STABLE CALCULATION OF GAUSSIAN-BASED RBF-FD STENCILS

BENGT FORNBERG*, ERIK LEHTO†, AND COLLIN POWELL‡

Abstract. Traditional finite difference (FD) methods are designed to be exact for low degree polynomials. They can be highly effective on Cartesian-type grids, but may fail for unstructured node layouts. Radial basis function-generated finite difference (RBF-FD) methods overcome this problem and, as a result, provide a much improved geometric flexibility. The calculation of RBF-FD weights involves a shape parameter ε. Small values of ε (corresponding to near-flat RBFs) often lead to particularly accurate RBF-FD formulas. However, the most straightforward way to calculate the weights (RBF-Direct) becomes then numerically highly ill-conditioned. In contrast, the present algorithm remains numerically stable all the way into the $\varepsilon \to 0$ limit. Like the RBF-QR algorithm, it uses the idea of finding a numerically well-conditioned basis function set in the same function space as is spanned by the ill-conditioned near-flat original Gaussian RBFs. By exploiting some properties of the incomplete gamma function, it transpires that the change of basis can be achieved without dealing with any infinite expansions. Its strengths and weaknesses compared the Contour-Padé, RBF-RA, and RBF-QR algorithms are discussed.

Key words. RBF, radial basis functions, RBF-FD, RBF-GA, ill-conditioning, Gaussians.

1. Introduction.

The weights in finite difference (FD) stencils are typically determined by requiring that the resulting FD formula becomes exact for polynomials of as high degree as possible [7]. On lattice-based node sets, multi-dimensional FD formulas can then be obtained by combining 1-D formulas. This standard FD approach becomes problematic if multi-dimensional node sets are unstructured, since the linear systems that need to be solved for the weights then frequently become singular or highly ill conditioned. The RBF-FD idea [3, 23, 25, 26] is to require such approximations to be exact for radial basis functions (RBFs) rather than for multivariate polynomials. When using Gaussian RBFs, this procedure can never give rise to singularities, no matter how the nodes are distributed [19, 15, 21]. Results tend to become particularly accurate when using nearly flat RBFs (very small shape parameter ε) [13, 18, 27], but the resulting systems will then again become ill conditioned. However, in contrast to the multivariate polynomial case, the ill-conditioning that arises in the RBF case is not of a fundamental nature, and it can be avoided by using appropriate numerical algorithms. The first RBF algorithm that remained well conditioned all the way into the flat $\varepsilon \to 0$ limit (the Contour-Padé method) [14] was shortly after its discovery applied to the task of creating RBF-FD stencils [27].

The earliest RBF-FD studies described numerical solutions of elliptic and of convective-diffusive PDEs. The approach was soon afterwards shown to be well suited for computational fluid mechanics [1, 22, 24], more recently also in purely convective situations [5, 11]. Most of the studies quoted above provide estimates for the errors that are caused by replacing continuum equations with RBF-FD discretizations. The present study is not concerned with this, but instead with the accurate computation of the weights (coefficients) in RBF-FD formulas.

So far, only two numerical approaches have been presented that remain stable all the way into the $\varepsilon \to 0$ (increasingly flat) RBF limit: the *Contour-Padé/RBF-RA* methods, and the *RBF-QR* method. These two approaches are based on entirely

*University of Colorado, Department of Applied Mathematics, 526 UCB, Boulder, CO 80309, USA (fornberg@colorado.edu).

†Uppsala University, Department of Information Technology, Box 337, 751 05 Uppsala, Sweden (erik.lehto@it.uu.se).

‡Same address as the first author (collin.powell@gmail.com).

different principles. Contour-Padé/RBF-RA methods are limited to a relatively low number of RBF nodes (n slightly less than a hundred in 2-D, more in 3-D) [14, 17, 28]. The RBF-QR algorithm was first developed for the case of nodes distributed over the surface of a sphere, then allowing n-values in the thousands [12]. Generalizations of RBF-QR to d-D domains ($d = 1, 2, 3, \ldots$) has so far been achieved mainly for Gaussian (GA) RBFs [4, 9] (however, see [8] for an extension to Bessel-type RBFs [10]). The RBF-QR algorithm is used in [20] for calculating RBF-FD stencils, and in [2] for exploring RBF-FD approximations to Poisson's equation (showing that the low ε-regime is particularly important in RBF-FD contexts).

The algorithm introduced in the present paper has conceptual similarities to the RBF-QR method, but with the novelty of not making any use of truncated infinite expansions or of any other forms of numerical approximations. We describe it here primarily for the task of calculating explicit RBF-FD formulas in 2-D, but it generalizes readily to other RBF tasks in d-D. We denote it the RBF-GA method since (i) the present implementation is specific to Gaussian RBFs and (ii) the incomplete gamma function plays a critical role in the algorithm.

2. Governing equations for FD and RBF-FD weights. In the case of explicit FD formulas, one wants to find weights w_i to use at node locations $\underline{x}_i$, $i = 1, 2, \ldots, n$ so that one obtains the exact value at a location $\underline{x} = \underline{x}_c$ (where the underline in $\underline{x}$ indicates a vector quantity with d components in d-D) for a linear operator L and some set of test functions $\psi_k(\underline{x})$, $k = 1, 2, \ldots, n$. The following relation needs then to hold

$$\begin{bmatrix} \psi_1(\underline{x}_1) & \psi_1(\underline{x}_2) & \cdots & \psi_1(\underline{x}_n) \\ \psi_2(\underline{x}_1) & \psi_2(\underline{x}_2) & \cdots & \psi_2(\underline{x}_n) \\ \vdots & \vdots & & \vdots \\ \psi_n(\underline{x}_1) & \psi_n(\underline{x}_2) & \cdots & \psi_n(\underline{x}_n) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} L\psi_1(\underline{x})|_{\underline{x}=\underline{x}_c} \\ L\psi_2(\underline{x})|_{\underline{x}=\underline{x}_c} \\ \vdots \\ L\psi_n(\underline{x})|_{\underline{x}=\underline{x}_c} \end{bmatrix}. \tag{2.1}$$

The most common set of 1-D test functions are the monomials $\psi_k(x) = x^{k-1}$, in which case (2.1) becomes a standard Vandermonde system. If $\psi_k(\underline{x})$ instead are d-dimensional RBFs $\phi(\|\underline{x} - \underline{x}_k\|)$, one similarly obtains

$$\begin{bmatrix} \phi(\|\underline{x}_1 - \underline{x}_1\|) & \cdots & \phi(\|\underline{x}_n - \underline{x}_1\|) \\ \vdots & & \vdots \\ \phi(\|\underline{x}_1 - \underline{x}_n\|) & \cdots & \phi(\|\underline{x}_n - \underline{x}_n\|) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} L\phi(\|\underline{x} - \underline{x}_1\|)|_{\underline{x}=\underline{x}_c} \\ \vdots \\ L\phi(\|\underline{x} - \underline{x}_n\|)|_{\underline{x}=\underline{x}_c} \end{bmatrix}. \tag{2.2}$$

We will consider this latter system in 2-D and 3-D, using Gaussian (GA) RBFs, i.e. with the radial function $\phi(r) = e^{-(\varepsilon r)^2}$. As noted above, the parameter ε is denoted the *shape parameter*.

A commonly used variation of (2.2) is to also include some low order monomial basis functions and add matching constraints to the associated RBF expansion coefficients. For example, in this case of including $1, x, y$ in 2-D, (2.2) becomes replaced

by

$$\begin{bmatrix} & & & | & 1 & x_1 & y_1 \\ & A & & | & \vdots & \vdots & \vdots \\ & & & | & 1 & x_n & y_n \\ - & - & - & + & - & - & - \\ 1 & \cdots & 1 & | & & & \\ x_1 & \cdots & x_n & | & & 0 & \\ y_1 & \cdots & y_n & | & & & \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \\ - \\ w_{n+1} \\ w_{n+2} \\ w_{n+3} \end{bmatrix} = \begin{bmatrix} L\phi(\|\underline{x} - \underline{x}_1\|)|_{\underline{x}=\underline{x}_c} \\ \vdots \\ L\phi(\|\underline{x} - \underline{x}_n\|)|_{\underline{x}=\underline{x}_c} \\ - \\ L\ 1\ |_{\underline{x}=\underline{x}_c} \\ L\ x\ |_{\underline{x}=\underline{x}_c} \\ L\ y\ |_{\underline{x}=\underline{x}_c} \end{bmatrix}, \tag{2.3}$$

where components beyond w_n of the solution vector should be ignored. The top left submatrix A is the same matrix as in (2.2). For medium-to-large ε-values, the latter form (2.3) (in particular, including a constant) is usually advantageous. However, since our present focus is on small ε values, we focus in this study on (2.2). Still another possible generalization, mainly relevant for time-independent PDEs, is to employ implicit (Hermite-type) RBF-FD stencils [27].

We note that the A-matrix above is exactly the same as the one that arises in the RBF-Direct approach for finding the interpolant

$$s(\underline{x}, \varepsilon) = \sum_{k=1}^{n} \lambda_i \phi(\|\underline{x} - \underline{x}_i\|) \tag{2.4}$$

to scattered data $\{\underline{x}_i, f_i\}, i = 1, 2, \ldots, n$, in which case the coefficients λ_i can be found as the solution to the linear system

$$A\ \underline{\lambda} = \underline{f}\ . \tag{2.5}$$

A stable algorithm will in this interpolation application arrive at $s(\underline{x}, \varepsilon)$ without the (for small ε) highly ill-conditioned intermediate calculation of the $\underline{\lambda}$-vector.

The equations (2.5) and (2.4) mathematically define the RBF interpolant $s(\underline{x}, \varepsilon)$, which is well conditioned with respect to node locations and the data values at these locations. Similarly (2.2) and (2.3) define a well-conditioned set of RBF-FD weights $\underline{w}$ for any value of ε. As noted above, the challenge is to arrive at these final answers without using any intermediate step that resembles an inversion of the ill-conditioned A-matrix.

3. The extent of the *A*-matrix ill-conditioning. In the 2-D case, with the nodes not located on any kind of lattice, the n eigenvalues of the (positive definite) GA A-matrix were in [16] found to be of the sizes $\{\mathcal{O}(1)\}$, $\{\mathcal{O}(\varepsilon^2), \mathcal{O}(\varepsilon^2)\}$, $\{\mathcal{O}(\varepsilon^4), \mathcal{O}(\varepsilon^4), \mathcal{O}(\varepsilon^4)\}$, $\{\mathcal{O}(\varepsilon^6), \mathcal{O}(\varepsilon^6), \mathcal{O}(\varepsilon^6), \mathcal{O}(\varepsilon^6)\}, \ldots$ etc., until all the n eigenvalues have been accounted for (i.e. the last group might contain fewer than its maximally allowed number of entries). From this sequence follows that the condition number for A grows very rapidly to infinity for decreasing ε:

$$\text{cond}(A) = \mathcal{O}(\varepsilon^{-\ 2\lfloor(\sqrt{8n-7}-1)/2\rfloor}) \tag{3.1}$$

where $\lfloor\cdot\rfloor$ denotes the integer part. For simplicity in describing the algorithm, we focus the present discussion on n-values taken from the sequence $n = \frac{1}{2}k(k+1),\ k = 1, 2, 3, \ldots$, i.e. $n = 1, 3, 6, 10, \ldots$. In these cases, all the eigenvalue groups that are present are complete. For scattered nodes in d-D, the corresponding sequence becomes $n = \binom{d+k-1}{d},\ k = 1, 2, 3, \ldots$. The resulting algorithm will however work as well also for other n-values.

4. Some ideas for overcoming the *A*-matrix ill-conditioning. We first note two approaches that may give some benefits, but which have severe limitations (including leaving the $\varepsilon \to 0$ limit out of reach): (i) Regularization of the A-matrix, and (ii) Use of high precision arithmetic. As an example of the former approach, one may replace regular Gaussian elimination-type factorization of A with an SVD-decomposition of A, and then somehow adjust the smallest singular values. A conceptual problem with this approach is that irretrievable loss of information will occur already when the A-matrix is formed (using standard precision arithmetic). While subsequent 'regularization' can produce smooth solutions, it is not clear what improvement (if any) it offers over the use of fewer nodes or larger ε. High precision arithmetic is typically easy to apply, but tends to be very slow [14]; see also Figure 6.5 below. By means of exact formulas, such as (3.1), it is easy to estimate in advance just how costly high precision arithmetic will become in any special case (as a function of both n and ε).

The algorithm introduced in this study is closely related to the RBF-QR method. It is again based on making a clear distinction between a *basis* and the *space* that it spans. As an example, exactly the same space is spanned by the monomials $\{1, x, x^2, \ldots, x^{100}\}$ and by the Chebyshev polynomials $\{T_0(x), T_1(x), \ldots, T_{100}(x)\}$. Tasks such as interpolation, approximation of derivatives, calculations of weights, etc. will mathematically produce identical results when using the two different sets of basis functions. However, the former set (monomials) will lead to far worse conditioned numerics. Just like monomials, near-flat RBFs form a severely ill conditioned basis. This naturally raises the question if one can find a well conditioned basis within exactly the same space. One would then use the better conditioned basis for the actual numerical work, i.e. replace the RBFs in (2.2) by the better conditioned basis and then instead use (2.1).

QR factorization and Gram-Schmidt orthogonalization are both standard approaches for orthogonalizing a set of basis vectors. However, numerical orthogonalization of a set of nearly dependent vectors will inevitably result in severe numerical cancellations. What allows one to evaluate the Chebyshev basis functions with full accuracy is that there exist closed form expressions for them, obtained by analytic means and not from a direct numerical orthogonalization of the monomials. In the present Gaussian RBF case, we thus look for a way to form a much better conditioned new basis through some process in which the cancellations occur analytically rather than numerically. (The RBF-QR method also relies on a QR factorization, but for an entirely different purpose than orthogonalization.)

5. The RBF-GA algorithm in 2-D. A GA radial function $\phi(r) = e^{-(\varepsilon r)^2}$, centered at the point $\underline{x}_i = (x_i, y_i)$, $i = 1, 2, \ldots, n$, can be written

$$\begin{aligned} \psi_i^{(1)}(\underline{x}) &= e^{-\varepsilon^2((x-x_i)^2+(y-y_i)^2)} \\ &= e^{-\varepsilon^2(x_i^2+y_i^2)} \cdot e^{-\varepsilon^2(x^2+y^2)} \cdot e^{2\varepsilon^2(x\, x_i + y\, y_i)} \end{aligned} \tag{5.1}$$

We stay in exactly the same space if we ignore the scalar multipliers $e^{-\varepsilon^2(x_i^2+y_i^2)}$. We also note that a closed form exact remainder term is available for any truncation of the Taylor expansion of the exponential function. Thus, we can equivalently use as

basis functions

$$\psi_i^{(2)}(\underline{x}) = e^{-\varepsilon^2(x^2+y^2)} \cdot e^{2\varepsilon^2(x\ x_i + y\ y_i)} \tag{5.2}$$
$$= e^{-\varepsilon^2(x^2+y^2)} \cdot \left\{1 + \frac{1}{1!}[2\varepsilon^2(xx_i+yy_i)]^1 + \ldots + \frac{1}{(n-1)!}[2\varepsilon^2(xx_i+yy_i)]^{k-1} + G_k(z)\right\}$$

where $z = 2\varepsilon^2(xx_i + yy_i) = 2\varepsilon^2 \underline{x} \cdot \underline{x}_i$, and

$$G_k(z) = e^z - \sum_{j=0}^{k-1} \frac{z^j}{j!} \tag{5.3}$$
$$= \frac{e^z}{(k-1)!} \int_0^z e^{-t} t^{k-1} dt \tag{5.4}$$
$$= e^z \cdot gammainc(z,k). \tag{5.5}$$

Here $gammainc(z,k)$ is equivalent to MATLAB's built-in function with the same name. This MATLAB routine is computationally fast, and it does not suffer from cancellation of significant digits (as can happen with direct implementations of (5.3)).

If ε is small, the successive terms in the Taylor expansion in (5.2) decrease rapidly in size. The apparent difficulty (or rather, as it will turn out, the opportunity) is that the dominant leading Taylor terms feature very strong linear dependence between different i-values. We are thus ready to form a third set of basis functions $\psi_i(\underline{x}) = \psi_i^{(3)}(x,y,x_i,y_i)$, still staying in exactly the same GA space, by forming suitable linear combinations of the $\psi_i^{(2)}(x,y,x_i,y_i)$ functions. The key idea is to do this in such a way that all the Taylor coefficients in (5.2) will get cancelled out *analytically,* allowing them to simply be omitted rather than cancelled numerically (in which case we would destroy significant digits). Letting "*null*" denote MATLAB's built-in function for finding an orthogonal base for a matrix' null-space, we thus arrive at the third basis function set (still spanning exactly the same finite GA space):

$$[\psi_1(\underline{x})] = e^{-\varepsilon^2(x^2+y^2)} \cdot \frac{1}{\varepsilon^0} [B_0] \left[G_0(2\varepsilon^2 \underline{x} \cdot \underline{x}_1)\right] \tag{5.6}$$

with $B_0 = 1$;

$$\begin{bmatrix}\psi_2(\underline{x}) \\ \psi_3(\underline{x})\end{bmatrix} = e^{-\varepsilon^2(x^2+y^2)} \cdot \frac{1}{\varepsilon^2} \begin{bmatrix} B_1 & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} G_1(2\varepsilon^2 \underline{x} \cdot \underline{x}_1) \\ G_1(2\varepsilon^2 \underline{x} \cdot \underline{x}_2) \\ G_1(2\varepsilon^2 \underline{x} \cdot \underline{x}_3) \end{bmatrix} \tag{5.7}$$

with $B_1 = \left[null\left[\begin{array}{ccc}1 & 1 & 1\end{array}\right]\right]^T$;

$$\begin{bmatrix}\psi_4(\underline{x}) \\ \psi_5(\underline{x}) \\ \psi_6(\underline{x})\end{bmatrix} = e^{-\varepsilon^2(x^2+y^2)} \cdot \frac{1}{\varepsilon^4} \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & B_2 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} G_2(2\varepsilon^2 \underline{x} \cdot \underline{x}_1) \\ G_2(2\varepsilon^2 \underline{x} \cdot \underline{x}_2) \\ G_2(2\varepsilon^2 \underline{x} \cdot \underline{x}_3) \\ G_2(2\varepsilon^2 \underline{x} \cdot \underline{x}_4) \\ G_2(2\varepsilon^2 \underline{x} \cdot \underline{x}_5) \\ G_2(2\varepsilon^2 \underline{x} \cdot \underline{x}_6) \end{bmatrix} \tag{5.8}$$

with $B_2 = \begin{bmatrix} null \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ y_1 & y_2 & y_3 & y_4 & y_5 & y_6 \end{bmatrix} \end{bmatrix}^T$, etc.

The dots in the B-matrices above serve to graphically illustrate the matrix sizes. In

the next stage, we get four more basis functions $\psi_7(\underline{x}), \ldots, \psi_{10}(\underline{x})$ with use of the null-space of

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} \\ y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 & y_9 & y_{10} \\ x_1^2 & x_2^2 & x_3^2 & x_4^2 & x_5^2 & x_6^2 & x_7^2 & x_8^2 & x_9^2 & x_{10}^2 \\ x_1y_1 & x_2y_2 & x_3y_3 & x_4y_4 & x_5y_5 & x_6y_6 & x_7y_7 & x_8y_8 & x_9y_9 & x_{10}y_{10} \\ y_1^2 & y_2^2 & y_3^2 & y_4^2 & y_5^2 & y_6^2 & y_7^2 & y_8^2 & y_9^2 & y_{10}^2 \end{bmatrix}.$$

In stage k of the algorithm in d-D, we compute $\binom{d+k-1}{d-1}$ new basis functions using the $\binom{d+k-1}{d-1} \times \binom{d+k}{d}$ matrix B_k, obtained from the null-space of a $\binom{d+k-1}{d} \times \binom{d+k}{d}$ polynomial matrix. It clearly suffices to form only the last of these rectangular matrices and then apply the null-operator to appropriately shaped top-left aligned submatrices of it in order to obtain all the required B-matrices. In order to gain some speed, the B-matrices were in the present work obtained by QR factorization (rather than the singular value decomposition employed by MATLAB's "*null*" routine).

A slight modification of the algorithm is necessary when a structured node set, such as nodes on a Cartesian grid, is used. In this case, some polynomial terms do not occur in the expansion and should thus not be cancelled. Let P_k denote the bivariate polynomial matrix of order k for all nodes, e.g.

$$P_1 = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \end{bmatrix}, \quad P_2 = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ x_1y_1 & x_2y_2 & \cdots & x_ny_n \\ y_1^2 & y_2^2 & \cdots & y_n^2 \end{bmatrix}, \text{ etc.,} \tag{5.9}$$

with $P_{k,j}$ corresponding to the first j columns of this matrix. The number of new basis functions in each stage is determined by the rank of P_k, which we will denote r_k. In stage k of the algorithm, $r_k - r_{k-1}$ new basis functions are computed using the null space of P_{k-1,r_k} in the same manner as before, and this procedure is repeated until all n basis functions have been found. Note also that the ordering of the nodes must be chosen with some care, such that $\text{rank}(P_{k,r_k}) = r_k$ for all k.

We also need to rapidly and accurately compute $G_k(z)$ and some of its derivatives (up to the highest order present in the operator L that is to be approximated by the RBF-FD stencil). With use of (5.5) and the *gammainc* function, all $G_k(z)$ values are very rapidly available (without any numerical cancellations). This is also the case for all its derivatives, since (5.3) implies

$$\frac{d^p}{dz^p} G_k(z) = G_{\max(0,k-p)}(z), \quad p = 0, 1, 2, \ldots \tag{5.10}$$

Appendix A provides further implementation details about evaluating the partial derivatives of $e^{-\varepsilon^2(x^2+y^2)} \cdot G_k(2\varepsilon^2(xx_i + yy_i))$ that are needed for the right hand side (RHS) of (2.1).

With the construction above, $\psi_1(\underline{x})$, as given by (5.6), will approach the constant 1 as $\varepsilon \to 0$. Next, $\{\psi_2(\underline{x}), \psi_3(\underline{x})\}$, given by (5.7) will approach two linear combinations of the functions x and y; from (5.8) follows that $\{\psi_4(\underline{x}), \psi_5(\underline{x}), \psi_6(\underline{x})\}$ approach independent combinations of $\{x^2, xy, y^2\}$, etc. The small deviations of the $\psi_k(\underline{x})$ functions

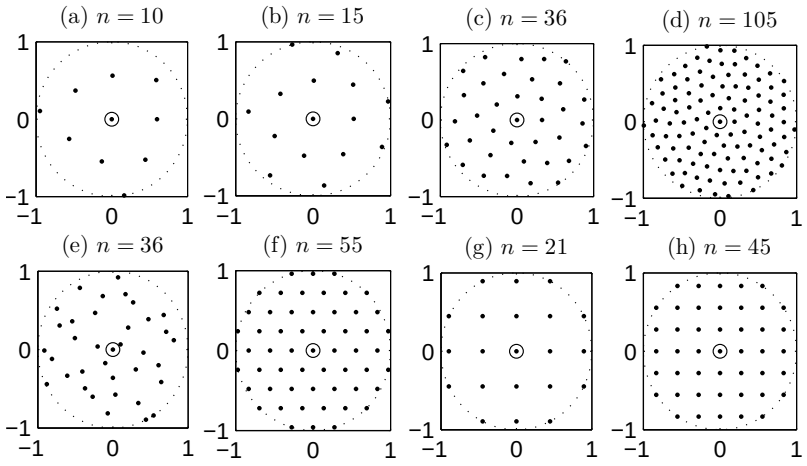


FIG. 5.1. *Node sets within the unit circle used in the present study. A ring marks the central node at which the differential operator is approximated. (a)–(d) show near-uniform nodes, (e) Halton-type, (f) hexagonal and (g)–(h) Cartesian nodes. The near-uniform nodes were obtained by projecting nodes onto a plane from a large near-uniform node set on the sphere.*

from exact polynomial form when ε is not quite zero are a consequence of the fact that we are still remaining in exactly the same approximation space that is spanned by the original GA basis. If we stop the process after having obtained $\psi_1(\underline{x}), \ldots, \psi_{15}(\underline{x})$, we have obtained a base that spans exactly the same space as the first 15 GA RBFs $\psi_i^{(1)}(\underline{x})$, $i = 1, \ldots, 15$, with the major differences that the new basis functions (i) are calculated without any numerical cancellations, and (ii) do not approach linear dependence even in the $\varepsilon \to 0$ limit. Because of the construction using null-spaces, all leading terms in the series expansions in (5.2) have vanished *analytically*. The remainder terms $G_k(z)$ are obtained without any loss of significant digits (even for very small ε). Since the B_k matrices depend only on the node locations, letting ε decrease towards zero cannot have any adverse effect on their calculation either.

5.1. Illustrations of basis functions. In the test calculations described in the present and in subsequent sections, we use in 2-D one or several of the node sets that are illustrated in Figure 5.1. Figure 5.2 shows GA RBFs with $\varepsilon = 10^{-2}$ in the $n = 15$ case. The condition number for the RBF A-matrix becomes (according to (3.1)) of size $O(\varepsilon^{-8}) = O(10^{16})$, making it unclear if RBF-Direct in double precision will suffice for even a single significant digit. Figure 5.3 shows the exactly same test case, but with the basis functions rearranged as described above. The basis functions have now become far more independent.

Reducing ε from 10^{-2} towards zero will cause the condition number for the RBF A-matrix to grow much further still, whereas the new basis functions will change minimally – their low condition number will remain intact. When increasing ε to one, both basis sets (the original and the new one) will be satisfactory. Making ε larger still will cause problems in the new basis set, as the sum in (5.3) then becomes small compared to the exponential term e^z, and the basis function independence will degrade. Figure 6.1(a) illustrates how the conditioning of the two basis sets (the original RBF set, and the new set) vary with ε. Simply choosing the set that has the best conditioning produces accurate results for all values of ε.

One potential concern, suggested by Figure 5.3, might be that the new basis

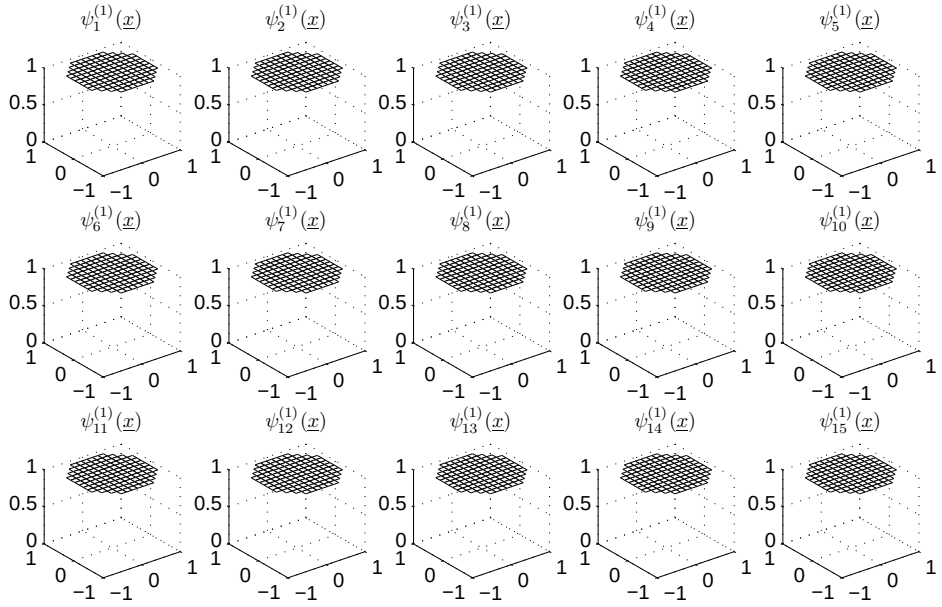


FIG. 5.2. *Original set of basis functions* $\psi_i^{(1)}(\underline{x}) = e^{-(\varepsilon^2||\underline{x}-\underline{x}_i||^2)}$ *in the case of* $\varepsilon = 10^{-2}$.

functions look reminiscent of a constant, two independent linear functions, three independent quadratic functions, etc., quite independently of the locations of the GA RBF centers. The test calculations described next will show that this similarity to increasing order polynomials in fact has a very limited adverse effect in the present RBF-FD context. The reason is that the number of basis functions required for a typical stencil is sufficiently small to avoid conditioning issues with the new basis. Additionally, numerical errors from approximation using high degree polynomials tend to be far larger at domain boundaries than in domain interiors (cf. the Runge phenomenon [6, 16]). In RBF-FD contexts, stencils typically extend in all directions away from a centrally located node, which is the only place at which we need good accuracy.

6. Test calculations. To illustrate the benefits and limitations of the proposed algorithm, some examples in 2-D and 3-D are presented below. Note that only errors in the computed weights are shown. The obtained accuracy when applying a stencil depends on a variety of factors in addition to the accuracy of the weights, such as the function it is applied to, the value of the shape parameter, the stencil size and the node layout. Such considerations however lie beyond the scope of the current work.

6.1. Examples in 2-D. The strengths of the RBF-FD method are most significant when computing weights for unstructured nodes, here exemplified by the near-uniform nodes and the Halton-type nodes shown in Figure 5.1(a)–(e). Nearly uniform nodes are typically preferable for approximating derivatives to high accuracy (in the vicinity of the center of a stencil) and the Halton-type nodes are foremost included to demonstrate the robustness of the RBF-GA algorithm with respect to the node layout.

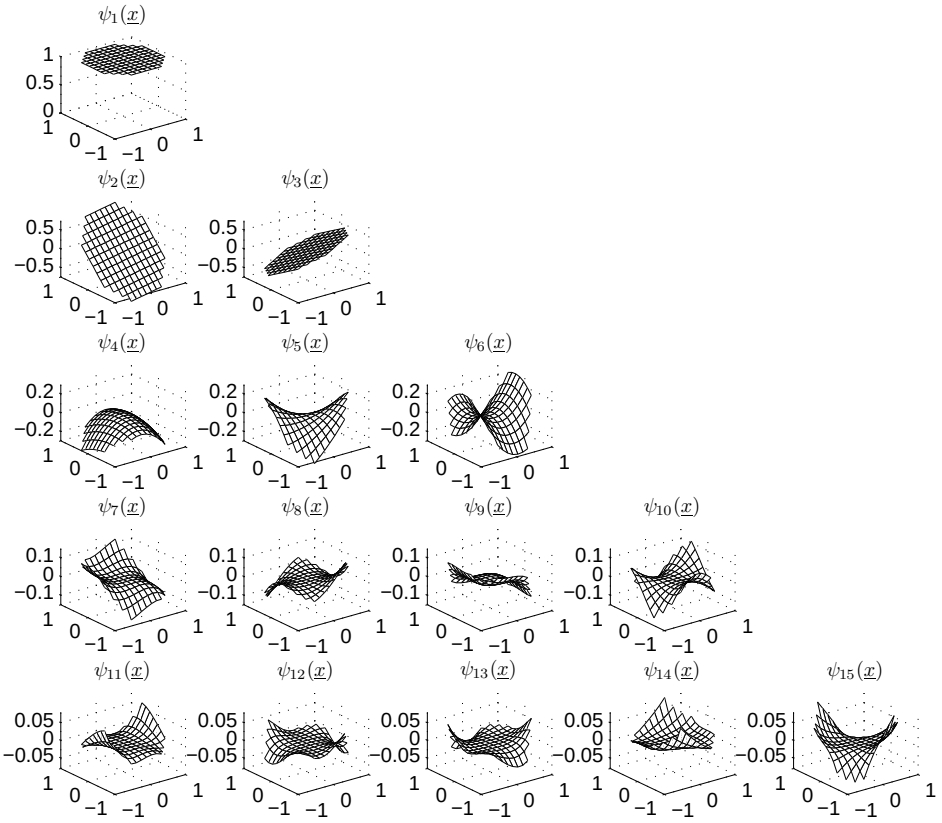
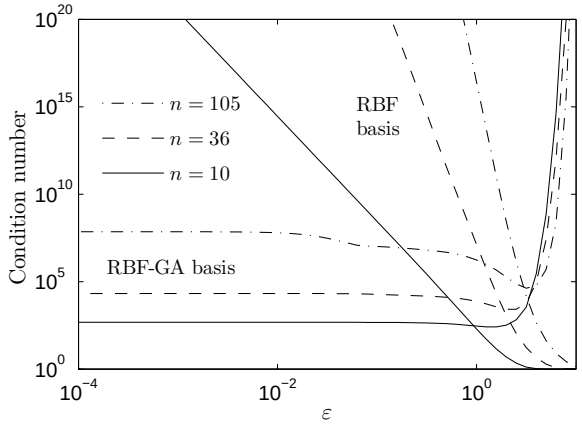


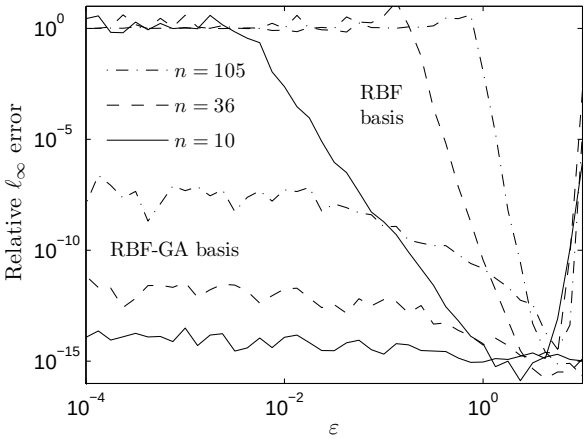
FIG. 5.3. *New set of basis functions $\psi_i(\underline{x})$ created by the RBF-GA algorithm, spanning exactly the same space as the ones shown in Figure 5.2, displayed in rows of subplots corresponding to equations (5.6), (5.7), (5.8), etc. Since ε is small (here $\varepsilon = 10^{-2}$), the functions displayed in row k are very close to k independent homogeneous polynomials of degree $k-1$.*

Figure 6.1(a) shows the condition number of the basis functions for the original basis and the RBF-GA basis with near-uniform nodes. A scaling of the basis functions by e^{2k}, where k is the stage in the RBF-GA algorithm, improves the condition number significantly and, while this scaling has no effect on the accuracy of the computed RBF-FD weights, it provides a better estimate of the loss of precision. The conditioning of the RBF-GA basis shows no dependence on ε for small values of the parameter, unlike the original RBF basis which becomes severely ill-conditioned as $\varepsilon \to 0$. Moving from moderate to large values of ε, the condition number of the RBF-GA basis rapidly increases, while the original RBF basis remains well-conditioned. For small node sets, there is an overlap region where both bases are well-conditioned and either one can be used. As the node set grows, the RBF-GA basis gradually becomes ill-conditioned and still more robust (but computationally slower) algorithms such as RBF-QR [9, 20] should be employed if the RBF-GA algorithm fails to provide the necessary accuracy.

The relative ℓ_∞ error of the computed weights, given by $e_\infty = \frac{\|\underline{w}-\underline{w}_{\text{exact}}\|_\infty}{\|\underline{w}_{\text{exact}}\|_\infty}$, for



(a) Condition number

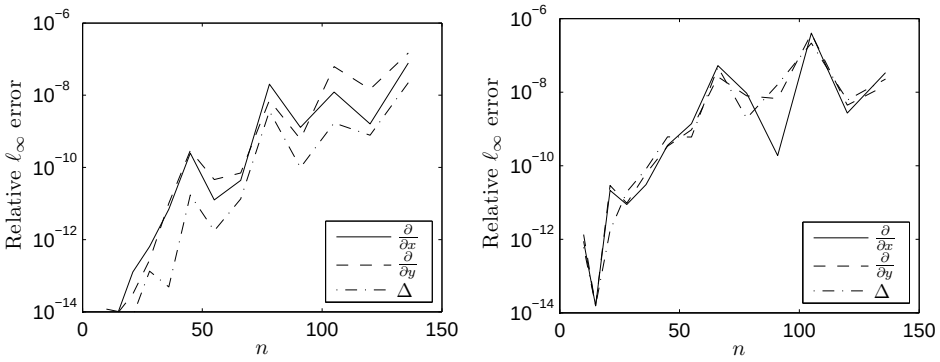


(b) Error in weights

FIG. 6.1. *In (a), the condition number, and in (b), the relative ℓ_∞ error in the computed RBF-FD weights when approximating $\partial/\partial x$, as functions of n and ε on near-uniform nodes in 2-D.*

$\frac{\partial}{\partial x}$ is shown in Figure 6.1(b). The exact weights $\underline{w}_{\text{exact}}$ were obtained by RBF-Direct using extended precision arithmetic. As expected from the conditioning results shown in Figure 6.1(a), accuracy is rapidly lost as ε decreases when using the RBF basis, regardless of stencil size. The error using the well-conditioned RBF-GA basis is on the other hand below 10^{-14} for all ε for the smallest stencil of 10 nodes, and around 10^{-12} and 10^{-8} for 36 and 105 nodes, respectively.

Figure 6.2 shows the error as a function of n at a fixed ε for $\frac{\partial}{\partial x}$, $\frac{\partial}{\partial y}$ and $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$, for both near-uniform and Halton-type nodes. The difference in error between the two node sets is minor, as is the difference in error between the different operators. As the RBF-GA basis depends on the node ordering, different orderings may influence conditioning and accuracy. Randomly permuting the order appears to have little effect on either for unstructured nodes, and any ordering, e.g. by distance from the center, may be used. The growth with n should be expected, since the new basis, while a vast improvement over the original one (cf. Figures 5.2 and 5.3) is still



(a) Near-uniform nodes

(b) Halton nodes

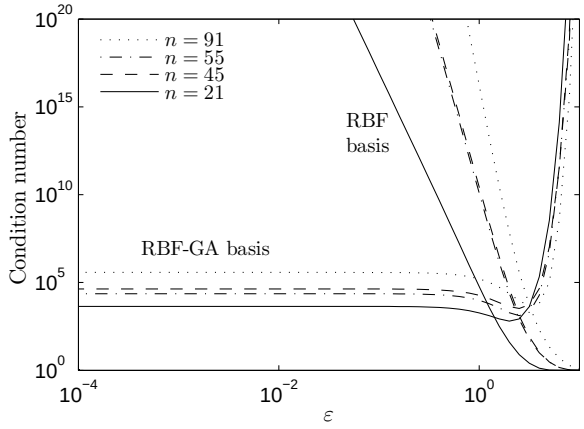
FIG. 6.2. *The relative ℓ_∞ error of the computed RBF-FD weights as a function of n at $\varepsilon = 10^{-4}$, using the RBF-GA algorithm on unstructured nodes. Note from Figure 6.1(b) that RBF-Direct gives $\mathcal{O}(1)$ errors throughout this range of n.*

far from orthogonal.

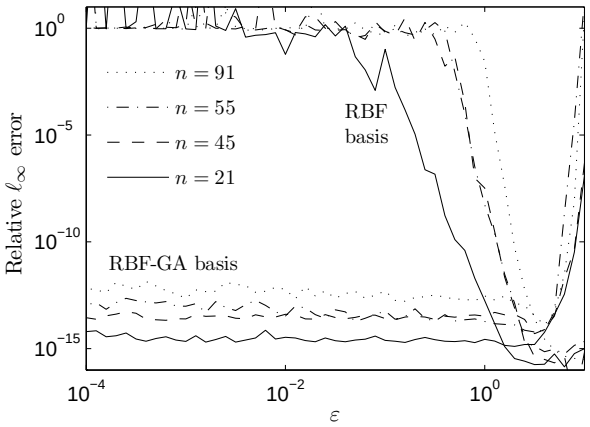
With the small modifications mentioned in Section 5, the RBF-GA algorithm performs well also on structured nodes. For the node sets studied in this example, shown in Figure 5.1(f)–(h), the condition numbers and relative errors are presented in Figures 6.3(a) and 6.3(b). The obtained accuracy is similar to that for unstructured nodes, in fact marginally higher in this example. This is also reflected in the condition numbers, which remain below 10^6 for all values of ε for these four structured node sets.

6.2. Example in 3-D. The behavior of the RBF-GA algorithm for unstructured nodes in 3-D is quite similar to that in 2-D, as apparent from Figures 6.4(a) and 6.4(b). The onset of ill-conditioning with increasing n for the RBF-GA basis is however more gradual and the algorithm is accurate up to at least 500 nodes, which should be sufficient for any application in which the RBF-FD method is of interest.

6.3. Timing example. The measured wall-clock time for computing an RBF-FD stencil in 3-D with a MATLAB implementation of the RBF-GA method, compared to the direct method, the RBF-QR method and the direct method using variable precision arithmetic (VPA) with 100 digit precision, is shown in Figure 6.5 (with only negligible cost savings possible from lowering the VPA precision). For the RBF-GA algorithm, computational cost is generally dominated by the QR factorization to obtain the B_k-matrices, followed by the evaluation of the incomplete gamma function. The runtime for a 2-D problem would be marginally higher for a given n, as more stages are required in this case. Compared to the direct method, RBF-GA is around 10 times slower, while the fastest direct method with VPA is in turn generally 100 times slower than the RBF-GA method. The RBF-QR method lies between the two, typically 2 to 4 times slower than RBF-GA. Note that the performance of the RBF-QR method depends on ε, with optimal performance obtained for small values of the parameter. In this example, the value $\varepsilon = 10^{-4}$ was chosen, which is in or close to the optimal region for all n. The runtime for large n approaches $\mathcal{O}(n^3)$ for all these methods, as expected.



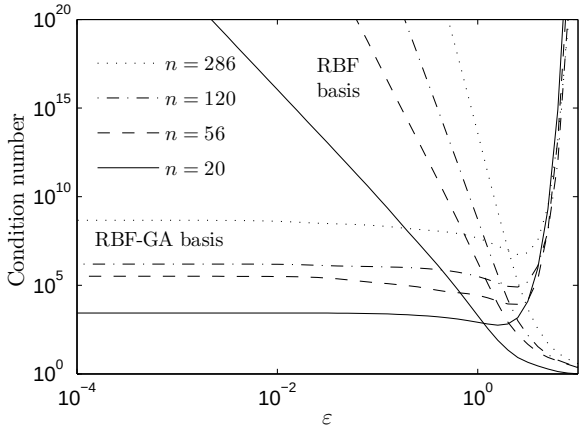
(a) Condition number



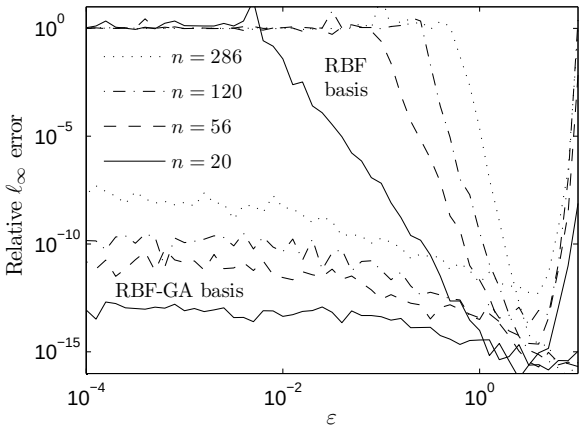
(b) Error in weights

FIG. 6.3. *In (a), the condition number and in (b), the relative ℓ_∞ error in the computed RBF-FD weights when approximating $\partial/\partial x$, as functions of n and ε on structured nodes in 2-D. The two largest node sets are of hexagonal type, and the smaller two are Cartesian node sets.*

7. Conclusions. The present test results show the RBF-GA algorithm to work effectively for up to a few hundred node points in 2-D and up to at least 500 node points in 3-D, which well exceeds the range provided by the Contour-Padé and the RBF-RA algorithms. Indeed, it covers the entire range that is likely to ever be of interest in RBF-FD contexts. Its main disadvantages compared to these two algorithms are (i) that it is strictly limited to GA-type RBFs and (ii) that derivative approximations need to be implemented using the more complex modified basis set rather than the original GA set (especially significant when creating compact stencils, cf. [27]). Compared to the RBF-QR algorithm, as implemented in [20] for generating RBF-FD formulas, it offers great algorithmic simplicity (especially in 3-D) as it involves neither Chebyshev, nor spherical harmonics expansions. This also results in a lower computational cost. On the other hand, when employing the pivoting strategies described in [20], these expansions provide higher accuracy for large node sets and may give RBF-QR a higher degree of robustness in 'degenerate' cases when the nodes



(a) Condition number



(b) Error in weights

FIG. 6.4. *In (a), the condition number and in (b), the relative ℓ_∞ error in the computed RBF-FD weights when approximating $\partial/\partial x$, as functions of n and ε on Halton-type nodes in 3-D.*

are located on low-dimensional manifolds (such as on a small near-flat patch on the surface of a sphere).

Appendix A. Differentiation of the $\psi_k(\underline{x})$ basis functions. The quantities that we need to evaluate partial derivatives for are (in 2-D) of the form

$$\psi(x,y) = e^{-\varepsilon^2(x^2+y^2)} \cdot G(2\varepsilon^2(xx_i + yy_i)) \tag{7.1}$$

where $G(z)$ is a function of one variable, with any number of derivatives immediately available by means of (5.10). The regular product rule allows for easy evaluation of low order partial derivatives of $\psi(x,y)$. Implementing the hyperviscosity approach for stabilizing convection-dominated PDEs [11] requires RBF-FD stencils that approximates powers of the Laplacian operator $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$. These calculations can be simplified by first rewriting (7.1) as the product of two functions of one variable only. This is achieved, separately for each (x_i, y_i), by a coordinate rotation $(x,y) \to (\xi,\eta)$

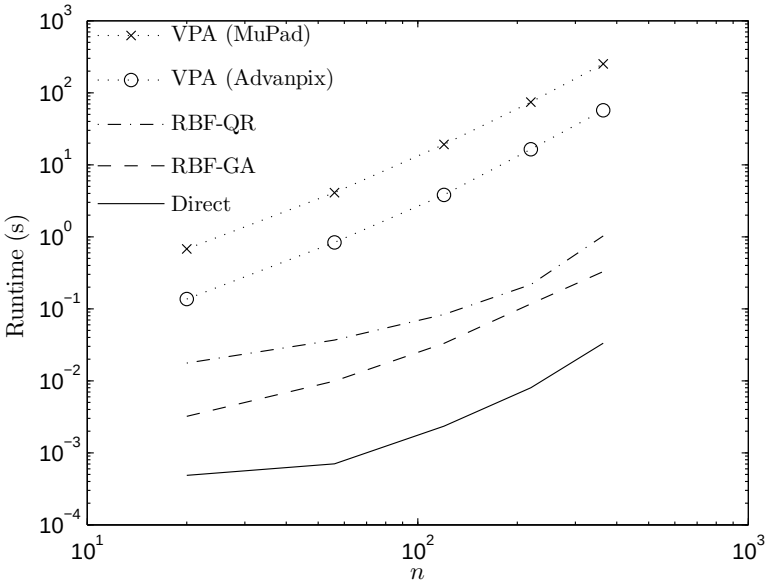


FIG. 6.5. *Measured computer times in Matlab 7.10 as a function of n for five different methods of computing stencil weights in 3-D. The dotted lines correspond to the direct method computed with variable precision arithmetic (VPA) to 100 digits of accuracy using the Symbolic Math Toolbox (marked ×) and Advanpix multi-precision toolbox 3.3.8 (marked ∘). The results were obtained on a PC with 4GB of RAM and an Intel Core2Duo SU7300 processor with two cores running at 1.3GHz.*

such that

$$xx_i + yy_i = r_i\xi, \tag{7.2}$$

where $r_i = \sqrt{x_i^2 + y_i^2}$. Also noting that $x^2 + y^2 = \xi^2 + \eta^2$, we obtain from (7.1)

$$\psi(x(\xi,\eta), y(\xi,\eta)) = \left\{e^{-\varepsilon^2\xi^2} G(2\varepsilon^2 r_i \xi)\right\} \cdot \left\{e^{-\varepsilon^2\eta^2}\right\} = K_1(\xi) \cdot K_2(\eta).$$

Since the coordinate change was a rotation, it leaves the Laplace operator invariant, i.e.

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)^m \psi(x,y) = \left(\frac{\partial^2}{\partial \xi^2} + \frac{\partial^2}{\partial \eta^2}\right)^m K_1(\xi) \cdot K_2(\eta), \quad m = 0, 1, 2, \ldots \tag{7.3}$$

The separation of variables implies

$$\begin{aligned} \Delta^1(K_1 \cdot K_2) &= K_1 K_2^{(2)} + K_1^{(2)} K_2 \\ \Delta^2(K_1 \cdot K_2) &= K_1 K_2^{(4)} + 2K_1^{(2)} K_2^{(2)} + K_1^{(4)} K_2 \\ \Delta^3(K_1 \cdot K_2) &= K_1 K_2^{(6)} + 3K_1^{(2)} K_2^{(4)} + 3K_1^{(4)} K_2^{(2)} + K_1^{(6)} K_2 \\ &\text{etc.,} \end{aligned} \tag{7.4}$$

where the superscripts in the RHS denote derivatives.

It thus only remains to repeatedly differentiate $K_1(\xi) = e^{-\varepsilon^2\xi^2} G(\beta\xi)$ (with $\beta = 2\varepsilon^2 r_i$) and $K_2(\eta) = e^{-\varepsilon^2\eta^2}$. With $H_k(z)$ denoting the regular k^{th} Hermite orthogonal

polynomials, it follows from their relation $\frac{d}{dz}(e^{-z^2}H_k(z)) = -e^{-z^2}H_{k+1}(z)$ that

$$
\begin{aligned}
K_1 &= e^{-\varepsilon^2\xi^2}\cdot\{\quad H_0(\varepsilon\xi)G\} \qquad (7.5)\\
K_1' &= e^{-\varepsilon^2\xi^2}\cdot\{-\varepsilon^1 H_1(\varepsilon\xi)G + \beta H_0(\varepsilon\xi)G'\}\\
K_1'' &= e^{-\varepsilon^2\xi^2}\cdot\{\quad \varepsilon^2 H_2(\varepsilon\xi)G - 2\beta\varepsilon H_1(\varepsilon\xi)G' + \beta^2 H_0(\varepsilon\xi)G''\}\\
K_1''' &= e^{-\varepsilon^2\xi^2}\cdot\{-\varepsilon^3 H_3(\varepsilon\xi)G + 3\beta\varepsilon^2 H_2(\varepsilon\xi)G' - 3\beta^2\varepsilon H_1(\varepsilon\xi)G'' + \beta^3 H_0(\varepsilon\xi)G'''\}\\
&\text{etc.}
\end{aligned}
$$

The pattern seen in (7.5), with coefficients according to Pascal's triangle, continues indefinitely. The case of $K_2(\eta)$ is a special case of (7.5) where the variable is η instead of ξ, and the function G is identically one, i.e. all terms in (7.5) with derivatives of G vanish, leaving in each case only the leading term remaining.

In case of 3-D, one replaces (7.2) by a rotation such that $xx_i + yy_i + zz_i = r_i\xi$, from which follows $\psi = \left\{e^{-\varepsilon^2\xi^2}G(2\varepsilon^2 r_i\xi)\right\}\cdot\left\{e^{-\varepsilon^2\eta^2}\right\}\cdot\left\{e^{-\varepsilon^2\zeta^2}\right\}$, etc.

Acknowledgements. Bengt Fornberg was supported by the NSF Grants DMS-0611681 and DMS-0914647. Erik Lehto received support from the Göran Gustafsson Foundation and the NSF grants ATM-0620100 and DMS-0934317. Helpful discussions with Natasha Flyer, Elisabeth Larsson and Grady Wright are gratefully acknowledged.

REFERENCES

[1] Chinchapatnam, P.P., Djidjeli, K., Nair, P.B. and Tan, M., A compact RBF-FD based meshless method for the incompressible Navier-Stokes equations, Proc. IMechE, Part M- J. Eng. for Maritime Env. 223 (2009), 275–290.

[2] Davydov, O. and Oanh, D.T., On the optimal shape parameter for Gaussian radial basis function finite difference approximation of the Poisson equation, Comput. Math. Appl. 62 (2011), 2143–2161.

[3] Driscoll, T.A. and Fornberg, B., Interpolation in the limit of increasingly flat radial basis functions, Comput. Math. Appl., 43 (2002), 413–422.

[4] Fasshauer, G.E. and McCourt, M.J., Stable evaluation of Gaussian RBF interpolants, SIAM J. Sci. Comput. 34(2) (2012), A737–A762.

[5] Flyer, N., Lehto, E., Blaise, S., Wright, G.B. and St-Cyr, A., A guide to RBF-generated finite differences for nonlinear transport: Shallow water simulations on a sphere, J. Comput. Phys. 231 (2012) 4078–4095.

[6] Fornberg, B., A Practical Guide to Pseudospectral Methods, Cambridge University Press (1995).

[7] Fornberg, B., Calculation of weights in finite difference formulas, SIAM Rev. 40 (3) (1998) 685–691.

[8] Fornberg, B., Larsson, E. and Flyer, N., Stable computations with Gaussian radial basis functions in 2-D, Report No. 2009-020, Dept. of Information Technology, Uppsala Univ., Uppsala, Sweden (2009).

[9] Fornberg, B., Larsson, E. and Flyer, N., Stable computations with Gaussian radial basis functions, SIAM J. Sci. Comp. 33 (2011), 869–892.

[10] Fornberg, B., Larsson, E. and Wright, G., A new class of oscillatory radial basis functions, Comput. Math. Appl., 51 (2006), 1209–1222.

[11] Fornberg, B. and Lehto, E., Stabilization of RBF-generated finite difference methods for convective PDEs, J. Comput. Phys. 230 (2011), 2270–2285.

[12] Fornberg, B. and Piret, C., A stable algorithm for radial basis functions on a sphere, SIAM J. Sci. Comp., 30 (2007), 60–80.

[13] Fornberg, B. and Piret, C., On choosing a radial basis function and a shape parameter when solving a convective PDE on a sphere, J. Comput. Phys. 227 (2008), 2758–2780.

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

[15] Fornberg, B., Wright, G. and Larsson, E., Some observations regarding interpolants in the limit of flat radial basis functions, Comput. Math. Appl., 47 (2004), 37–55.

[16] Fornberg, B. and Zuev, J., The Runge phenomenon and spatially variable shape parameters in RBF interpolation, Comput. Math. Appl., 54 (2007), 379–398.
[17] Gonnet, P., Pachŏn, R. and Trefethen, L.N., Robust rational interpolation and least-squares, Electr. Trans. on Numer. Anal. 33 (2011), 146–147.
[18] Larsson, E. and Fornberg, B., A numerical study of radial basis function based solution methods for elliptic PDEs, Comput. Math. Appl. 46 (2003), 891–902.
[19] Larsson, E. and Fornberg, B., Theoretical and computational aspects of multivariate interpolation with increasingly flat radial basis functions, Comput. Math. Appl. 49 (2005), 103-130.
[20] Larsson, E., Lehto, E., Heryudono, A.R.H. and Fornberg, B., Stable computation of differentiation matrices and scattered node stencils based on Gaussian radial basis functions, to be submitted.
[21] Schaback, R., Multivariate interpolation by polynomials and radial basis functions, Constr. Approx., 21 (2005), 293–317.
[22] Shan, Y. Y.; Shu, C.; Lu, Z. L., Application of Local MQ-DQ Method to Solve 3D Incompressible Viscous Flows with Curved Boundary. Comp. Modeling in Eng. & Sci., .25 (2008), 99–113.
[23] Shu, C., Ding, H. and Yeo, K.S., Local radial basis function-based differential quadrature method and its application to solve twodimensional incompressible Navier–Stokes equations, Comput. Methods Appl. Mech. Eng. 192 (2003) 941–954.
[24] Stevens, D., Power, H., Lees, M. and Morvan, H., The use of PDE centers in the local RBF Hermitean method for 3D convective-diffusion problems, J. Comput. Phys. 228 (2009), 4606–4624.
[25] Tolstykh, A.I., On using RBF-based differencing formulas for unstructured and mixed structured-unstructured grid calculations. In Proceedings of the 16th IMACS World Congress, Lausanne (2000).
[26] Tolstykh, A.I. and Shirobokov, D.A., On using radial basis functions in a "Finite difference mode" with applications to elasticity problems, Comp. Mech. 33 (2003), 68–79.
[27] Wright, G.B. and Fornberg, B., Scattered node compact finite difference-type formulas generated from radial basis functions, J. Comput. Phys., 212 (2006), 99–123.
[28] Wright, G.B. and Fornberg, B., An algorithm for stable computations with flat radial basis functions, to be submitted.