Regularized Symmetric Positive Definite Matrix Factorizations for Linear Systems arising from RBF interpolation and differentiation

Scott A. Sarra
Marshall University

April 24, 2014

Abstract

Scattered data interpolation using Radial Basis Functions involves solving an ill-conditioned symmetric positive definite (SPD) linear system (with appropriate selection of basis function) when the direct method is used to evaluate the problem. The standard algorithm for solving a SPD system is a Cholesky factorization. Severely ill-conditioned theoretically SPD matrices may not be numerically SPD (NSPD) in which case a Cholesky factorization fails. An alternative symmetric matrix factorization, the square root free Cholesky factorization, has the same flop count as a Cholesky factorization and is successful even when a matrix ceases to be NSPD. A regularization method can be used prevent the failure of the Cholesky factorizaiton and to improve the accuracy of both SPD matrix factorizations when the matrices are severely ill-conditioned. The specification of the regularization parameter is discussed as well as convergence/stopping criteria for the algorithm. Formation of differentiation matrices with the regularized SPD factorizations is demonstrated to improve eigenvalue stability properties of RBF methods for hyperbolic PDEs.

1 Introduction

The use of Radial Basis Function (RBF) methods for scattered data interpolation problems and for solving time-dependent Partial Differential Equations (PDEs) entails solving linear systems that are very ill-conditioned when small shape parameters are used and when the direct method is used to evaluate the problem. The matrices are symmetric and with proper choice of

basis function are symmetric positive definite (SPD). The de facto standard algorithm for factorizing SPD matrices is the Cholesky factorization which is available in most, if not all popular numerical linear algebra software libraries. For use with RBF methods a factorization that is not as prevalent in standard numerical linear algebra libraries, the square root free Cholesky factorization, may be a better choice. In general, the square root free Cholesky factorization factorization may be preferable for SPD matrices since it avoids the need to calculate square roots [19].

With either SPD factorization, a regularization technique that is implemented iteratively can be used to implement RBF methods over a wide range of shape parameter in a stable manner. The regularization technique has been previously used in the literature with RBF methods ([7],[28], and [2]) but it is unclear how to specify the regularization parameter of the method and what criteria should be used to terminate the iteration. The use of regularized SPD (RSPD) factorizations have not previously been described for RBF methods for time-dependent PDEs. The implementation and efficiency of the RSPD factorization in two popular scientific computing environments, Matlab [27] and the Python package SciPy [22], is examined.

We note that the RBF interoplation method in not intrinsically ill-conditioned, but it is the direct approach of solving the linear system determined by interpolation conditions that may be ill-conditioned. So-called bypass algorithms have been developed that evaluate the RBF interpolation problem in a stable manner for all values of the shape parameter without directly dealing with the linear system. Two bypass algorithms are the Contour-Padé algorithm [12] and the RBF-QR algorithm [11]. Both algorithms have been valuable theoretical tools but have limitations as far as serious applications are concerned. The Contour-Padé algorithm is limited by that only a small number of centers can be used. Early implementations of the RBF-QR method suffered from a large flop count. However, improvements to the RBF-QR algorithm continue to be made [8] and it has found uses in applications for the purposes of generation of RBF finite difference stencils [10] and differentiation matrices [24]. For now, the direct method which solves a linear system remains the most prevalent method used to evaluate a RBF interpolation problem.

In addition to bypass methods, another approach that has been used to allow the RBF method to be more accurate is the use of extended precision floating point arithmetic in the direct method [20, 35]. Some author's have found the extended precision preferable to the use of bypass algorithms, for example as stated in [25]: "Expensive strategies for a well-conditioned basis rearrangement have been developed by Fornberg and collaborators,

but we prefer Sarra's strategy, always simpler and often faster, of using multiple precision arithmetic". For the remainder of this work, only the direct method implemented with double precision is considered.

2 RBF Interpolation and Differentiation

RBF interpolation uses a set of N distinct points $X = \{x_1^c, \ldots, x_N^c\}$ in $\mathcal{R}^d$ called centers. No restrictions are placed on the shape of problem domains or on the location of the centers. A RBF

$$\phi(x) = \phi(\|x - x^c\|_2, \varepsilon), \qquad x, x^c \in \mathcal{R}^d \tag{1}$$

is an infinitely differentiable (compactly supported and global RBFs without a shape parameter and with less smoothness exist but are not considered in this work) function of one variable $r = \|x - x^c\|_2 \geq 0$ that is centered at x^c and that contains a free parameter ε called the shape parameter. The RBF interpolant assumes the form

$$\mathcal{I}_N f(x) = \sum_{k=1}^{N} a_k \phi\left(\|x - x_k^c\|_2, \varepsilon_k\right) \tag{2}$$

where a is a vector of expansion coefficients. The inverse quadratic (IQ) RBF

$$\phi(r) = \frac{1}{1 + \varepsilon^2 r^2} \tag{3}$$

is used throughout. The IQ is a representative member of the class of global, infinitely differentiable RBFs that have a shape parameter and that interpolate with exponential accuracy. The expansion coefficients are determined by enforcing the interpolation conditions

$$\mathcal{I}_N f(x) = f(x_k^c), \qquad k = 1, 2, \ldots, N \tag{4}$$

which result in a $N \times N$ linear system

$$Ba = f. \tag{5}$$

The matrix B with entries

$$b_{jk} = \phi(\left\|x_j^c - x_k^c\right\|_2, \varepsilon_k), \qquad j, k = 1, \ldots, N \tag{6}$$

is called the system matrix. The evaluation of the interpolant (2) at M points x_j is accomplished by multiplying the expansion coefficients by the $M \times N$ evaluation matrix H that has entries entries

$$h_{jk} = \phi(\|x_j - x_k^c\|_2, \varepsilon_k), \qquad j = 1, \ldots, M \text{ and } k = 1, \ldots, N. \tag{7}$$

By linearity, the RBF interpolant can be differentiated as

$$\mathcal{D}\left(\mathcal{I}_N f(x)\right) = \sum_{k=1}^{N} a_k \mathcal{D}\phi\left(\|x - x_k^c\|_2, \varepsilon_k\right) \tag{8}$$

where $\mathcal{D}$ is a linear differential operator. The operator $\mathcal{D}$ may be a single differential operator or a linear differential operator such as the Laplacian. Evaluating (8) at the centers X can be accomplished multiplying the expansion coefficients by the evaluation matrix $H_{\mathcal{D}}$ with entries

$$h_{jk} = \mathcal{D}\phi(\left\|x_j^c - x_k^c\right\|_2, \varepsilon_k), \qquad j, k = 1, \ldots, N. \tag{9}$$

That is, $\mathcal{D}f \approx H_{\mathcal{D}}a$. Alternatively, derivatives can be approximated by multiplying the grid function values $\{f(x_k^c)\}_{k=1}^N$ by the differentiation matrix $D = H_{\mathcal{D}}B^{-1}$ since

$$\mathcal{D}f \approx H_{\mathcal{D}}a = H_{\mathcal{D}}(B^{-1}f) = (H_{\mathcal{D}}B^{-1})f. \tag{10}$$

The shape parameter ε_k may take on different values at each center x_k^c. Such an approach is called a variable shape parameter strategy. Several variable shape strategies [23, 38] have been suggested and present some advantages. A drawback of variable shape parameter strategies is that they cause the RBF system matrix to be non-symmetric. A constant shape parameter results in the RBF system matrix being symmetric and for this reason a constant shape parameter has been used throughout.

Both equations (5) for the expansion coefficients and (10) for the differentiation matrix assume that the system matrix is invertible. The IQ system matrix is SPD and thus invertible. While it is invertible, the system matrix is typically very poorly conditioned. The eigenvalues of B satisfy $0 < \lambda_{min} = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N = \lambda_{max}$ and the matrix condition number in the 2-norm of is $\kappa(B) = \lambda_{max}/\lambda_{min}$. The eigenvalues of the IQ system matrix satisfy the bounds [42]

$$\lambda_{\max} \leq N \tag{11}$$

and

$$\lambda_{\min} \geq K(\varepsilon, d)\, q_X^{-d/2-1/2} \exp\left(\frac{-2C_d}{\varepsilon\, q_X}\right). \tag{12}$$

The constant $C_d \leq 6.32d$ depends on the space dimension d and the constant $K(\varepsilon, d)$ depends on the shape parameter and the space dimension [42, 6]. The minimum separation distance is defined as

$$q_X = \frac{1}{2} \min_{i \neq j} \left\| \mathbf{x}_i^c - \mathbf{x}_j^c \right\|_2 . \tag{13}$$

The exponentially decaying bound for the smallest eigenvalue is the main factor in the system matrix becoming severely ill-conditioned as the shape parameter and minimum separation distance are decreased. The exponentially decaying lower bound (12) for the minimum eigenvalue may be overly pessimistic as numerical experiments indicate that the smallest eigenvalues may actually decay at an algebraic rate [13]. A regularization technique will be used in section 4 to increase the size of the minimum eigenvalue of the matrix that is actually factorized.

For a fixed set of centers, the shape parameter affects both the accuracy of the method and the conditioning of the system matrix. The RBF method is most accurate for smaller values of the shape parameter where the system matrix is ill-conditioned. The attainable error and the condition number of the system matrix cannot both be kept small. In connection with the RBF-direct approach, this relationship has been dubbed the uncertainty principle [39].

Recent monographs [3, 6, 37, 42] on RBF methods can be consulted for more information.

3 Direct methods for SPD systems

The most popular method for the solution of a general linear system $Ba = f$ is LU factorization which partial pivoting [40] which has as the leading term in its flop count $\frac{2N^3}{3}$. A SPD matrix can be factorized using half as many flops via Cholesky factorization, $B = LL^T$, where L is lower triangular. Throughout, the Cholesky factorization is referred to as a LL factorization. Both Matlab and SciPy wrap the LAPACK [1] routine DPOTRF for LL factorization. A theoretically SPD matrix may not be numerically SPD (NSPD). In this case, an attempted application of the LL factorization algorithm encounters a square root of a negative number and fails.

The square root free Cholesky factorization is $B = LDL^T$ where L is lower triangular with ones on the main diagonal and D is a diagonal matrix. Throughout, the square root free Cholesky factorization is referred to as a LDL factorization. The diagonal elements of D are the quantities in the LL

factorization algorithm that are the arguments of the square root function. When the matrix being factorized is NSPD all the diagonal elements of D are greater than zero.

Both the LL and LDL algorithms for SPD matrices are backward stable without the need for pivoting [19]. Pseudocode for both factoization algorithms can be found in standard linear algebra references such as [15]. The two algorithms have essentially the same flop count. The major difference between the two algorithms is that when a SPD matrix is not NSPD, the LL factorization fails whereas the LDL factorization does not. After a matrix has been factorized using either, LU, LL, or LDL, $\mathcal{O}\left(N^2\right)$ forward substitution and back substitution algorithms are used to solve a linear system.

The LDL factorization should not be confused with the factorization $B = PLDL^TP^T$ for symmetric indefinite matrices [19] where P is a permutation matrix and D is a block diagonal matrix with diagonal blocks of dimension one or two. The algorithm is sometimes called a block LDL factorization.

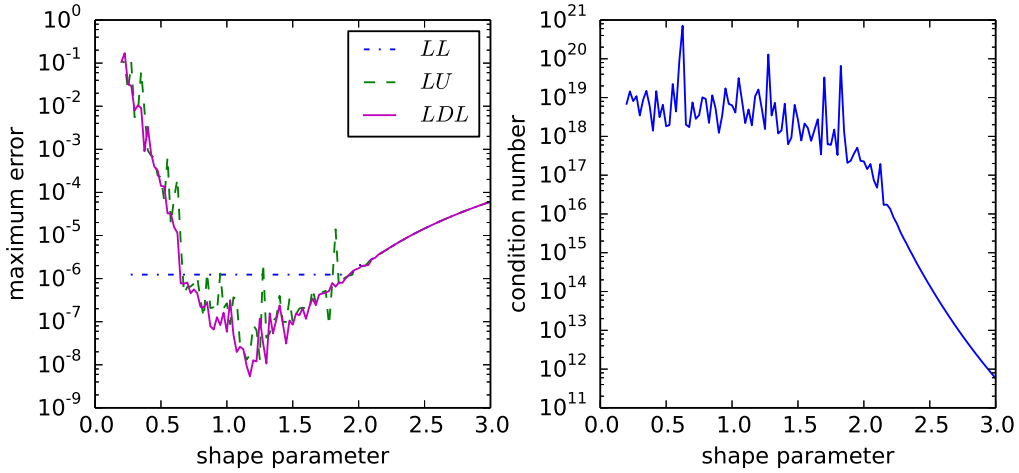

Figure 1: LL, LDL, and LU factorizations utilized for interpolating function (14) over a range of shape parameters.

The LDL factorization is not as prevalent in numerical software as is the LL factorization. The majority of Matlab and SciPy numerical linear algebra routines wrap LAPACK routines and the LDL algorithm is not a LAPACK routine. Matlab has had the function LDL since version 7.3 (R2006b) but it implements the block LDL algorithm $B = PLDL^TP^T$ for symmetric indefinite matrices and not the square root free Cholesky algorithm. The

LDL algorithm is a part of the add on Signal Processing (DSP) toolbox for Matlab. SciPy does not feature a LDL factorization. In the examples, a C implementation of the LDL algorithm that is wrapped in Python has been used.

This section ends with an example in order to establish terms and illustrate features of RBF interpolation. The function

$$f(x) = e^{\sin(\pi x)} \tag{14}$$

is interpolated at $N = 55$ equally spaced centers on the interval $[-1, 1]$ and the interpolant is evaluated at $M = 175$ equally spaced centers. The left image of figure 1 displays the error versus the shape parameter resulting from solving the problem with three different matrix factorization. The condition number of the system matrix versus the shape parameter is in the right image of the figure. The error curve decreases smoothly as the shape parameter decreases until approximately $\varepsilon = 2.1$ is reached at which the condition number of the system matrix is $\mathcal{O}(10e16)$. Below this point numerical instability sets in and as a result the error curve oscillates but still trends in an overall decreasing direction until $\varepsilon = 1.2$ is reached. LL factorization fails at $\varepsilon = 1.95$ and lower due to the system matrix failing to be NSPD. The horizontal line above smaller shape parameters in the left image of the figure indicates that the computation is not possible in double precision using a LL factorization. The LDL factorization also detects that the system matrix is not NSPD at $\varepsilon = 1.95$ and lower by producing elements of the diagonal matrix D that are less than zero. Unlike the LL factorization, the LDL factorization does not fail and instead produces results as good as LU factorization, but at half the computational cost. At the smallest shape parameter used, $\varepsilon = 0.3$ the smallest negative diagonal element of the matrix D of the LDL factorization is -1.15e-12. The range of shape parameter for which both the LDL computation is stable and for which D has positive diagonal elements and that the LL algorithm succeeds and is stable can be increased by a large amount by using a regularization technique as described in the next section.

4 Regularized SPD (RSPD) matrix factorizations

Instead of solving the system

$$Ba = f \tag{15}$$

the regularized system

$$Cy = f \tag{16}$$

where $C = B + \mu I$ is solved. The parameter μ is a small positive constant called the regularization parameter and I is the identity matrix. The technique is called the method of diagonal increments (MDI) and its first use dates back to the 1940's [29]. Matrix C is better conditioned than B as

$$\kappa(C) = \frac{\lambda_{max} + \mu}{\lambda_{min} + \mu} < \kappa(B) = \frac{\lambda_{max}}{\lambda_{min}}.$$

For small μ, $(B + \mu I)^{-1}$ is close to B^{-1} and MDI simply replaces B with $(B + \mu I)$ in computing the solution of a system. Equation

$$B^{-1} - (B + \mu I)^{-1} = \mu^2 B^{-1}(I + B^{-1}/\mu)B \tag{17}$$

quantifies how close that $(B + \mu I)^{-1}$ and B^{-1} are [16, 18]. For very small μ the difference is negligible.

Rather than just accepting a solution that is close to the desired solution is was shown in the 1950's that an additional step [33] could be taken to recover more accuracy. The solution a of (15) can be computed from a series expansion involving the solution y of the regularized system (16) as

$$B^{-1} = \frac{1}{\mu} \sum_{k=1}^{\infty} \left(\mu C^{-1}\right)^k. \tag{18}$$

Series (18) converges since the spectral radius of μC^{-1} is less than one as

$$0 < \frac{\mu}{\lambda_i + \mu} < 1, \qquad i = 1, 2, \ldots, N$$

for all $\mu > 0$. $\kappa(C)$ may be significantly smaller than $\kappa(B)$ if $\mu > \lambda_{min}$. If $\mu \ll \lambda_{min}$ then $\mu/(\lambda_i + \mu) \ll 1$ for all i the series converges rapidly. The parameter μ needs to be selected to be large enough to improve conditioning but small enough so that the convergence of the method is fast.

The solution of the original system (15) can be written as

$$\begin{aligned} a &= B^{-1} f && (19) \\ &= \frac{1}{\mu} \sum_{k=1}^{\infty} \left(\mu C^{-1}\right)^k f && (20) \\ &= \frac{1}{\mu} \sum_{k=1}^{\infty} \left(\mu C^{-1}\right)^{k-1} y && (21) \\ &= y + \mu C^{-1} y + \left(\mu C^{-1}\right)^2 y + \cdots && (22) \end{aligned}$$

Reorganizing equation 22 as

$$a = y + \mu C^{-1}\left[y + \mu C^{-1} y + \cdots\right] \tag{23}$$

allows for a to be recovered iteratively. The first term of the solution series is the solution of the regularized system, that is $a^{[0]} = C^{-1}f$, and then repeated correction terms

$$a^{[k+1]} = a^{[k]} + \mu C^{-1} a^{[k]}, \qquad k = 0, 1, \ldots \tag{24}$$

are taken to recover the solution of the original system. Each calculation of a correction term via equation (24) is called an iteration. In the modern literature the method has become known as Riley's method after the author of [33].

In summary, to implement the method B is factorized once using $\mathcal{O}\left(\frac{1}{3}N^3\right)$ flops and then the factorization is used to solve systems with $\mathcal{O}\left(N^2\right)$ flops. First $Cy = f$ is solved and then $C\hat{a} = a^{[k]}$ is repeatedly solved for $k = 0, 1, \ldots$ and the solution is updated according to (24) until a maximum number of iterations have been reached or convergence criteria are satisfied.

In recent years, from the recommendations of [7], Riley's method has seen application in solving RBF interpolation problems ([28] and [2]). At this point what remain as open questions are how to best specify the regularization parameter and how to specify stopping criteria for the iteration in order to best balance accuracy and efficiency.

4.1 regularization parameter choice and stopping criteria

Previous recommendations for values of the regularization parameter include the following. In first paper to describe the method [33] it was recommended that $\mu = 10^{\alpha - p}$ where p is the desired precision and $\alpha = 2$ or 3. In double precision at most fifteen decimal of accuracy can be expected, thus a potential regularization parameter would be 1e-11 according to this guidance. The numerical examples in [7] used the value $\mu =$ 1e-10 but no information about the number of iterations or convergence criteria was given. In the Scilab RBF toolbox [2], the default value of the regularization parameter is set to $\mu =$ 1e-6. No information about the number of iterations or convergence criteria was given in the numerical examples, but the number of iterations in the computer code is hard coded to two. Riley's method was used to evaluate the RBF interpolation problem in [28] but no information on regularization parameter selection or convergence criteria was given.

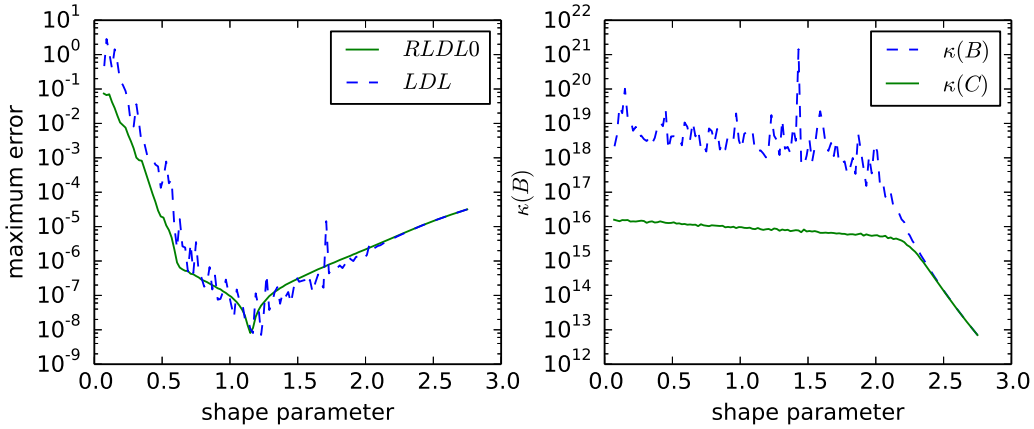

Figure 2: Left: LDL and RLDL0 (the method of diagonal increments) with μ = 5e-15. The smallest RLDL0 error is 7.99e-9 at ε = 1.15. Right: the condition number of the regularized matrix C is approximately 10^{16} in the previously unstable region.

The author states that "choosing the regularization parameter μ to maximize stability but minimize the summation length is not a straightforward procedure, but we wont concern ourselves with it here."

The interpolation example from section 3 is repeated throughout this section and is solved by several RSPD algorithms with varying parameter choices. The results in figure 2 use the RLDL0 method with μ =5e-15 and compares the accuracy of the method with the LDL factorization over a range of the shape parameter. The figure illustrates that the RLDL0 algorithm can produce more accurate solutions in the shape parameter range for which the system matrix is severely ill-conditioned and also produce a non-oscillatory error curve over a large range of shape parameters for which the LDL factorization could not. The details of the algorithm and the explanation of parameter choices follow.

As illustrated by the example in figure 1, the linear system is being solved accurately and in a stable manner (no oscillation of the error curve) with system matrices with condition number of 10^{16} and less. However, the smallest error is achieved in the unstable region with ε = 1.15. At this value of the shape parameter the system matrix has a condition number of 1.4e19 and smallest eigenvalue of -7.23e-16 when calculated in double precision. Neither value is accurate. Using extended precision the accurate

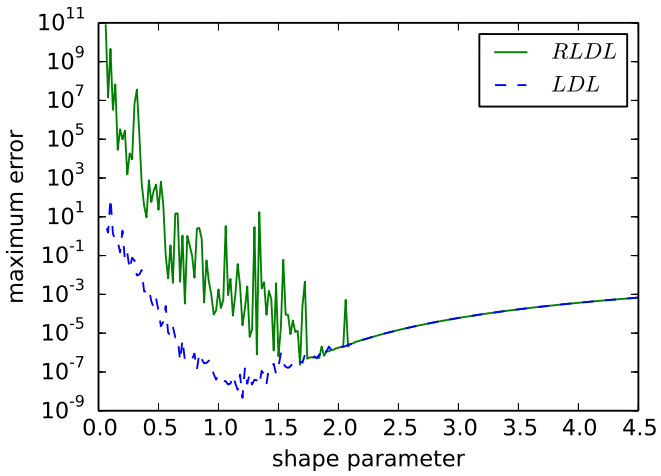

Figure 3: RSPD algorithm with μ = 5e-15 and without stopping criteria number 2 of section 4.1 after 100 iterations. Without the stopping criteria the method diverges in double precision for shape parameters smaller than $\varepsilon = 2.05$.

condition number is found to be 1.6e30 and the smallest eigenvalue is 2.2e-29. Specifying μ smaller than the minimum eigenvalue results in the series (18) in Riley's method converging quickly but doing so is impossible in this case since it results in a matrix C that has a condition number well above 10^{16} which can not be accurately factorized in double precision. However, specifying the regularization parameter well above the smallest eigenvalue of B but close to machine epsilon is effective. The range 5e-13 $\leq \mu \leq$ 5e-15 was used in all example problems that follow. For example, with μ = 5e-15 the matrix C has a condition number of 7.3e15. In extended precision, the spectral radius of μC^{-1} is 0.9999999999999954 indicating that theoretically the method converges very slowly. The actual convergence of the method depends however on what the spectral radius of μC^{-1} is when computed in double precision. In this case it is 1.17 and Riley's method diverges when implemented in double precision. Despite being divergent, the first few iterations often lead to corrections that increase the accuracy of the solution of the system.

When the matrix is NSPD the behavior of the method is different. For example, again taking μ =5e-15 but with $\varepsilon = 3.0$ results in the matrix B having a $\mathcal{O}\left(10^{12}\right)$ condition number. The condition number of C is approx-

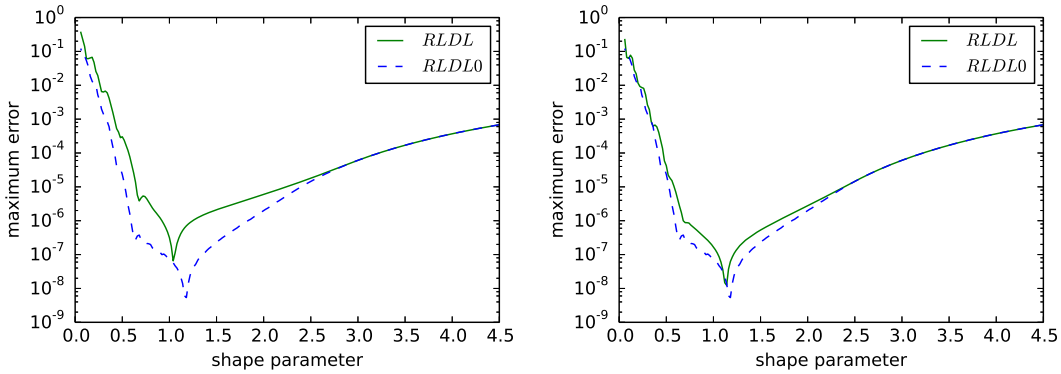

Figure 4: RLDL with μ = 1e-10 and the maximum allowable iterations set to 10 (left) and 1000 (right) compared to RLDL0 with μ = 5e-15. A larger reularization parameter causes very slow convergence.

imately the same as the condition number of B. The minimum eigenvalue of B is 3.5e-11 and the spectral radius of μC^{-1} is 1.4e-04. Both quantities can be accurately calculated in double precision. The series (18) converges rapidly to a solution that is already close to the desired solution as the difference (17) is 1e-41. In this case either zero or very few iterations of (24) are necessary. When the matrix is NSPD, a tolerance on the minimum allowable relative change should be set or the method will continue to take hundreds of iterations with small relative changes that decrease in size but that do not result in a more accurate solution.

Based on these observations, the following are recommended for use in a RSPD algorithm in calculating over a range of the shape parameter for which the system matrix may or may not be NSPD. A regularization parameter in the range 5e-13 $\leq \mu \leq$ 5e-15 is recommended along with the following three termination criteria for stopping the iteration (24):

1. Stop when size of the correction term $\mu C^{-1} a^{[k]}$ relative to the solution y of the regularized system (16) falls below a given tolerance which by default is set to 1e-4. The tolerance limits the number of iterations to zero or one if it is applied to a NSPD matrix. Otherwise the algorithm would take tiny correction steps up to the maximum number of allowable iterations (criteria 3) in this region that would not improve the solution.

2. Terminate when the relative change in criteria 1 ceases to decrease

from one iteration to the next. This indicates the method is beginning to diverge. The example of this section is repeated without using stopping criteria 2 and the result after 100 iterations is shown in figure 3. If more iterations were to be taken the solution would eventually become infinite below $\varepsilon = 2.05$.

3. For safety, the algorithm also limits the maximum number of iterations. The default number is five since most of gain in accuracy occurs in the first few iterations with the suggested regularization parameter range.

The RSPD solver with stopping criteria is as follows:

```
def RSPD(A,b,mu=5e-15,maxIt=5,tol=1e-4):
    N = len(b)
    C = A + mu*eye(N)
    L, d = LDL_factor(C)       # or L = LL_factor(C)
    y = LDL_solve(L, d, b)     # or y = LL_solve(L,b)
    sizeY = norm(y,2)
    x = y

    relativeChange, relativeChangeOld, it = 0, 1e15, 0
    while it<maxIt:
        y = mu*LDL_solve(L, d, y)
        correctionSize = norm(y,2)
        relativeChange = correctionSize/sizeY
        if relativeChange>relativeChangeOld or relativeChange<tol: break
        it += 1
        relativeChangeOld = relativeChange
        x = x + y

    return x
```

Either a LL or LDL factorization can be used in the algorithm. If a LL factorization is used μ must be large enough to ensure that the matrix C is NSPD.

Most of the improvement in the accuracy of the solution takes place in the first iteration when a small regularization parameter is used. A simpler RSPD algorithm, RSPD1, that avoids the need to test stopping criteria and instead only takes one iteration is

```
def RSPD1(A,b,mu=5e-15):
    N = len(b)
    C = A + mu*eye(N)
    L, d = LDL_factor(C)       # or L = LL_factor(C)
    x0 = LDL_solve(L, d, b)    # or y = LL_solve(L,b)
```

```
6       x1 = mu*LDL_solve(L, d, x0)
7       return x0 + x1
```

An even simpler strategy that is effective is to just use the method of diagonal increments and not use Riley iteration at all. That is, take the solution of $(B + \mu I)a = f$ to be the solution of $Ba = f$. Continuing the example with μ =5e-15 that has been carried through this section, the difference between B^{-1} and $(B + \mu I)^{-1}$ as given by equation (17) is 1.67e-40 in the 2 norm. In addition to MDI this approach is also called RSPD0 to indicate that zero iterations of equation (24) are taken. Figure 5 compares the accuracy of the RSPD, RSPD1, and RSPD0 algorithms using the default parameter values.

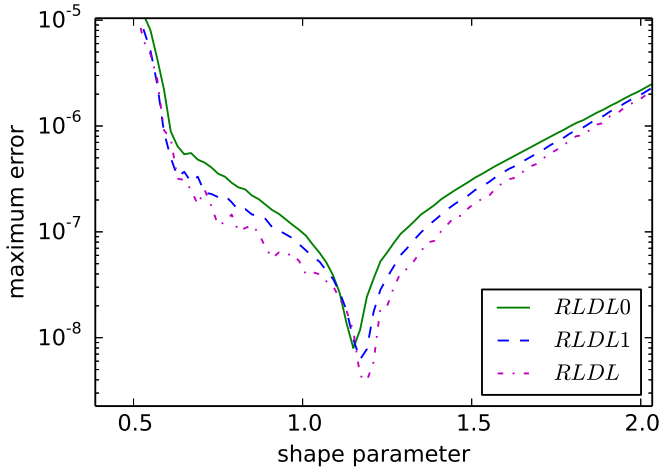

Figure 5: The smallest RLDL0 error is 7.99e-9, the smallest RLDL1 error is 6.24e-9, and the stopping criteria of RLDL terminates the algorithm after 4 iterations with a minimum error of 3.91e-9.

Previous works ([7] and [2]) suggested much larger values of the regularization parameter but our experience with using the recommended values was that the convergence of the method was extremely slow. For example, the previous example is repeated with $\mu = 1e$-10. The larger regularization parameter keeps the condition number of the regularized system under 10^{12}. However, the improvement in conditioning is at the expense of convergence. With this value of the shape parameter the difference (17) is now 1.3e-27, so the initial solution is now further away from the exact solution than if a larger regularization parameter had been used. The spectral radius of

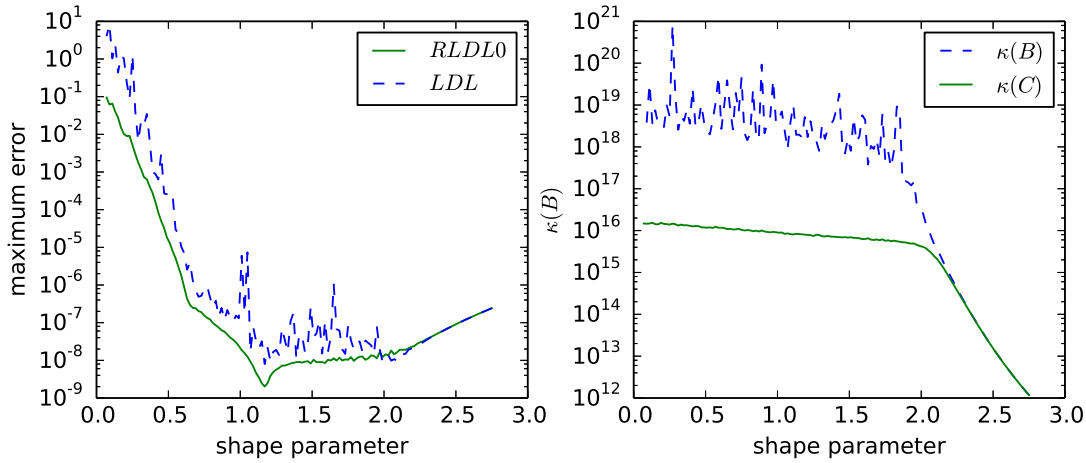

Figure 6: Interpolation with $N = 55$ non-uniformly spaced centers with location given by equation (25). The interpolant is evaluated at 175 evenly spaced centers. With a regularization parameter of μ =5e-15 the smallest RLDL0 error is 2.02e-9 at $\varepsilon = 1.17$.

μC^{-1} is 0.9999999999, so in addition to being further away from the desired solution, the convergence will be very slow. The results after 10 and 1000 iterations are shown in figure 4. After 10 iterations the smallest error is 6.53-8 and after 1000 iterations the minimum error is 1.24e-8. With $\mu = 1e$-10 the error is larger after 1000 iterations than it was after after zero iterations with RSPD0 with $\mu = 5e$-15.

The example of this section is repeated one more time with unequally spaced centers with locations given by

$$x_k = \sin^{-1}\left(-\gamma \cos(k\pi/(N-1))\right) / \sin^{-1}(\gamma) \tag{25}$$

with $\gamma = 0.99$ and for $k = 0, 1, \ldots, N-1$.. The output of the RLDL0 algorithm with $\mu = 5e$-15 is shown in the left image of figure 6 and the condition number of the system matrix and the regularized matrix C are shown in the right image of the figure. The smallest error of 2.02e-9 is at $\varepsilon = 1.17$.

As the number of centers increase and/or the shape parameter decreases, RBF interpolants with equally spaced centers and constant shape parameters suffer from the Runge phenomena in a similar manner as their closely related global polynomial interpolants do ([31], [30]). For example, with a

fixed value of the shape parameter, a RBF interpolant of a smooth function on a closed interval will not converge uniformly as $N \to \infty$ unless the function is analytic in a larger region of the complex plane in which the shape of the region depends on the shape parameter and on the location of the centers. Clustering centers more densely near the ends of the interval avoids this difficulty, but unfortunately, analytic expressions for the location of such centers are not known and they need to be computed for each RBF and shape parameter.

Reference [30] shows that a good set of centers for avoiding the Runge phenomena while interpolating the function

$$f(x) = \sin(10x) \tag{26}$$

on the interval $[-1, 1]$ with the IQ RBF and shape parameter $\varepsilon = 1.0$ is given by equation (25) with $\gamma = 0.9$. Figure 7 displays the maximum error of interpolating the function with an increasing number of centers using both LDL and RLDL0 in double precision (in which numbers are accurately represented to approximately fifteen decimal places). In double precision the convergence trend is not able to continue beyond $N = 25$. Beyond this point the LDL error curve begins to oscillate and increase while the RLDL error curve remains approximately constant. The LDL calculation is repeated in extended precision with 63 decimal places of accuracy and the convergence trend continues while avoiding the Runge phenomena.

5 Numerical examples

In this section the RSPD solvers are used in 2d scattered data interpolation, to form differentiation matrices for the solution of time-dependent PDE problems, and are bench marked for efficiency.

5.1 Franke function

The Franke function

$$\begin{aligned} f(x,y) &= \frac{3}{4}e^{\left[\frac{-1}{4}(9x-2)^2-\frac{1}{4}(9y-2)^2\right]} + \frac{3}{4}e^{\left[\frac{-1}{49}(9x+1)^2-\frac{1}{10}(9y+1)^2\right]} \\ &= \frac{1}{2}e^{\left[\frac{-1}{4}(9x-7)^2-\frac{-1}{4}(9y-3)^2\right]} - \frac{1}{5}e^{\left[-(9x-4)^2-(9y-7)^2\right]} \end{aligned} \tag{27}$$

has often been used as a test function for RBF methods since it was first introduced in a paper [14] on scattered data approximation methods in 1982. The function is interpolated over a range of shape parameters using 618

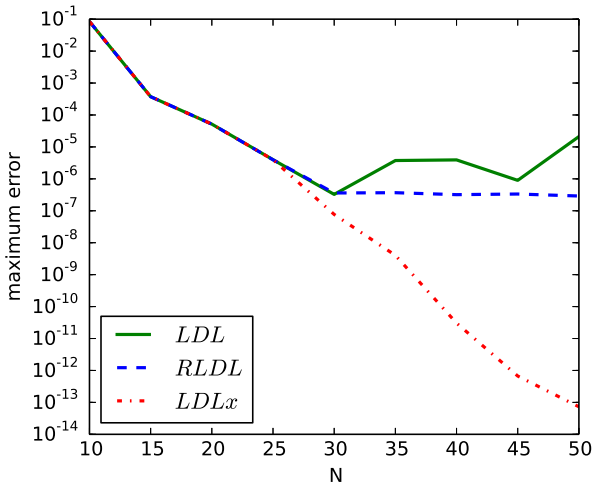

Figure 7: LDL, RLDL0 (with μ = 5e-15), and LDL with extended precision (decimal precimal of 63) floating point arithmetic (LDLx) for interpolating function (26) over a range of N and a fixed shape parameter of $\varepsilon = 1$.

scattered centers in a quarter circular domain shown in figure 8. The interpolant is evaluated at 900 points within the domain. The left image of figure 9 compares the accuracy from using a LDL solver and the RLDL0 solver. The system matrix is not NSPD for $\varepsilon < 4.475$ and a LL factorization would fail in this range. With a regularization parameter of μ =1e-14 the RLDL0 solver has a non-oscillatory error curve over the entire range of shape parameter used and is about two decimal places more accurate in the shape parameter region where the system matrix is severely ill-conditioned.

5.2 Efficiency of RSPD solver for RBF systems

Table 1 lists the execution time in seconds of six Matlab and seven Python implementations of methods for solving 1000 IQ RBF systems with N = 500. All the reported results were carried out on a computer running 64-bit Windows 7 and that was equipped with an Intel I7-950 quad-core @ 3.07 GHz CPU and 12 GB RAM. The times in the middle column result from solving systems with matrices that are NSPD while the results in the right column are from systems with matrices that are not NSPD. The algorithms listed in the left column that begin with a M are Matlab functions and that begin with a P are Python functions. The particular algorithms and

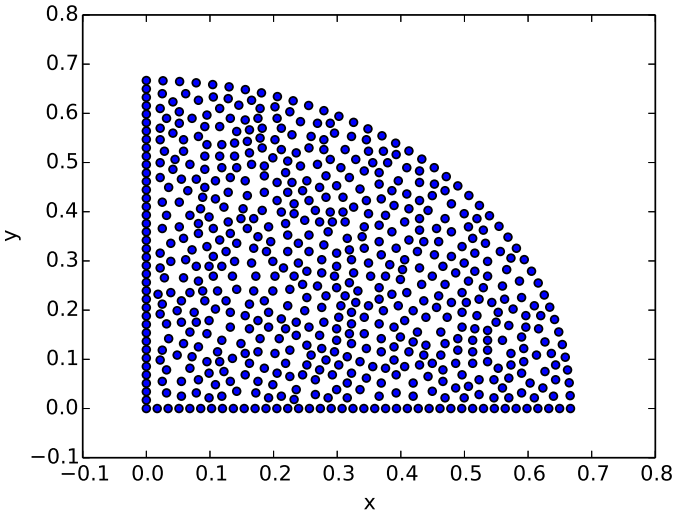

Figure 8: 618 scattered centers on a non-square domain.

function calls are:

M-backslash The backslash operator or equivalently the function mldivide. A black box solver that attempts a Cholesky factorization of a symmetric matrix. If the factorization fails it determines the matrix is not SPD and instead uses LU decomposition with partial pivoting.

M-LL Cholesky factorization via the function chol.

M-LDL Block LDL^T factorization for symmetric indefinite matrices by the function LDL.

M-RLL0 The method of diagonal increments. Cholesky factorization of $B+\mu I$ via the chol function and then the factorization is used to solve $(B + \mu I)a = f$.

M-RLL1 Solution of the regularized system (16) via the chol function and one iteration of (23).

M-RLL Solution of the regularized system (16) via the chol function and a number of iterations of (23) determined by the stopping criteria of section 4.1.

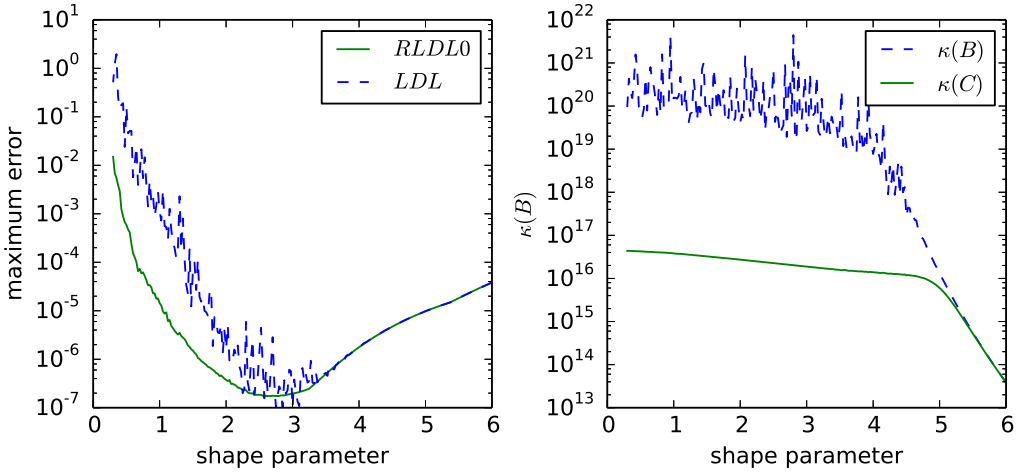

Figure 9: Interpolation of the Franke function (27). Left: The minimum RLDL0 error with μ =1e-14 is 1.73e-07 at the shape parameter 2.62. Right: The condition number of the system matrix B and of the regularized matrix C.

P-solve The function SciPy.linalg.solve. By default it performs LU factorization with partial pivoting. It accepts an argument to tell the function that the matrix is SPD which causes the function to use Cholesky factorization instead.

P-LL Cholesky factorization via the functions SciPy.linalg.cho_factor and SciPy.linalg.cho_solve.

P-LL* Cholesky factorization via the functions SciPy.linalg.lapack.dpotrf and SciPy.linalg.lapack.dpotrs. The same as the algorithms in P-LL but without the overhead of error checking.

P-LDL LDL factorization implemented in C and wrapped in Python. The method is not implemented using BLAS subroutines and thus not optimized for multi-core CPUs.

P-RLL0 The method of diagonal increments. Cholesky factorization of $B + \mu I$ via SciPy.linalg.lapack.dpotrf and solution of $(B + \mu I)a = f$ with SciPy.linalg.lapack.dpotrs.

P-RLL1 Cholesky factorization via the function SciPy.linalg.lapack.dpotrf

	$\varepsilon = 25$, NSPD	$\varepsilon = 15$, not NSPD
M-backslash	3.26	11.41
M-LL	2.84	n.a.
M-LDL (block)	18.61	28.3
M-RLL0	3.61	3.61
M-RLL1	4.11	4.11
M-RLL	5.04	5.04
P-solve	3.52	5.09
P-LL	4.44	n.a.
P-LL*	2.83	n.a.
P-LDL	38.08	38.08
P-RLL0	3.06	3.06
P-RLL1	3.25	3.25
P-RLL	3.71	3.71

Table 1: Time in seconds needed to solve 1000 500×500 linear systems.

and one iteration of (23) using SciPy.linalg.lapack.dpotrs to solve the system.

P-RLL Cholesky factorization via the functions SciPy.linalg.lapack.dpotrf and SciPy.linalg.lapack.dpotrs and a number of iteration of (23) determined by the stopping criteria of section 4.1.

Without optimized LDL factorizations available either in Matlab or SciPy, the Cholesky factorization remains the most efficient solver in both packages. Both Matlab and SciPy wrap the LAPACK functions dpotorf and dpotrs for their Cholesky functions. The Matlab block LDL function and the LDL function coded in C and wrapped in Python were the slowest algorithms. The Matlab and SciPy Cholesky functions are based on basic linear algebra subroutines (BLAS) from the highly optimized Math Kernel Library (MKL) [21] which are able to take advantage of multicore CPUs. The P-LDL function coded in C does not use BLAS routines and runs considerably slower as it uses only one CPU thread.

Reference [7] reports that their RLL implementation is five percent slower than M-backslash. The regularization parameter choice and the stopping criteria used here lead to a different conclusion. When the system is NSPD, the RSPD solvers are either slightly more or less efficient than M-backslash, depending on whether the Matlab or Python implementation is

considered. When the system is not NSPD and regularization is required, all the RSPD solvers implemented in either language are both more efficient and more accurate than M-backslash. In particular, P-RLL0 is over four times as fast.

In many RBF related works, such as in this work, plots are made over a range of shape parameters and computations are made with NSPD matrices and matrices that are not NSPD. M-backslash is perhaps the most popular algorithm used for such a calculation, but the RSPD solvers that have been described in this work are more accurate and more efficient choices.

5.3 Comparison to the truncated SVD

The singular value decomposition (SVD) $A = U\Sigma V^T$ of an invertible square matrix is known as a very stable factorization for use in solving linear systems [15]. However, the dominant term in the SVD flop count is $2N^3$ which is six times larger than that of both the LL and LDL factorizations. The diagonal matrix $\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_N)$ contains the singular values of the matrix that satisfy $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_N > 0$. A simple regularization method that is often effective for solving extremely ill-conditioned linear systems is the truncated SVD (TSVD) [17]. The TSVD disregards singular values that are less than a regularization parameter μ. The logic behind the truncation is that very small singular values may not be calculated accurately. The TSVD [4] and its variants [26, 5, 41] have been employed in the implementation of RBF methods. Here, the experience of [37] where the TSVD was found not to be a particularly effective solver for use in RBF methods is echoed as well as the conclusion of reference [4] where it was stated that the TSVD "cannot be applied in general to large scale problems, because the technique that eliminates the singular values simultaneously removes their respective singular eigenvectors, degrading the basis of the space."

To illustrate these points the Franke function interpolation problem from section 5.1 is repeated using the TSVD. In the left image of figure 10 the error versus the shape parameter is shown for three values of the regularization parameter and for the full SVD. For this problem, a value of the regularization parameter that resulted in better accuracy than the full SVD could not be found. In the right image of the figure, the accuracy of LDL, RLDL0, and the SVD are compared. The SVD is more accurate than LDL, but the accuracy of RLDL0 and the SVD are similar and RLDL0 is slightly more accurate when the system matrix is very ill-conditioned. In this example, RLDL0 performs slightly better than the SVD while being six times more efficient.

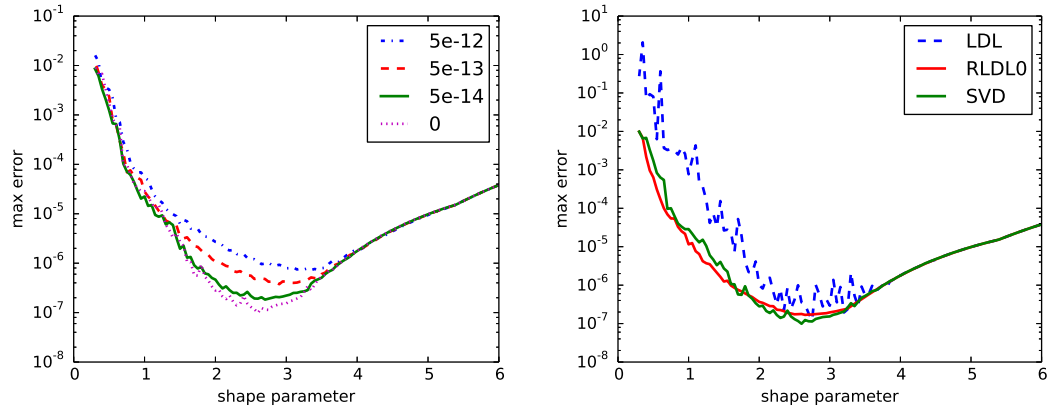

Figure 10: Repetition of the Franke function interpolation problem that is illustrated in figure 9 using the truncated SVD. Left: TSVD using four different regularization parameters. Right: accuracy of LDL, RLDL0, and the SVD.

5.4 Differentiation matrices and eigenvalue stability

RBF methods for time-dependent hyperbolic PDEs are plagued by eigenvalue stability issues which have been examined in several works [9, 32, 34]. Frequently, the discretized linear operators have eigenvalues with large positive real parts which prevent stable time integration. The problem is particularly severe [34] with small shape parameters for which the methods are most accurate but for which the system matrices are severely ill-conditioned and are not NSPD.

Figure 11 compares the accuracy of LU factorization with partial pivoting and RLDL0 in calculating the derivative of the interpolant from figure 6. The smallest RLDL0 error is 4.45e-7 at $\varepsilon = 1.18$. With this value of the shape parameter, the space derivative of the 1d advection equation

$$\frac{\partial u}{\partial t} - \frac{\partial u}{\partial x} = 0 \tag{28}$$

is discretized on the interval $\Omega = [-1, 1]$ with boundary condition $u(1, t) = 0$. The boundary condition is enforced by setting the last row of the differentiation matrix (10) (DM) to zero. Figure 11 shows the eigenvalues of the DM. In the left image of the figure the DM was formed using RLDL0 with $\mu =$ 5e-15. The largest real part of any eigenvalue is 3.2e-2. In the right

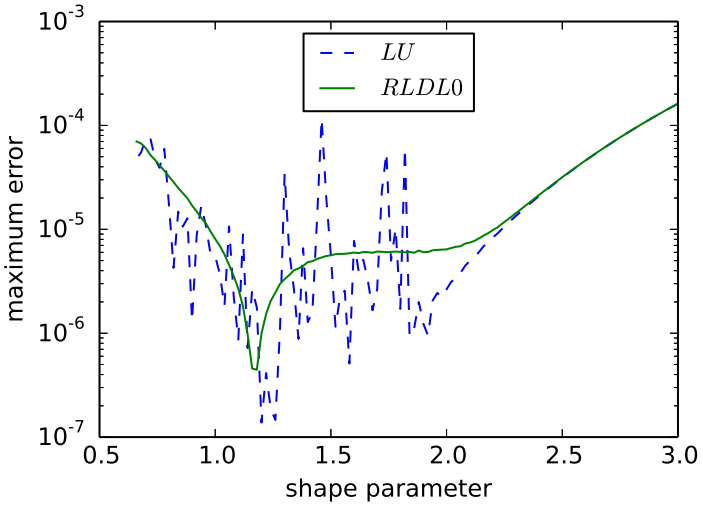

Figure 11: Derivative of function (14) calculated with LU factorization and RLDL0 with μ = 5e-15. The smallest RLDL0 error is 4.45e-7 at $\varepsilon = 1.18$.

image of the figure the DM was formed using LU factorization with partial pivoting. The largest real part of any eigenvalue is 47.2.

Figure 13 plots the accuracy of RBF approximation of the differential operator $\mathcal{L} = \frac{\partial}{\partial x} + \frac{\partial}{\partial y}$ applied to the function (27) over a range of the shape parameter using LU factorization and the RLL1 algorithm with μ = 5e-13 to evaluate the linear systems. The domain and centers are in figure 8. The differentiation matrix is modified to enforce zero Dirichlet boundary conditions $u(0, y, t) = 0$ and $u(x, 0, t) = 0$ for the two dimensional advection equation

$$\frac{\partial u}{\partial t} - \frac{\partial u}{\partial x} - \frac{\partial u}{\partial y} = 0. \tag{29}$$

The boundary conditions are enforced by zeroing rows corresponding to boundary centers.

The right image in figure 9 shows that the system matrix B is severely ill-conditioned at shape parameter $\varepsilon = 1.0$. Figure 14 shows the eigenvalues of the DM for the advection equation (29). In the right image of the figure the DM formed with LU factorization has eigenvalues with large positive real parts. The eigenvalues of the DM formed using RLL1 with μ = 5e-13 are in the left image of the figure. The largest positive real part of the eigenvalues is 1.27e-3.

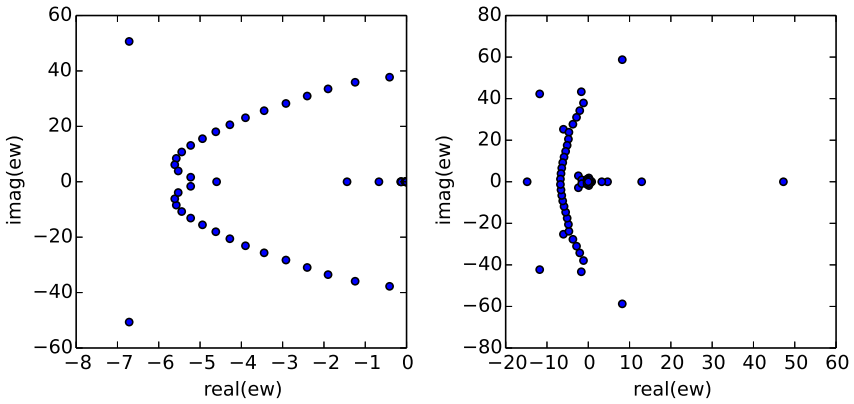

Figure 12: Eigenvalues of the discretized space operator of equation (28). The derivative matrix was formed using two different algorithms. Left: RLDL0 with μ = 5e-15. Right: LU factorization with partial pivoting.

RSPD methods do not necessarily produce DMs with eigenvalues that all have non-positive real parts. However, the positive real parts are tiny when compared to DMs produced with a LU factorization without regularization. The RSPD DMs may be used to accurately integrate time-dependent PDEs over short to moderate time intervals. The RSPD DMs are certainly a better starting point for use with hyper-viscosity methods [9] than are DMs formed using LU factorization via the M-backlash operator as is often done.

6 Conclusions

Many RBFs, such as the IQ that has been used throughout, have a system matrix that is symmetric positive definite. The standard algorithm for factorizing a SPD matrix is the LL factorization. The uncertainty principle that is associated with the RBF-direct approach dictates that the approach is most accurate when the system matrix is highly ill-conditioned. Highly ill-conditioned system matrices associated with accurate RBF methods are likely not NSPD and an attempted LL factorization fails. This either forces the use of a larger shape parameter and overall less accuracy or the use of a computationally more expensive LU factorization. LDL seems like a more appropriate factorization to use with RBF methods, but optimized LDL functions are currently not available in major scientific software packages.

The shape parameter range for which the LL factorization is applicable

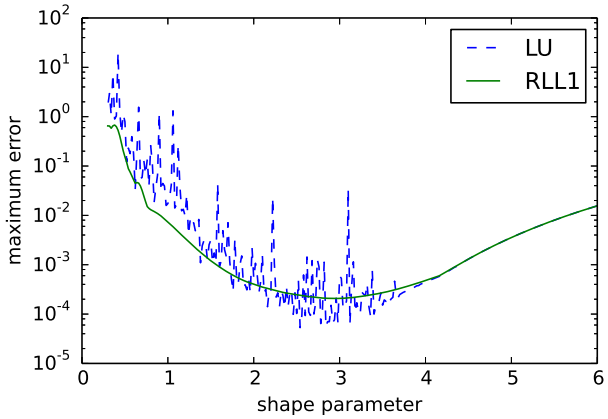

Figure 13: RBF approximation of the differential operator $\mathcal{L} = \frac{\partial}{\partial x} + \frac{\partial}{\partial y}$ applied to the function (27).

can be significantly extended via regularization techniques. The recommended regularization parameter range for two regularization techniques, the method of diagonal increments and Riley's method, has been made as 5e-13 $\leq \mu \leq$ 5e-15. Criteria for stopping the Riley's iteration with the recommended range of regularization parameter have been described. Several implementations of the RSPD methods have been described: RSPD0, RSPD1, and RSPD. RSPD0, or the method of diagonal increments, is particularly simple and effective. The RSPD0 method has essentially the same flop count as a LL factorization but it is more accurate and it is applicable with very ill-condition matrices. The RSPD0 method deserves consideration as the standard method for linear systems arising from RBF methods. If more accuracy is desired than RSPD0 provides, a slight improvement in accuracy is often possible via the RSPD1 or RSPD algorithms at the expense of a slightly higher flop count.

A large amount of research in the RBF field has been done with the M-backslash algorithm as a primary tool. In this case, an ill-conditioned RBF system matrix is factorized with LU factorization that does not consider the symmetry of the matrix and that uses twice as many flops as a LL factorization. The RLL0 algorithm is as many as four times more efficient, as well as more accurate, than is M-backslash when a theoretically SPD matrix that is not NSPD is factorized. Finally, the eigenvalue stability properties of RBF DMs for time-dependent hyperbolic PDEs formed with the RSPD0 or RSPD1 methods are vastly better than the properties of DMs

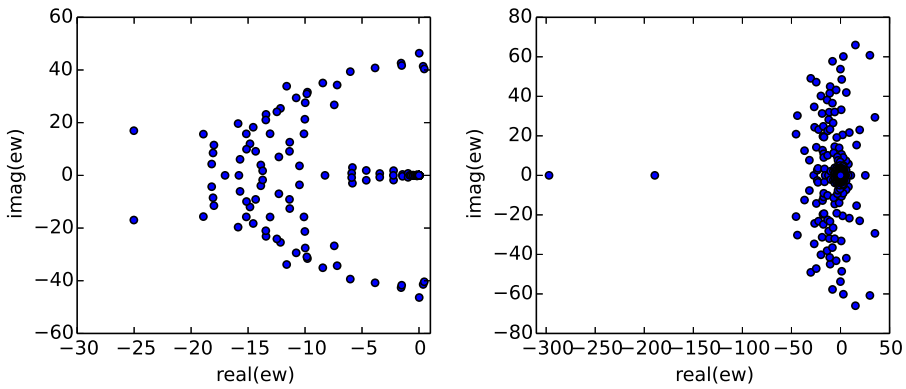

Figure 14: Eigenvalues of the RBF space discretization of equation (29) with $\varepsilon = 1.0$. Left: RLL1 with μ =5e-13. Right: LU factorization.

formed using M-backslash.

The Matlab and Python computer code used to produce the examples is available on the author's web site [36].

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, third edition, 1997. 3

[2] A. Bassi. A Scilab Radial Basis Function toolbox. Master's thesis, University of Padova, 2012. 1, 4, 4.1, 4.1

[3] M. D. Buhmann. *Radial Basis Functions*. Cambridge University Press, 2003. 2

[4] D. A. Cervantes Cabrera, P. Gonzlez-Casanova, C. Gout, L. Hctor Jurez, and L. Rafael Resndiz. Vector field approximation using radial basis functions. *Journal of Computational and Applied Mathematics*, pages 163–173, 2013. 5.3

[5] A. Emdadi, E. J. Kansa, N. Ali Libre, M. Rahimian, and M. Shekarchi. Stable PDE solution methods for large multiquadric shape parameters. *Computer Modeling In Engineering And Sciences*, 25(1):23–42, 2008. 5.3

[6] G. E. Fasshauer. *Meshfree Approximation Methods with Matlab*. World Scientific, 2007. 2, 2

[7] G. E. Fasshauer. Tutorial on meshfree approximation methods with Matlab. *Dolomite Research Notes on Approximation*, 1, 2008. 1, 4, 4.1, 4.1, 5.2

[8] G. E. Fasshauer and M. McCourt. Stable evaluation of Gaussian RBF interpolants. *SIAM Journal of Scientific Computing*, 34(2):737–762, 2012. 1

[9] B. Fornberg and E. Lehto. Stabilization of RBF-generated finite difference methods for convective PDEs. *Journal of Computational Physics*, 230:2270–2285, 2011. 5.4, 5.4

[10] B. Fornberg, E. Lehto, and C. Powell. Stable calculation of gaussian-based rbf-fd stencils. *Computers and Mathematics with Applications*, 65:627–637, 2013. 1

[11] B. Fornberg and C. Piret. A stable algorithm for flat radial basis functions on a sphere. *SIAM Journal on Scientific Computing*, 30:60–80, 2007. 1

[12] B. Fornberg and G. Wright. Stable computation of multiquadric interpolants for all values of the shape parameter. *Computers and Mathematics with applications*, 48:853–867, 2004. 1

[13] B. Fornberg and J. Zuev. The runge phenomenon and spatially variable shape parameters in rbf interpolation. *Computers and Mathematics with Applications*, 54:379–398, 2007. 2

[14] R. Franke. Scattered data interpolation: Tests of some methods. *Mathematics of Computation*, pages 181–200, 1982. 5.1

[15] G. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins Press, third edition, 1996. 3, 5.3

[16] L. Guttman. Enlargement methods for computing the matrix inverse. *The Annals of Mathematical Statistics*, 17(3):336–343, 1946. 4

[17] P. C. Hansen. The truncated SVD as a method for regularization. *BIT*, 27(4):534–553, 1987. 5.3

[18] H. V. Henderson and S. R. Searle. On deriving the inverse of a sum of matrices. *SIAM Review*, 23(2):53–60, 1981. 4

[19] N. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2002. 1, 3

[20] C.-S. Huang, C.-F. Leeb, and A.H.-D. Cheng. Error estimate, optimal shape factor, and high precision computation of multiquadric collocation method. *Engineering Analysis with Boundary Elements*, 31:614–623, 2007. 1

[21] Intel. Math kernel library. http://developer.intel.com/software/products/mkl/. 5.2

[22] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. 1

[23] E. J. Kansa. Multiquadrics - a scattered data approximation scheme with applications to computational fluid dynamics I: Surface approximations and partial derivative estimates. *Computers and Mathematics with Applications*, 19(8/9):127–145, 1990. 2

[24] E. Larsson and E. Hehto. Stable computation of differentiation matrices and scattered node stencils based on gaussian radial basis functions. *SIAM Journal on Scientific Computing*, 35:A2096–A2119, 2013. 1

[25] S. Li and J. P. Boyd. Symmetrizing grids, radial basis functions, and Chebyshev and Zernike polynomials for the d4 symmetry group; interpolation within a squircle, part i. *Journal of Computational Physics*, 258:931–947, 2014. 1

[26] N. Ali Libre, A. Emdadi, E. J. Kansa, M. Rahimian, and M. Shekarchi. A stabilized RBF collocation scheme for Neumann type boundary value problems. *Computer Modeling In Engineering And Sciences*, 24(1):61–80, 2008. 5.3

[27] MATLAB. *version 8.2.0 (R2013b)*. The MathWorks Inc., Natick, Massachusetts. 1

[28] M. McCourt. A stochastic simulation for approximating the log-determinant of a symmetric positive defnite matrix. http://www.thefutureofmath.com/mathed/logdet.pdf. 1, 4, 4.1

[29] W. Piegorsch and G. Casella. The early use of matrix diagonal increments in statistical problems. *SIAM Review*, 31:428–434, 1989. 4

[30] R. Platte. How fast do radial basis function interpolants of analytic functions converge? *IMA Journal of Numerical Analysis*, 31(4):1578–1597, 2010. 4.1

[31] R. Platte and T. Driscoll. Polynomials and potential theory for gaussian radial basis function interpolation. *SIAM Journal of Numerical Analysis*, 43(2):750–766, 2005. 4.1

[32] R. Platte and T. Driscoll. Eigenvalue stability of radial basis functions discretizations for time-dependent problems. *Computers and Mathematics with Applications*, 51:1251–1268, 2006. 5.4

[33] J. D. Riley. Solving systems of linear equations with a positive definite, symmetric, but possibly ill-conditioned matrix. *Mathematical Tables and Other Aids to Computation*, 9(51):96–101, 1955. 4, 4, 4.1

[34] S. A. Sarra. A numerical study of the accuracy and stability of symmetric and asymmetric rbf collocation methods for hyperbolic PDEs. *Numerical Methods for Partial Differential Equations*, 24(2):670 – 686, 2008. 5.4

[35] S. A. Sarra. Radial basis function approximation methods with extended precision floating point arithmetic. *Engineering Analysis with Boundary Elements*, 35(1):68–76, 2011. 1

[36] S. A. Sarra. Computer code that accompanies this manuscript. http://www.scottsarra.org/math/papers/RSPD.zip, 2013. 6

[37] S. A. Sarra and E. J. Kansa. Multiquadric radial basis function approximation methods for the numerical solution of partial differential equations. *Advances in Computational Mechanics*, 2, 2009. 2, 5.3

[38] S. A. Sarra and D. Sturgill. A random variable shape parameter strategy for radial basis function approximation methods. *Engineering Analysis with Boundary Elements*, 33:1239–1245, 2009. 2

[39] R. Schaback. Error estimates and condition numbers for radial basis function interpolation. *Advances in Computational Mathematics*, 3:251–264, 1995. 2

[40] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, first edition, 1997. 3

[41] K. Volokh and O. Vilnay. Pin-pointing solution of ill-conditioned square systems of linear equations. *Applied Mathematics Letters*, 13:119–124, 2000. 5.3

[42] H. Wendland. *Scattered Data Approximation*. Cambridge University Press, 2005. 2, 2, 2