of radial basis functions (RBFs) and polynomials; the matrices arising from such approximants can be rank-deficient or ill-conditioned. Our goal is to explore the interplay of parameters from both the approximants and the randomized SVD algorithm with an eye toward improving cost-accuracy profiles.

AN APPLICATION OF THE RANDOMIZED SINGULAR VALUE DECOMPOSITION (SVD) TO POLYNOMIAL AND RADIAL BASIS FUNCTION (RBF) APPROXIMATIONS

by

Hannah Bruns

A Senior Thesis Proposal Submitted to the Faculty of The University of Utah
In Partial Fulfillment of the Requirements for the Degree

Bachelor of Computer Science
School of Computing
The University of Utah
November 2020

Approved by:

Varun Shankar	H. James de St. Germain	Mary Hall
Thesis Faculty Advisor	Director of Undergraduate Studies	Director of School of Computing

Abstract

Matrix decompositions have widespread application in scientific computing and numerical analysis. The Singular Value Decomposition (SVD) is of particular interest, due to its applicability in solving rank-deficient and ill-conditioned linear systems, and in solving linear least squares problems. Traditional algorithms for computing the SVD are computationally expensive, sometimes prohibiting their use in numerical approximation. We propose to apply a modern randomized technique for computing the SVD to deterministic approximation problems. Our target approximation problems involve linear approximation with approximants comprising of radial basis functions (RBFs) and polynomials; the matrices arising from such approximants can be rank-deficient or ill-conditioned. Our goal is to explore the interplay of parameters from both the approximants and the randomized SVD algorithm with an eye toward improving cost-accuracy profiles.

Contents

1 Introduction **6**

1.1 Polynomial Approximation 6

1.2 Radial Basis Functions 6

1.2.1 Traditional algorithms for the SVD 7

1.3 Randomized SVD 7

2 Polynomial Approximations **9**

2.1 Polynomial Least Squares 9

2.2 Implementation of the Martinsson Algorithm 10

2.3 Results 11

2.3.1 Timings 11

2.3.2 2D Polynomial Approximations 12

2.3.3 3D Polynomial Approximations 14

3 Polynomial and RBF Approximations **17**

3.1 RBF Approximations 17

3.2 RBFs and Polynomial Approximations 18

3.3 Results 19

3.3.1 2D RBFs with Augmented Polynomials 19

3.3.2 3D RBFs Augmented with Polynomials 22

4 Conclusion **26**

List of Figures

2.1 Timings of SVD Computation of Square and Rectangular Vandermonde Matrices 12

2.2 Relative error as a function of number of nodes N and tolerance ϵ for polynomial least squares approximation of the Runge function. 13

2.3 Relative error as a function of number of nodes N and r values for polynomial least squares approximation of the Runge function. 13

2.4 Relative error as a function of number of nodes N and Vandermonde matrix composition for polynomial least squares approximation of the Runge function. 14

2.5 Relative error as a function of number of nodes N and tolerance ϵ for polynomial least squares approximation of the Runge function. 15

2.6 Relative error as a function of number of nodes N and r values for polynomial least squares approximation of the Runge function. 15

2.7 Relative error as a function of number of nodes N and Vandermonde matrix composition for polynomial least squares approximation of the Runge function. 16

2.8 Relative error as a function of number of nodes N and Vandermonde matrix composition for polynomial least squares approximation of the Runge function, $r = 9$ 16

3.1 Relative error as a function of number of nodes N and tolerance ϵ for RBF and polynomial approximation of the Runge function. 20

3.2 Relative error as a function of number of nodes N and r values for RBF and polynomial approximation of the Runge function. 21

3.3 Relative error as a function of number of nodes N and tolerance ϵ for RBF and polynomial approximation of the Runge function, $\phi(s) = s^m$ with $m = 2\ell + 1$. 21

3.4 Relative error as a function of number of nodes N and r values for RBF and polynomial approximation of the Runge function, $\phi(s) = s^m$ with $m = 2\ell + 1$. 22

3.5 Relative error as a function of number of nodes N and tolerance ϵ for RBF and polynomial approximation of the Runge function. 23

3.6 Relative error as a function of number of nodes N and r values for RBF and polynomial approximation of the Runge function 23
3.7 Relative error as a function of number of nodes N and tolerance ϵ for RBF and polynomial approximation of the Runge function, $\phi(s) = s^m$ with $m = 2\ell + 1$. 24
3.8 Relative error as a function of number of nodes N and tolerance ϵ for RBF and polynomial approximation of the Runge function, $\phi(s) = s^m$ with $m = 2\ell + 1$. 24

Chapter 1

Introduction

In this chapter we will describe polynomial and RBF approximation and introduce some of the pitfalls of solving these approximations systems. While there are several methods to solve these systems, most involve expensive dense linear algebra. For example, computing the SVD of a $n \times n$ matrix has a complexity of $O(n^3)$. However, it may not always be necessary to solve such systems exactly, deterministically, or to machine precision. The goal of this thesis is to apply an existing randomized SVD algorithm to the solution of ill-condition linear systems arising from interpolation and least squares problems. To that end, we focus on two popular approximation schemes: polynomial least squares, and radial basis functions (RBFs).

1.1 Polynomial Approximation

Fitting data to a model is a fundamental practice in data science and numerical analysis. In 1D, given some data, one can use polynomial approximation to find a function that is an exact fit for the data [5]. This technique is called interpolation. However, the matrices arising from polynomial interpolation can be rank-deficient in anything higher than 1D for scattered nodes (Mairhuber-Curtis theorem, see [5]). Therefore, in order to perform approximation with polynomials, one option is to deduce the matrix rank using a rank-revealing decomposition such as the singular value decomposition (SVD). An alternate approach is to use Radial Basis Functions (RBFs) instead.

1.2 Radial Basis Functions

Radial Basis Functions (RBFs) are used for interpolation of multidimensional scattered data [1, 12]. The ability to construct an unknown function from scattered data has many applications in data science and neural networks [14]. The technique of multidimensional scattered interpolation has progressed to a mesh-free method that can numerically solve partial differential equations on irregular domains [1, 15]. However, these methods result in a large, ill-conditioned linear system [15]. While the SVD can be used to handle ill-conditioning, the modern approach to combat ill-conditioning in RBF interpolation matrices is to augment the RBF interpolants with moderate-degree polynomials [6]. These methods use parameter-free RBFs augmented with polynomials. Unfortunately, while blocks arising from RBFs are full rank on scattered nodes, the polynomial blocks may not be, depending on the degree of the polynomial and the number of points. Thus, while this technique overcomes ill-conditioning, it trades this off with rank-deficiency, especially if the points lie on a surface [6, 11, 12]. Again, solving the resulting linear systems necessitates a rank-revealing decomposition like the SVD. Even if the matrices involved are full rank, it may be possible to enforce polynomial reproduction on the RBF interpolants in a least squares sense. To that end, a non-deterministic (but fast) algorithm for approximating the SVD up to a user-specified tolerance is desirable.

1.2.1 Traditional algorithms for the SVD

SVDs are a powerful tool for solving linear systems involving ill-conditioned or rank-deficient matrices. Consequently, they can be useful in both polynomial and RBF approximation. However, classical (deterministic) methods of constructing SVDs, and many other matrix decompositions, have proven to be inadequate in dealing with problems involving large matrices. There are two reasons for this: first, these algorithms have high (and fixed) computational complexity, thus resulting in poor scaling to large problems [3, 4]; second, these algorithms attempt to create decompositions that when recombined match the original matrix perfectly (within machine epsilon) [9].

1.3 Randomized SVD

Large matrices commonly arise in modern data science and scientific computing applications [9]. However, in many of these contexts, it is often sufficient to create a matrix decomposition or factorization that is accurate to some user-specific (or application-specific tolerance). This has led to the rise of randomized algorithms for matrix decompositions [7]. In this work, we focus specifically on the algorithms described in [7], which are explicitly designed to produce such approximate factorizations using projections onto random vectors. In addition, these algorithms are rank-revealing in two ways: first, they are capable of computing a

close-to-optimal low rank approximation in the case of rank-deficient matrices; and second, they can reveal effective numerical rank in highly ill-conditioned matrices.

In the case of a rank-deficient linear system where the rank is not known in advance, the randomized algorithms from [7] enable the solution of the linear systems in a least-squares sense. On the other hand, when used on a full-rank but ill-conditioned matrix, these algorithms can enable efficient storage and meaningful solution of the linear systems while ameliorating the effects of ill-conditioning. Further, these algorithms explicitly take as input a user-supplied tolerance parameter that allow the algorithms to trade speed for precision. However, to the best of our knowledge, such algorithms have not found widespread use in scientific computing or approximation. Thus, very little is known about how the algorithm parameters interact with approximation parameters such as the number of data points, the number of RBFs, the polynomial degree, and the desired accuracy/tolerance of the approximation.

Chapter 2

Polynomial Approximations

2.1 Polynomial Least Squares

As previously stated, given some data, one can use polynomial approximation to find a function that is an exact fit for the data. This is called interpolation. In cases where data is noisy, a common method used to fit polynomials to data is polynomial least squares. For the 1D case, let $x = \{x_k\}_{k=0}^{N}$ be some set of points at which we are provided some samples of some function $f(x)$. Thus, we are given $y_k = f(x_k)$, $k = 0, \ldots, N$ and seek a polynomial p of the form:

$$p(x) = \sum_{k=0}^{M} a_k x^k, \tag{2.1}$$

$$p(x_j) \approx y_j, j = 0, \ldots, N \tag{2.2}$$

where a_k are the unknown approximation coefficients. To find these coefficients, we need to enforce the conditions $p(x_j) \approx y_j, j = 0, \ldots,$ N. This creates a linear system of the form

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & x_0^3 & \ldots & x_0^M \\ 1 & x_1 & x_1^2 & x_1^3 & \ldots & x_1^M \\ 1 & x_2 & x_2^2 & x_2^3 & \ldots & x_2^M \\ 1 & x_3 & x_3^2 & x_3^3 & \ldots & x_3^M \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & x_N^3 & \ldots & x_N^M \end{bmatrix}}_{A} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{2.3}$$

The Vandermonde matrix, A, varies in size depending on the polynomial degree. Let ℓ be the polynomial degree and d the dimension, then $M = \binom{l+d}{d}$. Higher dimension analogues are obtained by using monomials

corresponding to the total degree polynomial of degree ℓ in d dimensions.

To solve for the unknown a_k values, we must solve the linear system 2.3. If $M > N$, this is an overdetermined linear system that has a unique solution if the matrix A has full column-rank. Consequently, one can use the SVD to find the least squares solution as:

$$\begin{aligned} \mathrm{U}\Sigma V^T \mathbf{a} &= \mathbf{y} \\ \implies \mathbf{a} &= V\Sigma^{\dagger} U^T \mathbf{y} \end{aligned} \tag{2.4}$$

We will approximate the SVD using the Algorithm 4.2 from [7], which is a fast randomized algorithm. We now describe this algorithm below.

2.2 Implementation of the Martinsson Algorithm

Our first task was to implement Algorithm 4.2 (the adaptive randomized range finder) from [7] (henceforth called the Martinsson Algorithm). In our implementation of the algorithm, the parameters are: an $m \times n$ matrix A, a tolerance ϵ, and an integer r. The algorithm computes an orthonormal matrix Q that we use to calculate the SVD. The tolerance ϵ acts as a threshold of the accuracy of this algorithm. The algorithm guarantees to maintain $\|(\mathbf{I} - QQ^*)\,\boldsymbol{A}\| \leq \varepsilon$ while building the orthonormal matrix Q. The integer r is necessary to balance the computational cost as well as the reliability of the algorithm. This algorithm draws a random set of r Gaussian vectors to interact with the original matrix A. As this happens, we examine the subspace formed from these interactions with random vectors. We do this for each iteration of a loop until it reaches the threshold for accuracy. The output is an orthonormal Q matrix, which can be used to compute the SVD in a deterministic fashion. Since Q is much smaller than the original matrix A, computing the SVD will be much more efficient. That being said, one goal of this paper is to examine the relationships between r and ϵ [7].

Due to the nature of this algorithm, the orthonormal Q matrix is much smaller than A. This is because we are not trying to replicate A, but rather form an approximate basis for the column space of A. As with most matrices, not all columns are pertinent in forming this basis, which is how this algorithm can be substantially faster than traditional methods.

Algorithm 1: Martinsson's Algorithm

Result: SVD(A)

input : $M \times N$ matrix A, an integer r, and a tolerance ϵ

output: SVD of A

Draw standard Guassian vectors $\boldsymbol{w}^{(1)}, ..., \boldsymbol{w}^{(r)}$;

For i = 1, 2,...,r, compute $\boldsymbol{y}^{(i)} = \boldsymbol{A}\boldsymbol{w}^{(i)}$);

$j = 0$;

$\boldsymbol{Q}^{(0)} = []$, $M \times 0$ empty matrix;

while $\max\left\{\left\|\boldsymbol{y}^{(j+1)}\right\|, \left\|\boldsymbol{y}^{(j+2)}\right\|, \ldots, \left\|\boldsymbol{y}^{(j+r)}\right\|\right\} > \varepsilon/(10\sqrt{2/\pi})$ **do**

  $j = j + 1$;

  Overwrite $\boldsymbol{y}^{(j)}$ by $\left(\mathbf{I} - \boldsymbol{Q}^{(j-1)}\left(\boldsymbol{Q}^{(j-1)}\right)^*\right)\boldsymbol{y}^{(j}$;

  $\boldsymbol{q}^{(j)} = \boldsymbol{y}^{(j)} / \left\|\boldsymbol{y}^{(j)}\right\|$;

  $\boldsymbol{Q}^{(j)} = \left[\boldsymbol{Q}^{(j-1)}\boldsymbol{q}^{(j)}\right]$;

  Draw a standard Gaussian vector $\boldsymbol{w}^{(j+r)}$ of length N;

  $\boldsymbol{y}^{(j+r)} = \left(\mathbf{I} - \boldsymbol{Q}^{(j)}\left(\boldsymbol{Q}^{(j)}\right)^*\right)\boldsymbol{A}\boldsymbol{\omega}^{(j+r)}$;

  for $i = (j+1)$, $(j+2)$,...,$(j+r-1)$ **do**

    Overwrite $\boldsymbol{y}^{(i)}$ by $\boldsymbol{y}^{(i)} - \boldsymbol{q}^{(j)}\left\langle \boldsymbol{q}^{(j)}, \boldsymbol{y}^{(i)} \right\rangle$;

  end

end

$B = \boldsymbol{Q'}$;

$[U_b, S, V] = svd(B, 0)$;

$U = Q * U_b$;

return U, S, V;

2.3 Results

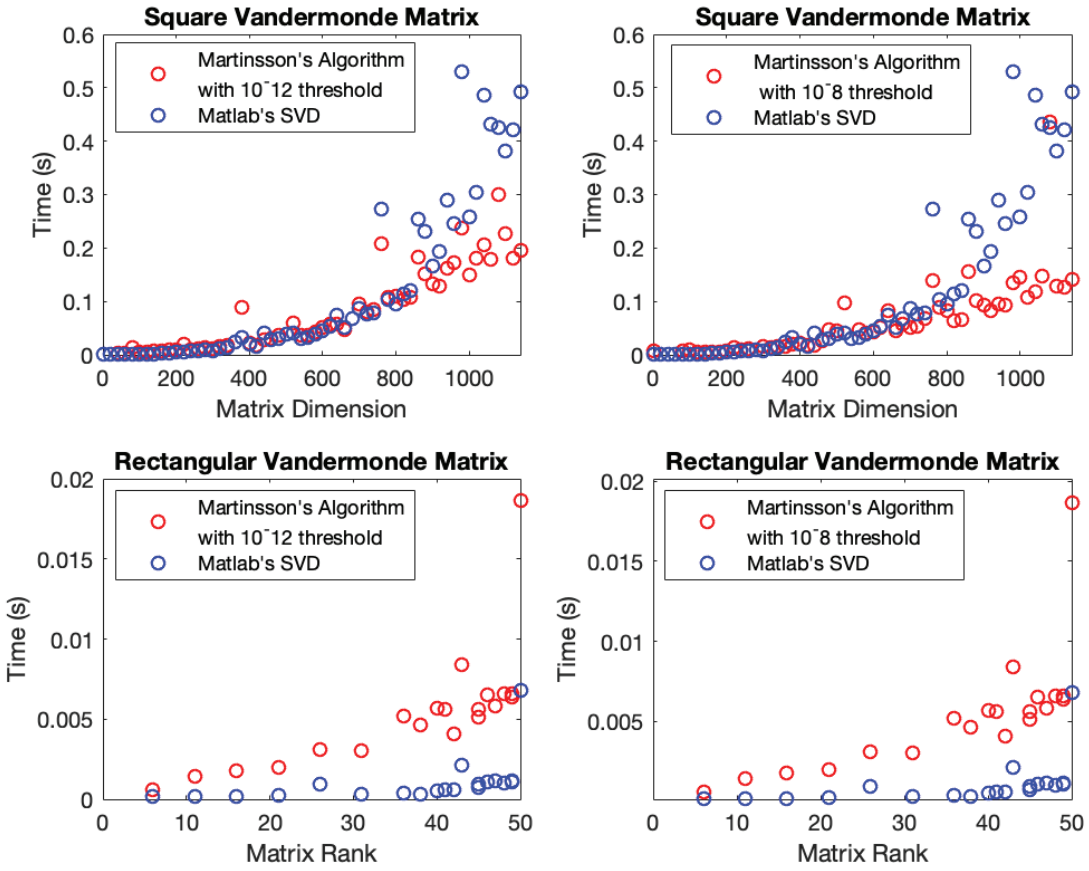
In this section, we will present numerical results for the approximate solution of the polynomial least squares problem using the Martinsson algorithm. First, in section 2.3.1, we present timings.

2.3.1 Timings

Our preliminary study showed potential for speedup over Matlab's built-in SVD on polynomial approximation problems. Below we can see that having a lower threshold for accuracy gives us substantial speedup

with large square polynomial matrices.

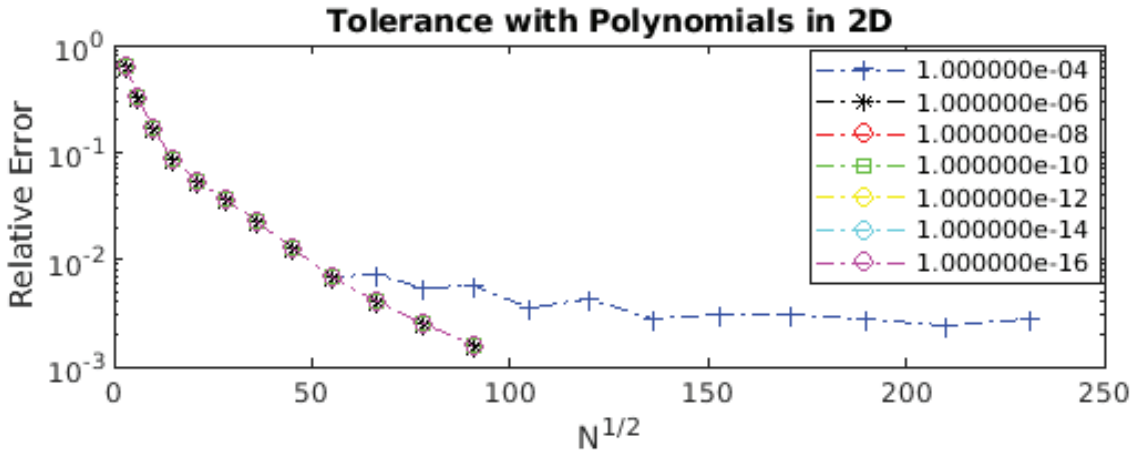
Figure 2.1: Timings of SVD Computation of Square and Rectangular Vandermonde Matrices



2.3.2 2D Polynomial Approximations

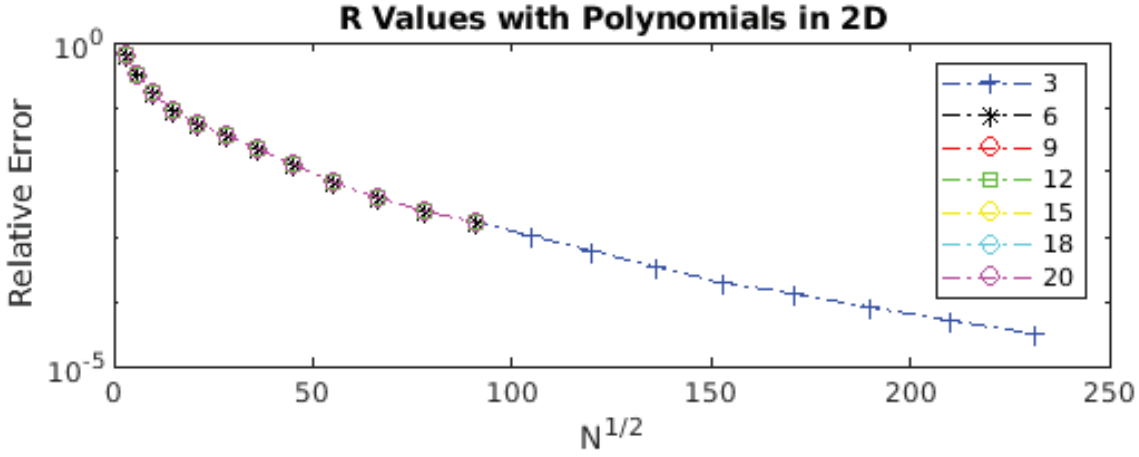
Our goal was to examine how the tolerance, ϵ, and the r value parameters affect the accuracy and performance of polynomial interpolation. We first tested this with polynomial least squares approximation of the Runge function, $f(x,y) = \frac{1}{1+25(x^2+y^2)}$ where x and y are the positions of which data is supplied. We created the point sets with Halton sequences in the domain of $[0,1]^2$, which are quasi-random nodes [8]. Testing the tolerance parameter is especially interesting because in Figure 2.1 we can see that the lower the tolerance gets, the more efficient the algorithm completes. Below we tested a range of tolerances to be used as a parameter in the Martinsson Algorithm. We then used the SVD to solve for the rectangular Polynomial Least Squares system. We compared those results against the actual data and graphed the relative error.

Figure 2.2: Relative error as a function of number of nodes N and tolerance ϵ for polynomial least squares approximation of the Runge function.



As Figure 2.2 shows, it looks like the tolerance has little influence on the accuracy of the computation of the SVD for Polynomial Approximations in 2D. This is an unexpected outcome, but one that shows the Martinsson Algorithm can make very fast and efficient computations of the SVD without compromising the accuracy. Next we examined the r parameter with the same tests as the tolerances.

Figure 2.3: Relative error as a function of number of nodes N and r values for polynomial least squares approximation of the Runge function.

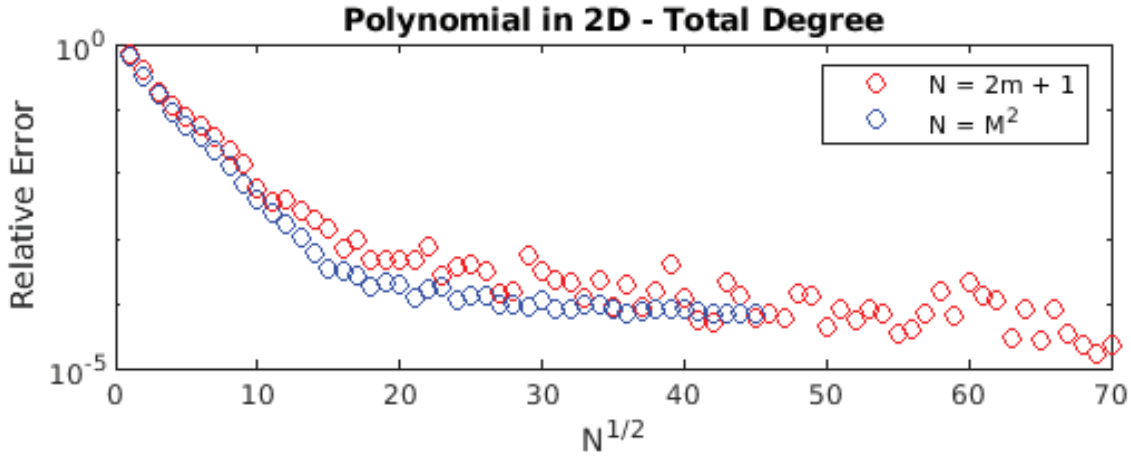


Similar to the tolerance tests, r values do not appear to influence the results significantly in 2D.

In addition to studying the parameters of the Martinsson algorithm, we were interested in investigating how the Martinsson algorithm's effectiveness changes with the composition of the the Vandermonde matrix in polynomial approximation.

We tested rectangular systems where the makeup of the Vandermonde matrix is either $N = 2M + 1$ or $N = M^2$ on the Martinsson algorithm. Specifically, we wanted to see if one or the other was better suited for the Martinsson algorithm and this created more accurate approximations.

Figure 2.4: Relative error as a function of number of nodes N and Vandermonde matrix composition for polynomial least squares approximation of the Runge function.

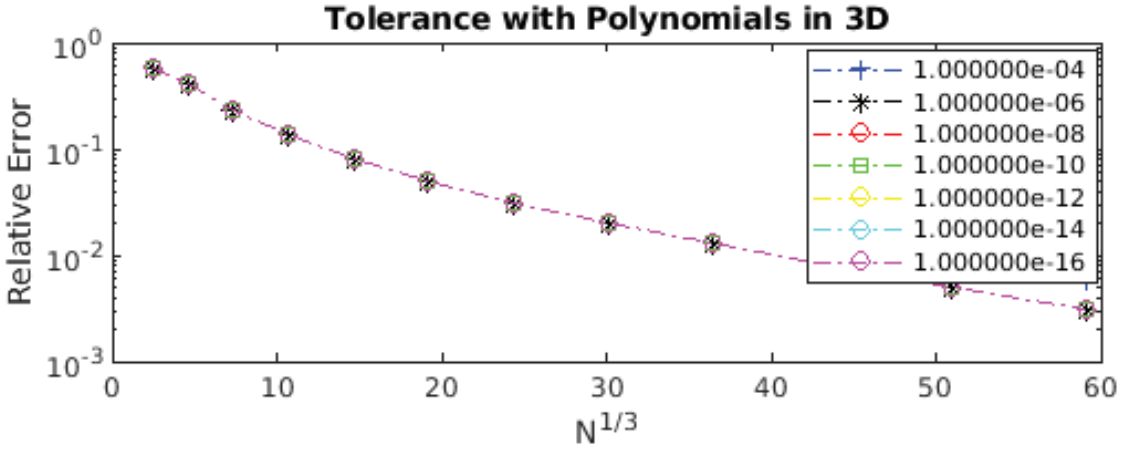


It is clear that polynomial approximations with N= 2M+1 converge more smoothly as the number of points is increased. First, the Martinsson algorithm is better equipped for matrices of this composition. Second, the tolerance and r value parameters have different 'best' values for $N = 2M + 1$ and $N = M^2$.

2.3.3 3D Polynomial Approximations

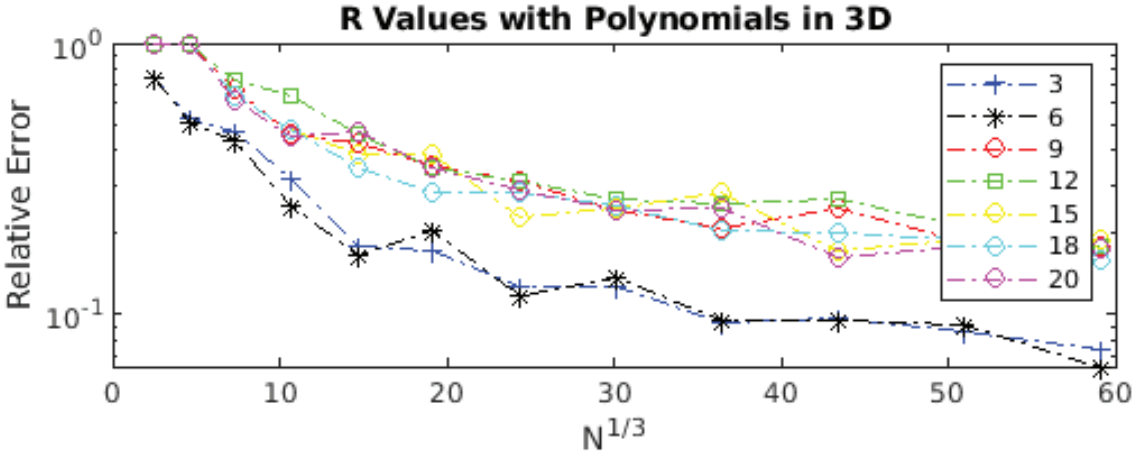
Next we ran the same tests of testing the tolerance ϵ and r value parameters, but with polynomial least squares approximation of the Runge function, $f(x, y, z) = \frac{1}{1+25(x^2+y^2+z^2)}$ where x, y, and z are the positions of which data is supplied. As we did in polynomial approximations in 2D, we created the point sets with Halton sequences, which are sequences that appear to be random. We ran the first test on the tolerance ϵ. Again, we tested a range of ϵ to see if they impacted the accuracy of the resulting polynomial interpolation. We do this because the SVD is used in the solution of the linear system created as a result of trying to find the best fit function to the given data.

Figure 2.5: Relative error as a function of number of nodes N and tolerance ϵ for polynomial least squares approximation of the Runge function.



Similar to the case of 2D, it doesn't appear that the tolerances have an affect on the accuracy of the system. This is meaningful because again, the Martinsson algorithm computes much faster with a lower tolerance. Next we ran the same test, but with a range of r values.

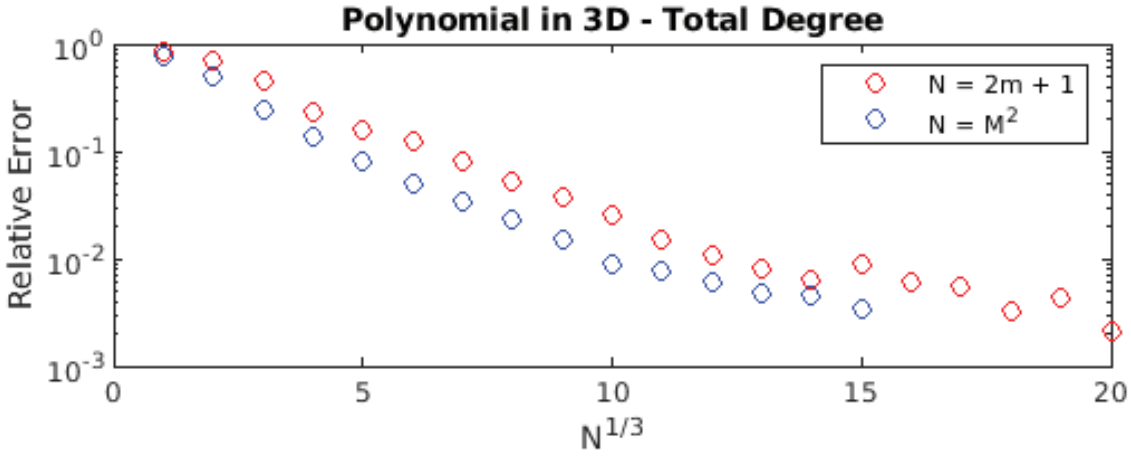
Figure 2.6: Relative error as a function of number of nodes N and r values for polynomial least squares approximation of the Runge function.



Unlike the r values in 2D, our tests show that r values in 3D are much more influential on the accuracy of the Martinsson algorithm. We can see from the graph that an r value less than 9 is ideal for 3D polynomial approximation using the Martinsson Algorithm.

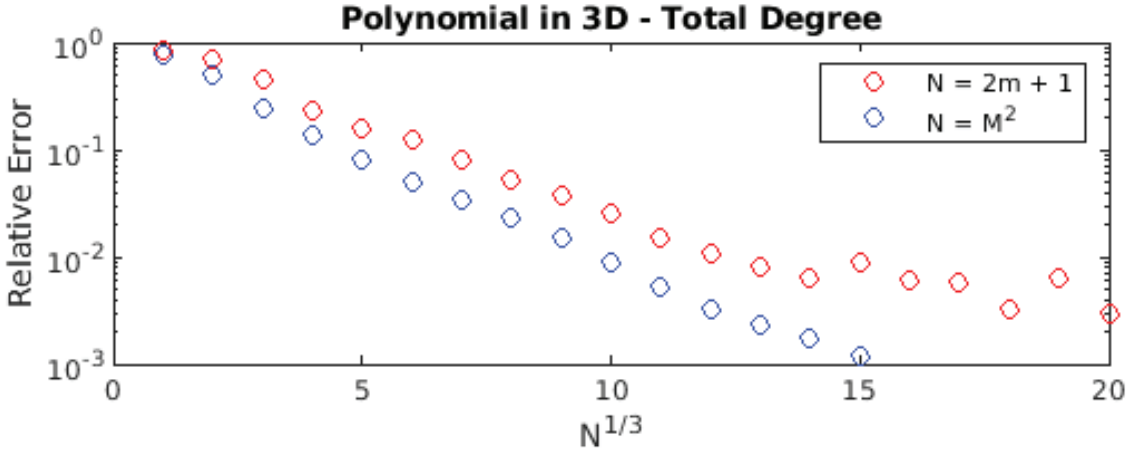
The last test we ran in 3D for polynomial approximation was on the composition of the Vandermonde matrix. Specifically, when $N = 2M + 1$ or $N = M^2$.

Figure 2.7: Relative error as a function of number of nodes N and Vandermonde matrix composition for polynomial least squares approximation of the Runge function.



In this case it appears that $N = M^2$ causes the relative error to decay faster. However, our tests above demonstrated that in the case of 3D, the r value should remain below 9. So with that in mind, we ran the same test again comparing $N = 2M + 1$ and $N = M^2$. As Figure 2.8 shows, there is some improvement with $N = M^2$.

Figure 2.8: Relative error as a function of number of nodes N and Vandermonde matrix composition for polynomial least squares approximation of the Runge function, $r = 9$



Chapter 3

Polynomial and RBF Approximations

3.1 RBF Approximations

Unlike polynomial interpolation, RBF interpolation is better set up for all dimensions [15]. Instead of using a Vandermonde matrix as in the polynomial case, we use a distance matrix with a radial basis function expansion when solving scattered data interpolation problems with RBFs [5, 13]. Given a set of points $X = \{\mathbf{x}_k\}_{k=1}^{N}$, we want to find some f that serves as a proxy function for the given data, y_k. So, we are constructing an RBF interpolant to the data of the following form where the interpolation coefficients are c_k [13]

$$p(\mathbf{x}) = \sum_{k=1}^{N} c_k \phi\left(\|\mathbf{x} - \mathbf{x}_k\|\right) \tag{3.1}$$

We find the interpolation coefficients by enforcing the conditions $p(x_k) = y_k, k = 1, \ldots, N$. This creates the linear system below

$$\underbrace{\begin{bmatrix} \phi\left(r_{1,1}\right) & \phi\left(r_{1,2}\right) & \ldots & \phi\left(r_{1,N}\right) \\ \phi\left(r_{2,1}\right) & \phi\left(r_{2,2}\right) & \ldots & \phi\left(r_{2,N}\right) \\ \vdots & \vdots & \ddots & \vdots \\ \phi\left(r_{N,1}\right) & \phi\left(r_{N,2}\right) & \ldots & \phi\left(r_{N,N}\right) \end{bmatrix}}_{A_X} \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix}}_{c_f} = \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix}}_{f_X} \tag{3.2}$$

where $r_{i,j} = \|\mathbf{x}_i - \mathbf{x}_j\|$.

Above, ϕ is the RBF and typically takes on the form of $\phi(s) = s^m$. RBF interpolation systems like this one are notorious for being ill-conditioned [5, 14] due to the minimum distance between points becoming smaller. Although there are a number of ways to help make this system more stable, like changing bases, one way is to augment a polynomial to this system [5, 12].

3.2 RBFs and Polynomial Approximations

RBF approximation implemented with augmented polynomials can restore approximation quality, but can also introduce rank-deficiency [6, 11, 12]. We modify the RBF system to include polynomial terms to create an equation of the form [2]

$$p(\mathbf{x}) = \sum_{k=1}^{N} c_k \phi\left(\|\mathbf{x} - \mathbf{x}_k\|\right) + \beta_1 + (c_2 x + \beta_3 y) \tag{3.3}$$

$$p(\mathbf{x}_k) = f(\mathbf{x}_k), k = 1 \dots N \tag{3.4}$$

where $\mathbf{x} = (x, y)$ and $\mathbf{x}_k = (x_k, y_k)$. To uniquely determine the interpolation coefficients, we also impose the constraints $\sum_{k=1}^{N} c_k = \sum_{k=1}^{N} c_k x_k = \sum_{k=1}^{N} c_k y_k = 0$. This creates the system below [2]

$$\begin{bmatrix} & & & | & 1 & x_1 & y_1 \\ & A & & | & \vdots & \vdots & \vdots \\ & & & | & 1 & x_N & y_N \\ - & - & - & + & - & - & - \\ 1 & \cdots & 1 & | & & & \\ x_1 & \cdots & x_N & | & & 0 & \\ y_1 & \cdots & y_N & | & & & \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_N \\ - \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_N \\ - \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{3.5}$$

The extension involves adding monomials up to degree ℓ in d dimensions. The system can be written as the following block system from Shankar [11]:

$$\underbrace{\begin{bmatrix} A & \Psi \\ \Psi^T & 0 \end{bmatrix}}_{\hat{A}} \underbrace{\begin{bmatrix} \mathbf{c} \\ \boldsymbol{\beta} \end{bmatrix}}_{\hat{c}} = \begin{bmatrix} \mathbf{f} \\ \mathbf{0} \end{bmatrix}, \tag{3.6}$$

In the above block linear system, $\mathbf{f}$ is the set of function values, Ψ is the A from Chapter 2, and A is the RBF interpolation matrix. As mentioned previously, A is typically ill-conditioned. While the polynomial block can help offset this ill-conditioning, this block itself could be rank-deficient on scattered nodes in high dimensions [1, 15]. The Martinsson algorithm is thus a good candidate to solving such linear systems.

When solving the above block linear system, we use the Schur complement [10]. This is defined as

$$S = 0 - \Psi^T A^{-1} \Psi \tag{3.7}$$

where S is the Schur complement. We can use this to solve for the block system like below.

$$S\boldsymbol{\beta} = \Psi^T A^{-1} \mathbf{f} \tag{3.8}$$

To be more efficient when finding A^{-1}, we used the LU decomposition as $A^{-1} = R^{-1}L^{-1}P$. This is beneficial since we are computing A^{-1} just once. $\boldsymbol{\beta}$ can be computed as

$$\boldsymbol{\beta} = V\Sigma^{\dagger}U^{T}(\Psi^{T}A^{-1}\mathbf{f}) \tag{3.9}$$

The factors V, Σ and U can be approximated with the Martinsson algorithm. Finally, we can solve for c as

$$\mathbf{c} = R^{-1}L^{-1}P(\mathbf{f} - \Psi\boldsymbol{\beta}) \tag{3.10}$$

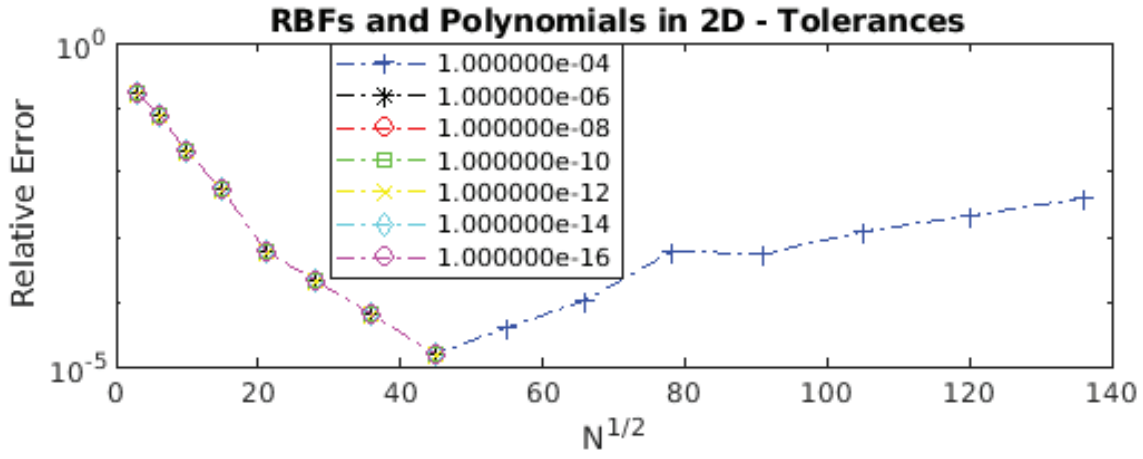
3.3 Results

In this section we will examine how the ϵ tolerance and r value parameters of the Martinsson algorithm affect RBFs with augmented polynomials. As with polynomial approximation, we used the Runge function, $f(x,y) = \frac{1}{1+25(x^2+y^2)}$ for 2D and $f(x,y,z) = \frac{1}{1+25(x^2+y^2+z^2)}$ for 3D. Again, we used the Halton sequence to generate pseudo-random point sets in the domain of $[0,1]^d$ where d is the dimension.

3.3.1 2D RBFs with Augmented Polynomials

We ran the same experiments as those for polynomial approximation. Our goal was to see how the tolerances and r value parameters influenced accuracy for RBF interpolation. We use the polyharmonic spline (PHS) radial basis function $\phi(s) = s^m$, where $m = \ell - 1$ if ℓ is even, and $m = \ell$ if ℓ is odd; here, ℓ is once again the polynomial degree.

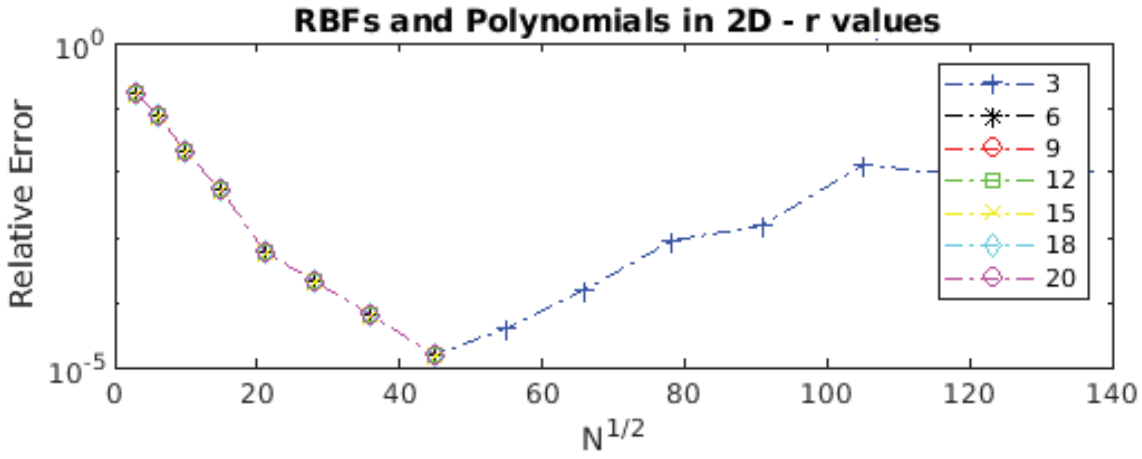
Figure 3.1: Relative error as a function of number of nodes N and tolerance ϵ for RBF and polynomial approximation of the Runge function.



Similar to the polynomial approximation tests in 2D shown in Figure 2.2, there appears to be no significant difference between the range of tolerances. This means that the Martinsson algorithm can approximate the SVD efficiently with a low tolerance threshold. However, one distinction between polynomial approximation and RBF interpolation in these tests is that the RBFs reach a lower error significantly faster and then spike back up after a number of N nodes. This is unlike the polynomial approximation (in Figure 2.2) which has a slower downward relative error trend throughout and does not reach a relative error as low.

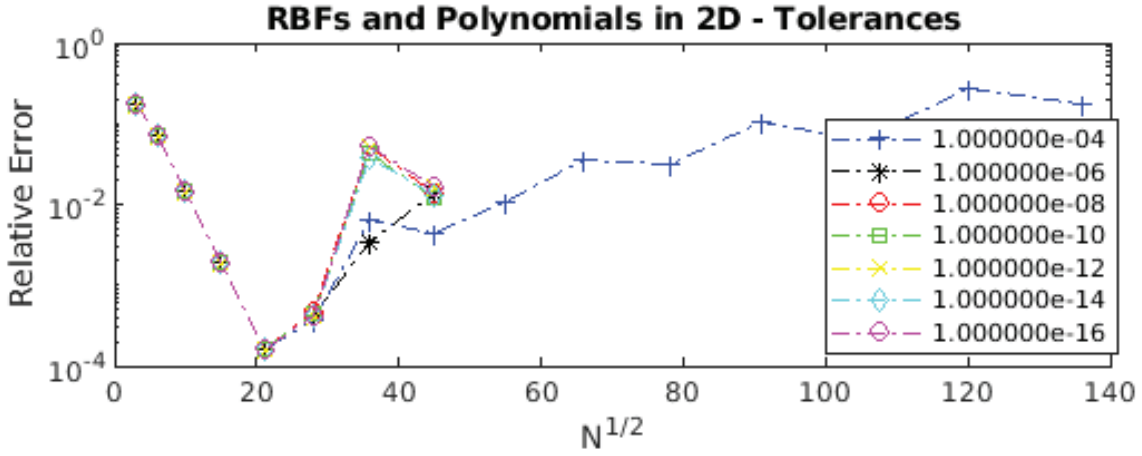
We also ran the same test as shown in Figure 3.1 with r values. Similar to the 2D polynomial approximation in Figure 2.3, it does not appear that the r value parameter influences accuracy. However, in both RBF with polynomials and polynomial least squares approximation a relative error of 10^{-5} is achieved although with polynomials the rate of relative error decay is significantly slower.

Figure 3.2: Relative error as a function of number of nodes N and r values for RBF and polynomial approximation of the Runge function.



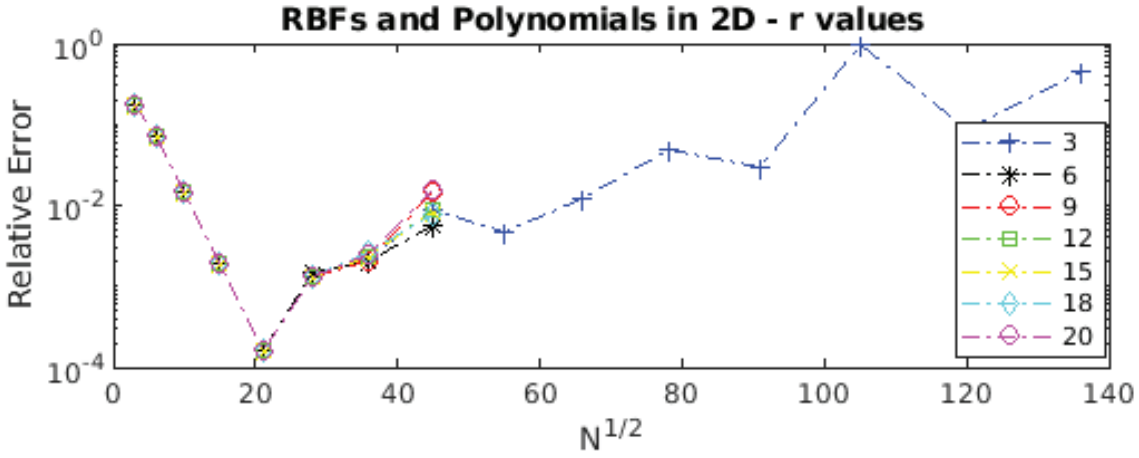
In addition to tests when $\phi(s) = s^m$ where $m = \ell - 1$ if ℓ is even and ℓ if ℓ is odd, we also ran the sames tests with $m = 2\ell + 1$ in 2D.

Figure 3.3: Relative error as a function of number of nodes N and tolerance ϵ for RBF and polynomial approximation of the Runge function, $\phi(s) = s^m$ with $m = 2\ell + 1$.



It appears that changing the polyharmonic spline, ϕ, has a significant influence on RBF interpolation with polynomials. With a new $m = 2\ell + 1$, the error spikes back up at a smaller rate and does not achieve the same relative error sa seen in Figure 3.1. The polynomial approximation tests shown in Figure 2.2, do not have a relative error upward spike after a certain N nodes unlike here in Figure 3.3. However, we are still reaching a lower relative error at a faster rate with RBFs than with polynomials. We also ran the same test, but with the r value parameter.

Figure 3.4: Relative error as a function of number of nodes N and r values for RBF and polynomial approximation of the Runge function, $\phi(s) = s^m$ with $m = 2\ell + 1$.

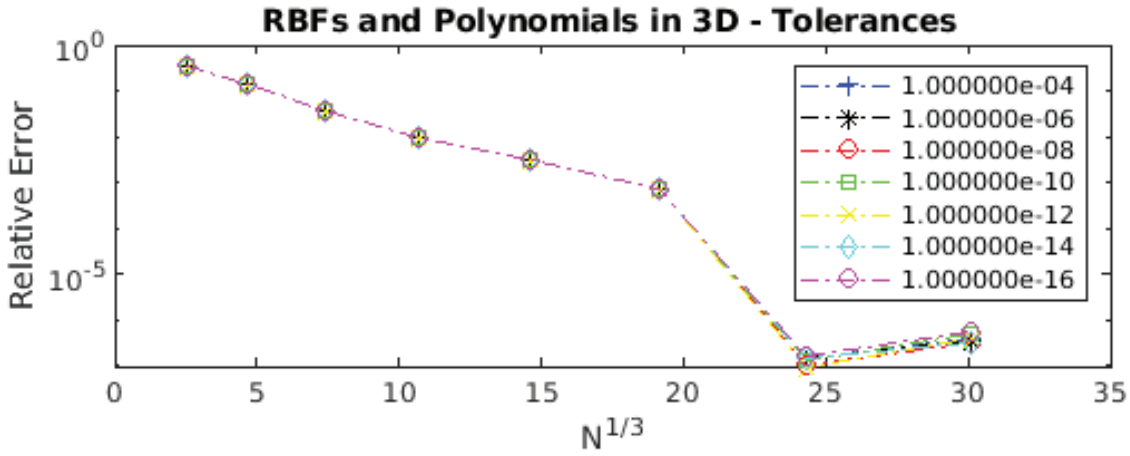


As with the tolerance parameter tests, the performance seems less desirable with a polyharmonic spline of $\phi(s) = s^m$ with $m = 2\ell + 1$ when N is large. The relative error spikes back after $N^{1/2} = 20$ whereas with $m = \ell$ or $m = \ell - 1$, the spike does not occur until $N^{1/2} = 40$. Additionally, above when $m = 2\ell + 1$, we do not reach a relative error of 10^{-5} like we do in Figure 3.1, but the rate of relative error decay is significantly faster. However, both values of ϕ achieve a lower relative error at a more rapid rate than with polynomials in Chapter 2.

3.3.2 3D RBFs Augmented with Polynomials

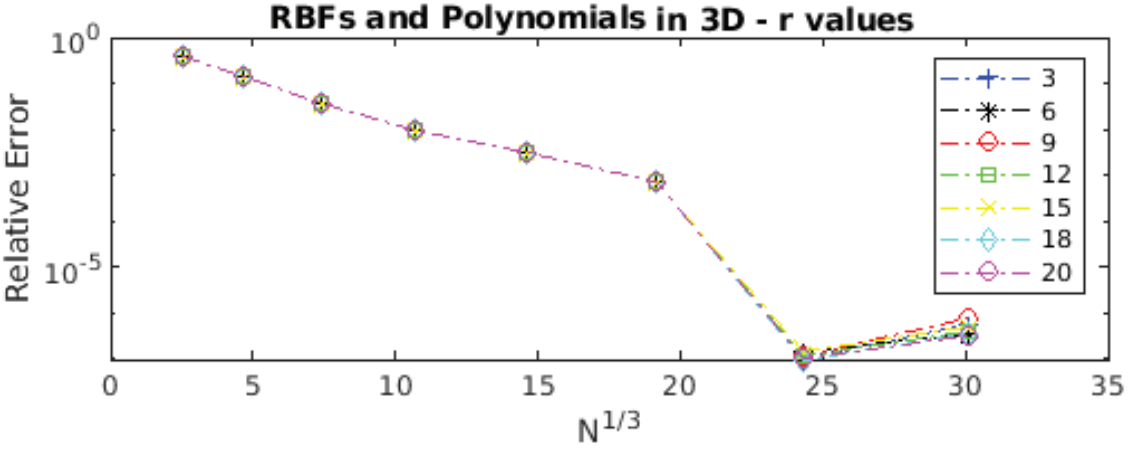
We ran the same experiments as above, approximating the Runge function but in 3D. Again, in our preliminary tests our radial basis function, $\phi(s) = s^m$, where $m = \ell$ or $m = \ell - 1$ and ℓ is the polynomial degree.

Figure 3.5: Relative error as a function of number of nodes N and tolerance ϵ for RBF and polynomial approximation of the Runge function.



Similar to the RBFs with augmented polynomials in 2D, there appears to be no significant difference between the range of tolerances. However, in 3D, we attain an even lower relative error at a more rapid rate of relative error decay than in 2D with RBFs shown in Figure 3.1 and both dimensions of polynomials in Figure 2.2 and 2.5. We also ran the same test with r values.

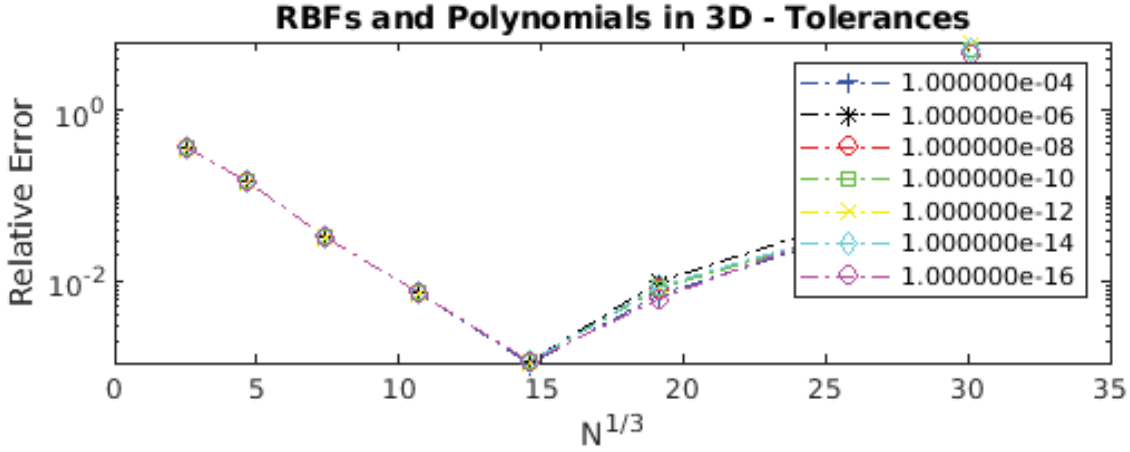
Figure 3.6: Relative error as a function of number of nodes N and r values for RBF and polynomial approximation of the Runge function



Similar to the 2D case seen in Figure 3.2, it doesn't appear that the r value parameter influences the accuracy. However, again, we are reaching a lower relativer error at a significantly quicker rate than with RBFs in 2D and both dimensions of polynomial least square approximations. Figure 3.5 and 3.6 are almost identical.

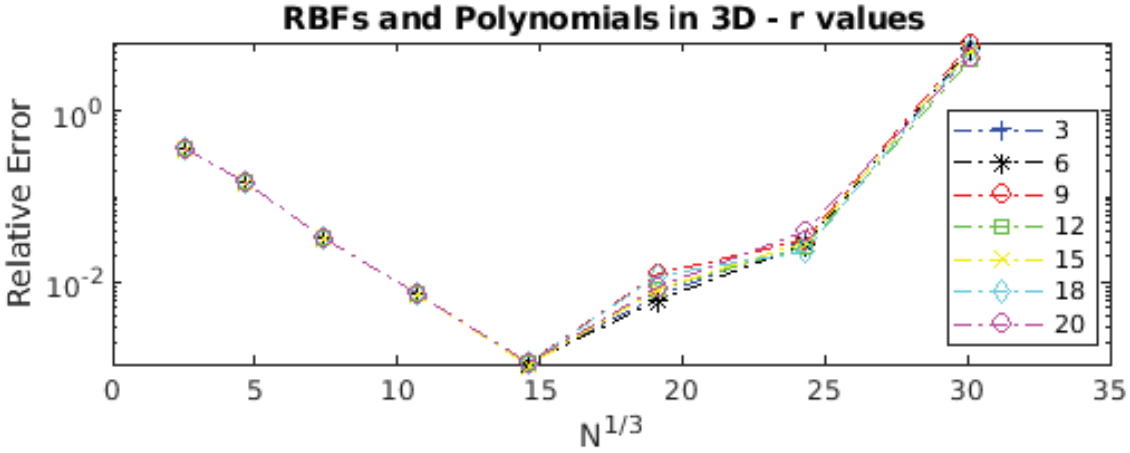
In addition to tests when $\phi(s) = s^m$ with $m = \ell$ or $m = \ell - 1$ in 2D, we also ran the sames tests with $m = 2\ell + 1$ in 3D.

Figure 3.7: Relative error as a function of number of nodes N and tolerance ϵ for RBF and polynomial approximation of the Runge function, $\phi(s) = s^m$ with $m = 2\ell + 1$.



It appears that changing polyharmonic spline, ϕ, does not influence the affect tolerances have on the Martinsson algorithm. However, the ϕ drastically affects the relative error and the rate of its decay. In Figure 3.7, the relative error is just past 10^{-2} whereas when $m = \ell$ or $m = \ell - 1$ in Figure 3.5, we achieved a relative error lower than 10^{-5} and at a more rapid pace. The relative error above is more similar to what we found with polynomial approximations in Figure 2.5. We also ran the same test, but with the r value parameter.

Figure 3.8: Relative error as a function of number of nodes N and tolerance ϵ for RBF and polynomial approximation of the Runge function, $\phi(s) = s^m$ with $m = 2\ell + 1$.



As with the tolerance parameter tests, the performance seems less desirable with a polyharmonic spline

with $m = 2\ell + 1$. The relative error spikes back after $N^{1/3} = 15$ whereas in the case of when $m = l - 1$, the spike does not occur until $N^{1/3} = 25$. Additionally, the relative error here is far from 10^{-5} that we see in Figure 3.6. However, it doesn't really seem to differ between the tolerances and r values, which again makes it seem that it is the polyharmonic spline causing the error spikes and higher relative errors. Again, it seems like changing the ϕ here makes the influence of the ϵ tolerance and the r values have a much different effect on RBF interpolation. Overall, RBF interpolation achieved lower relative errors at a more rapid rate than with polynomial approximations with a $\phi(s) = s^m$ where $m = \ell$ or $m = \ell - 1$.

Chapter 4

Conclusion

After in-depth testing of the Martinsson algorithm with polynomial approximation and RBF interpolation augmented with polynomials, we learned about how the algorithm acts with different types of input matrices. For instance, in our preliminary studies we demonstrated that even with a low tolerance, the Martinsson algorithm was able to compute the SVD more efficiently. Then, we found that in our tests for polynomial approximation that the tolerance had little effect on the relative error. However, although the tolerance had little affect we learned that the r values influence polynomials when $N = M^2$. According to our studies, when $N = M^2$, r should not exceed 9. This drastically reduces the relative error compared to using larger r values.

Additionally, we learned that interpolation with RBFs augmented with polynomials performed much better to the polynomials approximation when the polyharmonic spline was $\phi(s) = s^m$ when $m = \ell$ or $m = \ell - 1$. For both, the tolerance and r value parameters did not seem to affect the relative error. However, when we tried a new $\phi(s) = s^m$ when $m = 2\ell + 1$, the relative error decreased at a much slower rate and achieved similar relative error to polynomial approximations. This was consistent in both tests with the tolerance ϵ and the r values. In all cases (except $m = 2\ell + 1$), we found that using a loose tolerance was quite sufficient for both polynomial least squares approximation and interpolation with RBFs and polynomials. This opens up the potential to replace costly deterministic solves with randomized SVD algorithms in the context of practical applications where these techniques are used.

Our findings seem promising, but there are many studies that could be done to investigate further into how the Martinsson algorithm works and what it can be best used for. For instance, both polynomial least squares and RBF interpolation are applied in a local setting (rather than as global approximation

schemes). It seems likely that our findings would carry over to that case also, opening up the possibility of computing scattered node finite difference formulas efficiently using our mix of deterministic and randomized techniques. On the other hand, both polynomial and RBF approximants are used for *global* high-dimensional approximation. The techniques described herein are directly applicable to those contexts also.

Bibliography

[1] Victor Bayona, Miguel Moscoso, Manuel Carretero, and Manuel Kindelan. "RBF-FD formulas and convergence properties". In: *Journal of Computational Physics* 229.22 (2010), pp. 8281–8295. DOI: 10.1016/j.jcp.2010.07.008.

[2] Victor Bayona Revilla, Natasha Flyer, Bengt Fornberg, and Gregory A. Barnett. "On the role of polynomials in RBF-FD approximations: II. Numerical solution of elliptic PDEs". English. In: *Journal of Computational Physics* 332 (Mar. 2017), pp. 257–273. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2016.12.008.

[3] Christos Boutsidis, Michael W. Mahoney, and Petros Drineas. "An Improved Approximation Algorithm for the Column Subset Selection Problem". In: *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms* (Apr. 2009). DOI: 10.1137/1.9781611973068.105.

[4] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. "Fast Monte Carlo Algorithms for Matrices II: Computing a Low-Rank Approximation to a Matrix". In: *SIAM Journal on Computing* 36.1 (2006), pp. 158–183. DOI: 10.1137/s0097539704442696.

[5] Gregory E. Fasshauer. *Meshfree approximations methods with MATLAB*. World Scientific, 2007.

[6] Natasha Flyer, Bengt Fornberg, Victor Bayona, and Gregory A. Barnett. "On the role of polynomials in RBF-FD approximations: I. Interpolation and accuracy". In: *Journal of Computational Physics* 321 (2016), pp. 21–38. DOI: 10.1016/j.jcp.2016.05.026.

[7] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. "Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions". In: *SIAM Review* 53.2 (2011), pp. 217–288. DOI: 10.1137/090771806.

[8] Harald Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, 1992.

[9] T. Sarlos. "Improved Approximation Algorithms for Large Matrices via Random Projections". In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. 2006, pp. 143–152.

[10] *Schur complement*. Oct. 2020. URL: https://en.wikipedia.org/wiki/Schur_complement.

[11] Varun Shankar. "The overlapped radial basis function-finite difference (RBF-FD) method: A generalization of RBF-FD". In: *Journal of Computational Physics* 342 (2017), pp. 211–228. DOI: 10.1016/j.jcp.2017.04.037.

[12] Varun Shankar, Akil Narayan, and Robert M. Kirby. "RBF-LOI: Augmenting Radial Basis Functions (RBFs) with Least Orthogonal Interpolation (LOI) for solving PDEs on surfaces". In: *Journal of Computational Physics* 373 (2018), pp. 722–735. DOI: 10.1016/j.jcp.2018.07.015.

[13] Varun Shankar, Grady B. Wright, Aaron L. Fogelson, and Robert M. Kirby. "A radial basis function (RBF) finite difference method for the simulation of reaction-diffusion equations on stationary platelets within the augmented forcing method". In: *International Journal for Numerical Methods in Fluids* 75.1 (2014), pp. 1–22. DOI: 10.1002/fld.3880.

[14] Holger Wendland. *Scattered data approximation*. Cambridge University Press, 2010.

[15] Grady Wright and Bengt Fornberg. "Scattered Node Compact Finite Difference-Type Formulas Generated From Radial Basis Functions". In: *Computational Methods* (), pp. 1391–1395. DOI: 10.1007/978-1-4020-3953-9_59.