Fast calculation of Laurent expansions for matrix inverses

Bengt Fornberg *
Department of Applied Mathematics
University of Colorado
Boulder, CO 80309, USA

September 8, 2016

Abstract

Previously described algorithms for calculating the Laurent expansion of the inverse of a matrix-valued analytic function become impractical already for singularity orders as low as around $p = 6$, since they require over $O(28^p)$ matrix multiplications and correspondingly large amounts of memory. In place of using mathematically exact recursions, we show that, for floating point calculations, a rational approximation approach can avoid this cost barrier without any significant loss in accuracy.

Keywords: Matrix inverse, Laurent expansion, radial basis functions, RBF.
AMS classification codes: 15A09, 30E10, 65E05, 65F20.

1 Introduction

We consider in this study matrices whose elements are analytic functions of a complex variable z:

$$A(z) = A_0 + z\,A_1 + z^2\,A_2 + \ldots \tag{1}$$

If A_0 is non-singular, equating coefficients in the identity $A(z)A^{-1}(z) = I$ trivially gives a matching Taylor expansion for $A^{-1}(z)$. The case with A_0 singular arises in numerous applications, as summarized in the introductions of [1, 18]. These include Markov chains and singular perturbation problems for ODEs and PDEs. Connections with radial basis functions (RBF) will be discussed below in Section 4.

If $A^{-1}(z)$ exists in some punctured neighborhood of $z = 0$, it will there possess a Laurent expansion:

$$A^{-1}(z) = z^{-p}X_{-p} + z^{-p+1}X_{-p+1} + z^{-p+2}\,X_{-p+2} + \ldots. \tag{2}$$

Given the leading matrices $\{A_0, A_1, \ldots, A_k\}$ in (1), with $k \geq p$, the order p of the singularity as well as the matrices $\{X_{-p}, X_{-p+1}, \ldots, X_{k-2p}\}$ in (2) will become uniquely determined. Several different techniques have been presented for calculating these matrices [1, 2, 18, 19, 21, 25]. They all in some

* *Email*: fornberg@colorado.edu

		RBF	n -values	
p	$g(p)$	1-D	2-D	3-D
1	14	2	2-3	2-4
2	598	3	4-6	5-10
3	18694	4	7-10	11-20
4	544254	5	11-15	21-35
5	15287758	6	16-21	36-56
6	420038854	7	22-28	57-84
7	11368586038	8	29-36	85-120
8	304362660958	9	37-45	121-165

Table 1: The number $g(p)$ of $n \times n$ matrix multiplications required in the exact Laurent expansion method [17, 18]. The three right columns shows how p is related to the matrix size n in the contexts of scattered node RBF approximations in 1-D to 3-D.

sense generalize the approach from the non-singular case. The operation count was shown in [18] to grow exponentially with the value of p, taking the form $O(2n^3 g(p))$ where n is the size of A. The function $g(p)$ was given in closed form, together with its approximation $g(p) \approx 0.6759 \cdot (28.8636)^p$. The data in the second column of Table 1 is taken from this reference. A computer time in excess of an hour is reported for a case with $n = 17$ and $p = 6$. It is not known whether this growth rate with p is optimal for an algorithm that is mathematically exact. For later reference (cf. Section 4), the last three columns of Table 1 show how, in the case of RBF approximations, the value of p is related to n (with the matrix size n then also denoting the number of node points) [15].

We note in Section 2 that an immediate generalization of the scalar contour integration approach for Laurent coefficients may work well also in some matrix cases, and is computationally fast (with an operation count that is independent of p). However, this approach becomes problematic especially when $A(z)$ is singular not just at $z = 0$, but also at some more z-locations close to the origin. We introduce therefore a rational approximation method, which much improves the performance also in these cases. Section 3 starts by extending a previously considered test example, and then describes the present algorithm in more technical detail. Following some general remarks about RBFs in Section 4, a larger RBF-based test case is described, together with some comments on computational costs and on stable RBF algorithms. Section 5, contains some conclusions, and is followed by an Appendix containing a MATLAB code for the algorithm.

2 Introductory discussion of two approximation methods

2.1 The scalar case - contour integration and rational approximation

If an analytic function $f(z)$ has a pole of order p at the origin, it can near to it be represented by a Laurent expansion

$$f(z) = \sum_{k=-p}^{\infty} a_k z^k, \tag{3}$$

where the coefficients are given by

$$a_k = \frac{1}{2\pi i} \oint_C \frac{f(z)}{z^{k+1}} dz, \; k = -p, -p+1, \ldots \tag{4}$$

and the contour C encircles the origin in the positive direction, close enough not to include any further singularity of $f(z)$. The standard numerical discretization of integrals of the form (4) is to sample the integrand at n_z equidistant z-locations around a small circle of some radius r, providing the approximations

$$a_k \approx \frac{1}{n_z \cdot r^k} \sum_{j=0}^{n_z-1} \frac{f(r\,\omega^j)}{\omega^{k\,j}}, \tag{5}$$

where $\omega = e^{2\pi\,i/n_z}$ is the n_z^{th} root of unity. Since the integrand is periodic and analytic, this trapezoidal rule approximation converges exponentially fast with increasing n_z [24]. Considerations for choosing values for n_z and r include:

$$\begin{array}{ll} n_z & \begin{cases} \text{too large} & : \text{ increasing cost} \\ \text{too small} & : \text{ low accuracy in the trapezoidal rule} \end{cases} \\ r & \begin{cases} \text{too large} & : \text{ wrong result if the path includes additional singularities of } f(z) \\ \text{too small} & : \text{ excessive numerical cancellations in evaluating (5)} \end{cases} \end{array}$$

If $f(z)$ has singularities close to the pole at the origin $z = 0$, the two restrictions on r will overlap, leaving little or no suitable range for actual computations. An example of this will be shown in Section 2.3 (Example 1).

The alternate approach that we will generalize to the matrix case does not immediately aim for obtaining the coefficients in (3), but uses as an intermediate step a rational approximation of the form

$$f(z) \approx \frac{c_0 + z\,c_1 + z^2 c_2 + \ldots + z^{n_c-1} c_{n_c-1}}{b_0 + z\,b_1 + z^2 b_2 + \ldots + z^{n_b-1} b_{n_b-1}}, \tag{6}$$

which then in turn can be converted over to the desired power series form (3). If determination of the b's reveal that $b_0 = b_1 = \ldots = b_{p-1} = 0$ but $b_p \neq 0$, the order of the singularity has been found to be p. By including a number of b-coefficients past b_p, the denominator becomes capable of correctly incorporating several additional poles of the function $f(z)$, in case any such would happen to be located inside or on/close to the computational circle (at a priori unknown locations).

2.2 Generalization of the rational approximation method to the matrix case

Starting from a truncated expansion (1) for $A(z)$, which we denote $\tilde{A}(z)$, we evaluate its inverse $\tilde{A}^{-1}(z)$ numerically for n_z different complex z-values, which for example can be chosen as in the scalar contour integration case, equispaced around a circle of radius r. We next seek matrices $C_0, C_1, \ldots, C_{n_c-1}$ and scalar coefficients $b_0, b_1, \ldots, b_{n_b-1}$ to obtain good agreement for all these z-values in a rational approximation of the form

$$\widetilde{A}^{-1}(z) \approx \frac{C_0 + zC_1 + z^2 C_2 + \ldots + z^{n_c-1} C_{n_c-1}}{b_0 + zb_1 + z^2 b_2 + \ldots + z^{n_b-1} b_{n_b-1}} \tag{7}$$

(using throughout this paper the convention that matrices are denoted by upper case letters, and scalars with lower case). A key feature here is that all the n^2 entries of the matrix $\tilde{A}^{-1}(z)$ have their poles at exactly the same complex z-values, given by the zeros of $\det(\tilde{A}(z))$, implying that the denominator in (7) can be scalar valued. Multiplying up the denominator produces a linear system for the $n^2 \cdot n_c + n_b$ unknown coefficients in (7). This system becomes overdetermined when

the number $n^2 \cdot n_z$ of equations exceeds this number of unknowns. This large linear system will turn out to have a special structure that allows for a highly effective numerical solution procedure. Once this solution has been obtained, (7) is rearranged into the form (2). The MATLAB code shown in the Appendix carries out these steps.

The idea behind the present approach has some similarities to the implementation of the unified transform approach for Laplace's equation that is described in [6]. In that case, one also creates an approximate formula that ideally should hold for all values of a complex parameter (there denoted by k, here by z). With use of a sufficient number of different k-values, that led similarly to an overdetermined linear system for the unknown coefficients of the problem.

2.3 Example 1: Introductory test case, with an additional singularity near the origin

The linear matrix function

$$\tilde{A}(z) = \begin{bmatrix} -2.639295 & -2.159624 & -1.439718 \\ 2.089475 & 1.709720 & 1.139790 \\ -1.869505 & -1.529736 & -1.019802 \end{bmatrix} + z \begin{bmatrix} 0.01 & 0 & -0.02 \\ -0.08 & 0.03 & 0.02 \\ 0 & -0.01 & -0.02 \end{bmatrix} \tag{8}$$

(with all entries to be considered as exact numbers and not rounded approximations) has a corresponding Laurent expansion for which any number of coefficient matrices can be found in closed form. The expansion starts

$$\tilde{A}^{-1}(z) = \frac{1}{z} \begin{bmatrix} -960 & 210 & 1590 \\ -3840 & 840 & 6360 \\ 7520 & -1645 & -12455 \end{bmatrix} + \begin{bmatrix} 20998880 & -2999920 & -32998320 \\ 76996220 & -10999730 & -120994330 \\ -153992160 & 21999440 & 241988240 \end{bmatrix} + O(z). \tag{9}$$

Apart from at $z = 0$, $\tilde{A}(z)$ is singular also at $z = -0.0001$ and at $z = -1$.

Mathematica's exact recursions, when applied to (8) and set to use standard double precision, return a completely flawed expansion starting with $\tilde{A}^{-1}(z) = X_0 + z\,X_1$ where the matrices X_0 and X_1 have entries of sizes $O(10^{+11})$ and $O(10^{+18})$, respectively. Direct contour integration will require $r < 10^{-4}$ in order to not to have any additional singularity inside the computational circle. For such a small r-value, ill-conditioning when evaluating $\tilde{A}(z)^{-1}$ prevents (5) from returning accurate results, especially for higher expansion coefficients, cf. Figure 1 (c). In (5), errors in $f(r\,\omega^j)$ will be severely scaled up when the sum is divided by r^k. Subplots (d) and (e) of Figure 1 show much more 'robust' results for the present rational approximation approach, especially with regard to higher Laurent (matrix) coefficients (larger k-values), as well as a vastly increased flexibility in terms of placing the nodes z_i where $\tilde{A}(z)^{-1}$ is sampled. There is no longer any need to keep the nodes inside the nearest singularity, nor to place them in any special pattern (such as equispaced around a circle).

3 Implementation of the rational approximation method

We start this section by giving a somewhat larger example around which we then describe the rational approximation algorithm in more detail.

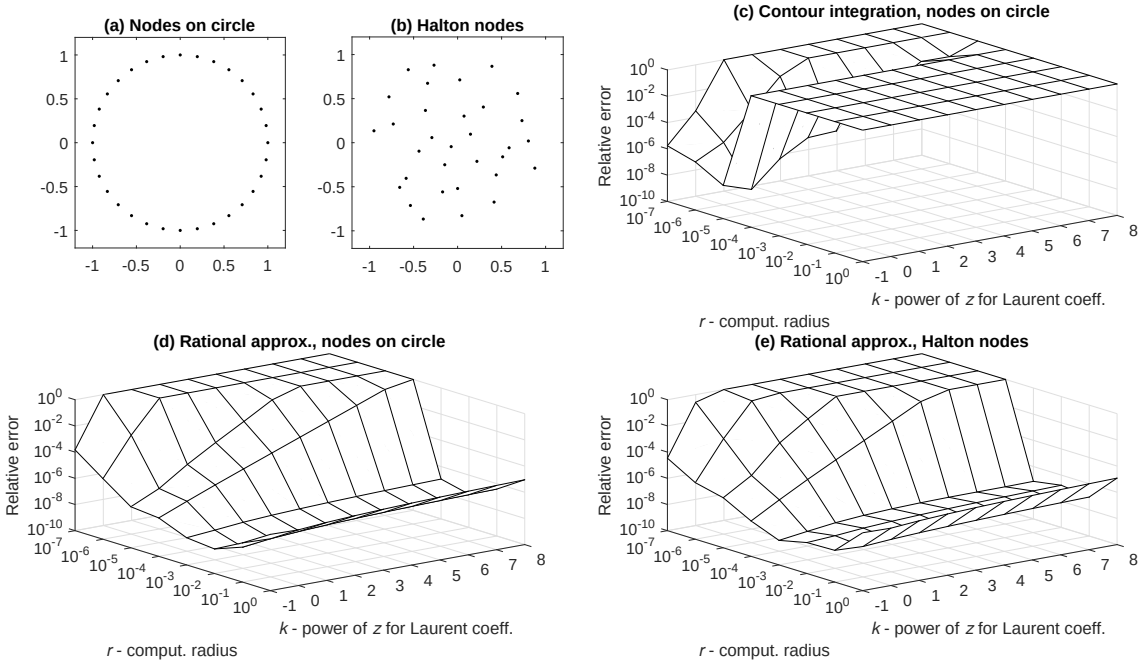

Figure 1: (a) $n_z = 32$ nodes equispaced around the unit circle; (b) $n_z = 32$ nodes quasi-randomly (Halton) scattered within the unit circle; (c) The largest relative errors in the entries of the computed matrices X_k for $k = -1, 0, 1, \ldots, 8$ as functions of the radial scaling r of the node locations shown in (a), when using the contour integration approach; (d) Same as for (c), but instead using the present rational function approach; (e) Results for the rational approximation approach when scaling with different radii r the node set (b) instead of the node set (a). The error surfaces in (c), (d) and (e) are truncated upwards at the level of Relative error = 1.

3.1 Example 2

A smaller $n = 3$ sized counterpart to the present $n = 5$ example below was previously considered in [17, 18]. Let $A(z)_{i,j} = \sqrt{1 + z(i-j)^2}$ for $1 \le i, j \le 5$, i.e.

$$A(z) = \begin{bmatrix} 1 & \sqrt{1+z} & \sqrt{1+4z} & \sqrt{1+9z} & \sqrt{1+16z} \\ \sqrt{1+z} & 1 & \sqrt{1+z} & \sqrt{1+4z} & \sqrt{1+9z} \\ \sqrt{1+4z} & \sqrt{1+z} & 1 & \sqrt{1+z} & \sqrt{1+4z} \\ \sqrt{1+9z} & \sqrt{1+4z} & \sqrt{1+z} & 1 & \sqrt{1+z} \\ \sqrt{1+16z} & \sqrt{1+9z} & \sqrt{1+4z} & \sqrt{1+z} & 1 \end{bmatrix}. \tag{10}$$

The Taylor expansion for $A(z)$ up through degree 5 becomes

$$\tilde{A}(z) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} + z \begin{bmatrix} 0 & \frac{1}{2} & 2 & \frac{9}{2} & 8 \\ \frac{1}{2} & 0 & \frac{1}{2} & 2 & \frac{9}{2} \\ 2 & \frac{1}{2} & 0 & \frac{1}{2} & 2 \\ \frac{9}{2} & 2 & \frac{1}{2} & 0 & \frac{1}{2} \\ 8 & \frac{9}{2} & 2 & \frac{1}{2} & 0 \end{bmatrix} + \ldots + z^5 \begin{bmatrix} 0 & \frac{7}{256} & 28 & \frac{413343}{256} & 28672 \\ \frac{7}{256} & 0 & \frac{7}{256} & 28 & \frac{413343}{256} \\ 28 & \frac{7}{256} & 0 & \frac{7}{256} & 28 \\ \frac{413343}{256} & 28 & \frac{7}{256} & 0 & \frac{7}{256} \\ 28672 & \frac{413343}{256} & 28 & \frac{7}{256} & 0 \end{bmatrix}. \tag{11}$$

This truncated expansion contains all the information that is needed to determine that the order of the singularity is $p = 4$, and then to obtain the first two terms in the corresponding Laurent expansion:

$$A(z)^{-1} \approx \frac{1}{z^4} \begin{bmatrix} -\frac{1}{1008} & \frac{1}{252} & -\frac{1}{168} & \frac{1}{252} & -\frac{1}{1008} \\ \frac{1}{252} & -\frac{1}{63} & \frac{1}{42} & -\frac{1}{63} & \frac{1}{252} \\ -\frac{1}{168} & \frac{1}{42} & -\frac{1}{28} & \frac{1}{42} & -\frac{1}{168} \\ \frac{1}{252} & -\frac{1}{63} & \frac{1}{42} & -\frac{1}{63} & \frac{1}{252} \\ -\frac{1}{1008} & \frac{1}{252} & -\frac{1}{168} & \frac{1}{252} & -\frac{1}{1008} \end{bmatrix} + \frac{1}{z^3} \begin{bmatrix} -\frac{55}{1764} & \frac{349}{3528} & -\frac{143}{1176} & \frac{251}{3528} & -\frac{61}{3528} \\ \frac{349}{3528} & -\frac{565}{1764} & \frac{61}{147} & -\frac{467}{1764} & \frac{251}{3528} \\ -\frac{143}{1176} & \frac{61}{147} & -\frac{115}{196} & \frac{61}{147} & -\frac{143}{1176} \\ \frac{251}{3528} & -\frac{467}{1764} & \frac{61}{147} & -\frac{565}{1764} & \frac{349}{3528} \\ -\frac{61}{3528} & \frac{251}{3528} & -\frac{143}{1176} & \frac{349}{3528} & -\frac{55}{1764} \end{bmatrix}. \tag{12}$$

For each further term that is provided in (11), another term becomes available in (12). While symbolic algebra packages can find expansions for orders up to around $p = 5$, rapidly increasing computer times for larger p-values is not the only problem. For example, Mathematica (versions 9 and 10) generate the two-term expansion (12) by the statements

$$\text{A=Table}[\sqrt{1 + \text{z(i-j)}^2}, \{\text{i}, 1, 5\}, \{\text{j}, 1, 5\}];\ \text{Inverse}[\text{Series}[\text{A}, \{\text{z}, 0, 11\}]]$$

but gives less or no terms if the statement $\text{Series}[\text{A}, \{\text{z}, 0, 11\}]$ is set to produce a shorter expansion for $A(z)$ than all the way up through $O(z^{11})$. As noted above, $O(z^5)$ should have sufficed.

3.2 More detailed description of the rational approximation algorithm

3.2.1 Main steps in the algorithm / code

In order to describe the steps in the algorithm as distinctly as possible and to simplify further explorations with it, MATLAB codes for this Example 2 are included in the Appendix. The code is split in three parts:

(i) Main script - only sets parameter values, and then calls the main Laurent expansion function L_exp.

(ii) Function A_z that evaluates the truncated Taylor expansion $\tilde{A}(z)$ of the $A(z)$ matrix.

(iii) Function L_exp that evaluates the matrix Laurent expansion.

While codes for (i) and (ii) are straightforward, the code (iii) is next described in some detail. In this description, lists of the form $\langle \ldots \rangle$ identify the corresponding line numbers in code. Following some guidelines that will be discussed in Section 3.2.3, the values for the five parameters r, n_b, n_c, n_z and bc (with $n_b < n_c < n_z$) are entered in the main script (code lines $\langle 5-9 \rangle$).

After multiplying up the denominator in (7) and moving all terms to the left hand side, the equation takes the form

$$\{C_0 + zC_1 + \ldots + z^{n_c-1}C_{n_c-1}\} - \tilde{A}(z)^{-1}\{b_0 + zb_1 + \ldots + z^{n_b-1}b_{n_b-1}\} = 0. \tag{13}$$

We place n_z complex points z_i, $i = 1, 2, \ldots, n_z$ equispaced around a circle of radius r surrounding the origin in the z-plane $\langle 32, 33 \rangle$. Each instance of z_i, when substituted into (13), produces n^2 relations (one relation for each matrix element in the equation). These altogether $n_z \cdot n^2$ relations are then organized as follows: First we write down all n_z relations associated with the (1,1) entries, then all with the (2,1) entries, etc. We order the unknowns as follows: First the (1,1)-entries of all the n_c different C-matrices, followed by the (2,1)-entries of these, etc. and conclude with the n_b different b-entries. If, by chance, $\widetilde{A}(z)$ would be singular (or nearly so) at some z_i location(s), possible ill effects are circumvented by a scaling of the equations $\langle 39 \rangle$. The resulting overdetermined linear system will then take the form shown in Figure 2 (a). The right hand side is the zero vector. All the blocks marked E are identical $\langle 34, 39 \rangle$, whereas the F-blocks become different from each other $\langle 43 - 48 \rangle$.

The solution to an overdetermined linear system remains unaffected if we multiply from the left with a unitary matrix. Hence, we split $E = Q \cdot R$ $\langle 42 \rangle$, and multiply from the left each block row with $Q^* \langle 46 \rangle$, leading to the matrix structure in Figure 2 (b). The order of the equations (matrix rows) can now be changed to instead give the matrix structure shown in part (c) $\langle 49, 50 \rangle$. At this point, we apply a svd factorization to the bottom rectangular block, obtaining matrices $[U, \Sigma, V^*]$ $\langle 51 \rangle$. The last column of V will contain the entries of the desired **b**-vector (as explained further in Section 3.2.2). The number of leading zeros of this vector gives the value of p, i.e. the order of the singularity. In the present example, the $n_b = 10$ entries of the **b**-vector become

$$\begin{array}{rl}
0.000000000000000 & +0.000000000000000i \\
-0.000000000000000 & -0.000000000000000i \\
-0.000000000000000 & -0.000000000000000i \\
0.000000000000000 & -0.000000000000000i \\
-0.000000165840613 & +0.000000010652139i \\
0.000003540753129 & -0.000000160016167i \\
-0.000062884217882 & +0.000000198148194i \\
0.003599142817789 & -0.000143329089913i \\
-0.067868253092651 & +0.003257462751109i \\
0.995956955021008 & -0.058651978351578i
\end{array} \tag{14}$$

clearly showing that $p = 4$ $\langle 53 \rangle$. A slight accuracy gain is achieved by setting these $p = 4$ leading entries to exactly zero and then re-calculate the **b**-vector $\langle 55 \rangle$. The remaining part of the least squares system is then solved by back substitution (using the same R-matrix for all block rows) $\langle 56 \rangle$. With this, numerical values have been obtained for all the C-matrices. Since the denominator

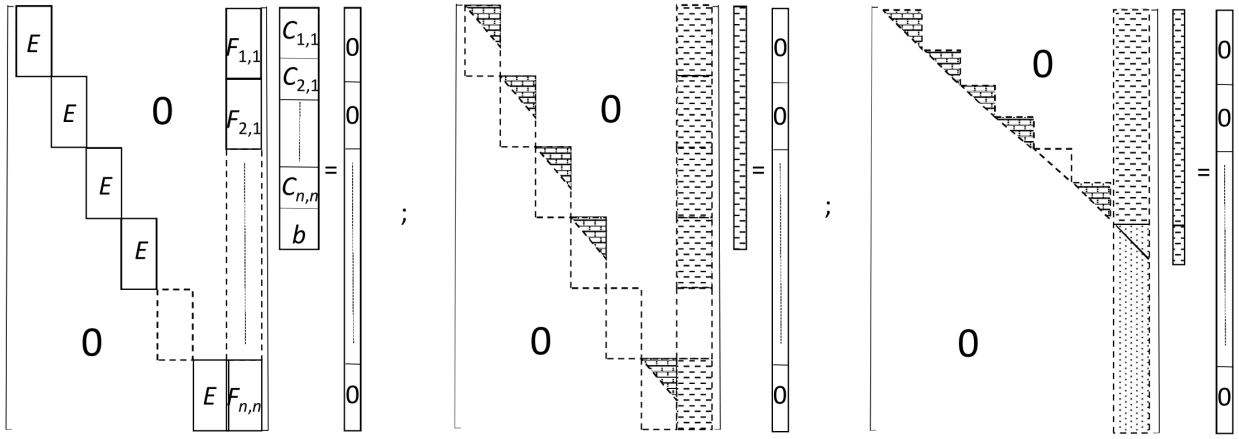

Figure 2: a-c. Structure of the overdetermined linear system during the three stages of the solution process.

of (7) is a scalar polynomial, conversion to the form (2) amounts only to another back substitution $\langle 57, 58 \rangle$. As a last step, X-matrices beyond the ones that are determined by the initial truncated Taylor expansion for $A(\delta)$ are eliminated $\langle 60 \rangle$. It can be noted that the common denominator for all the matrix elements make the emergence of *Froissart doublets* [16, 20] very unlikely.

3.2.2 Determination of the b-vector

The steps from the matrix structure in Figure 2 (a) to that in parts (b) and (c) are all based on the fact that least squares solutions are unaffected when a system is multiplied from the left by unitary matrices, and also when equations are re-ordered. The motivation for how **b** is obtained in the code (on line $\langle 52 \rangle$) needs some additional motivation. It is determined from the last set of equations in Figure 2 (c), illustrated again in the left part of equation (15):

$$\begin{bmatrix} \diagdown & - & - \\ - & \diagdown & - \\ - & - & \diagdown \\ - & B & - \\ - & - & - \\ - & - & - \\ - & - & - \end{bmatrix} \begin{bmatrix} \\ \mathbf{b} \\ \\ \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \\ \vdots \\ \\ 0 \end{bmatrix} \quad ; \quad \underbrace{\begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_{nb} \end{bmatrix}}_{\Sigma} \begin{bmatrix} \\ V^* \\ \\ \end{bmatrix} \begin{bmatrix} \\ \mathbf{b} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \underline{0} \\ \\ \end{bmatrix}. \qquad (15)$$

We split next the rectangular matrix B by a svd decomposition into $U\Sigma V^*$, after which the task becomes to solve the right equation in (15). Up through this point, the least squares systems have been unusual (but convenient for computing) in having a zero RHS. We now note that the $\underline{b}$-vector is undetermined with respect to a scalar multiplier (since multiplying all the C's and b's by the same constant cancels out in the ratio in (7)). One way to ensure a non-zero **b**-vector would be to set $b_0 = 1$. That would here be unsuitable, since a number of leading b-coefficients are expected to be zero. We therefore instead normalize so that $|| \mathbf{b} ||_2 = 1$. Then $V^*\mathbf{b}$ is also a unit length vector. Given that the singular values σ_i are decreasing, we get the best solution when $V^*\mathbf{b} = [0, 0, \ldots, 0, 1]^T$, which tells that **b** should be chosen as the last column of V.

3.2.3 Strategies for choosing the five parameters

While the choices for the five parameters r, n_b, n_c, n_z, and bc could be automated, we have not done so here, in order to improve the simplicity, clarity, and compactness of the code. Regarding the first four parameters, simply varying their values and monitoring the effect this has on the resulting X-matrices provide excellent insight in the achieved accuracy.

In this particular case of Example 2, we can compare with the exact result (12). Figure 3 shows how the largest error in any entry of X_{-4} and X_{-3} varies when one parameter at a time is altered from the setting $r = 0.03$, $n_b = 10$, $n_c = 32$, $n_z = 36$. No delicate 'optimal choice' is needed for any of the four parameters - they all have wide ranges throughout which the performance is roughly equally good. With the double precision machine rounding level at about 10^{-16}, around five places are seen to be lost, giving X_{-4} and X_{-3} with errors around 10^{-11}.

In typical cases, no exact results are available to compare against. However, multiplying (1) with the computed version of (2) gives almost equally good accuracy information, since the product ideally should be the identity matrix, with all other coefficients vanishing. In the case of Example 2 (as specified by (11)), the given information suffices only for the two terms in (12). Numerically, we obtain $\frac{1}{z^4}[O(10^{-16})] + \frac{1}{z^3}[O(10^{-14})]$, where $[O(10^{-d})]$ denotes a matrix with all elements less than 10^{-d} in magnitude. The accuracy loss using this measure is about 2 digits relative to machine rounding level of around $O(10^{-16})$.

Extending this Example 2 test problem to include four more terms in (11) produces four more terms in (12). Leaving the parameters r, n_b, n_c, n_z unchanged produces now a product with the resulting matrices of the sizes

$$\frac{1}{z^4}[O(10^{-15})] + \frac{1}{z^3}[O(10^{-14}) + \frac{1}{z^2}[O(10^{-12})] + \frac{1}{z}[O(10^{-11})] + [I + O(10^{-10})] + z\,[O(10^{-8})]] \quad (16)$$

Given the problem's genuine ill-conditioning (cf. Section 3.3 below), the loss of 8 digits is a remarkably good result. When full 16 digit machine accuracy is required, and especially for still more challenging test examples (with higher p-values), such losses of significant digits are most easily compensated for by using extended precision arithmetic; cf. Section 4.1 below.

The fifth parameter bc is used to separate leading zero entries in the $\underline{b}$-vector from non-zero ones. In the case of (14), any value in the range of 10^{-7} to 10^{-15} would have worked. The function L_exp returns the full $\underline{b}$-vector (in the variable bv), in order to make it easy to spot if, for some reason, an adjustment of the default value of 10^{-12} would be needed.

3.3 Mathematical conditioning of the Laurent expansion problem

In order to interpret the error results above, for ex. as given by (16), we need to note that the Laurent expansion task itself is a mathematically severely ill-conditioned problem. For ex. if the entries of A_0 are perturbed by random infinitesimal amounts, the matrix is likely to become non-singular, and the inverse will feature a $p = 0$ Taylor expansion instead of a $p = 4$ Laurent expansion. Likewise, infinitesimal perturbations of A_1 etc. will significantly affect the end result. Given this, it is somewhat surprising that (14) so clearly here identified $p = 4$.

Mathematica's Series expansion algorithm is based on (computationally costly) exact mathematical recursions. When these are applied to the present test case using double precision arithmetic

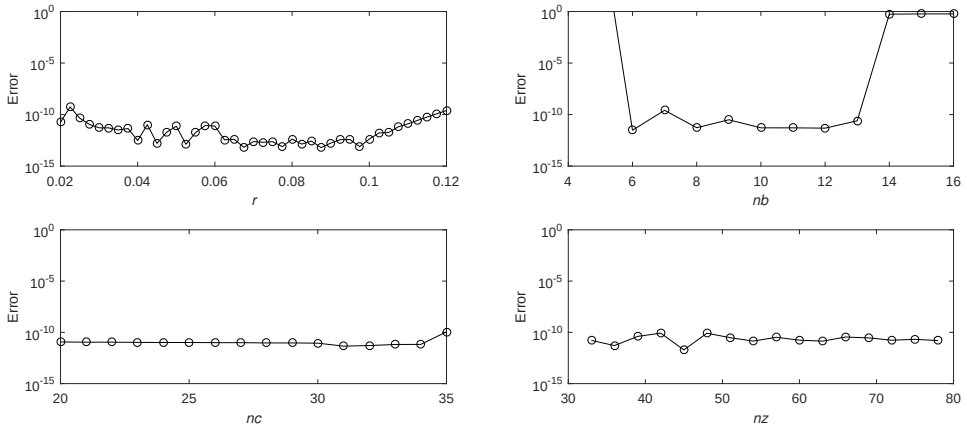

Figure 3: Largest error in any entry of X_{-4} or X_{-3} in Example 2 when one parameter at a time is varied away from the choice $\{r = 0.03,\ n_b = 10,\ n_c = 32,\ n_z = 36\}$.

(instead of employing its option for exact rational arithmetic), we obtain in place of (16)

$$\frac{1}{z^4}[O(10^{-15})] + \frac{1}{z^3}[O(10^{-15}) + \frac{1}{z^2}[O(10^{-14})] + \frac{1}{z}[O(10^{-13})] + [I + O(10^{-11})] + z\,[O(10^{-9})]] \quad (17)$$

The improvement over (16) is very minor, indicating that the accuracy of the present algorithm comes close to what the problem's intrinsic ill-conditioning permits.

4 Matrices arising in the context of RBF approximations

The idea of approximating scattered data by a linear combination of radially symmetric functions (RBFs) goes back at least to around 1970 (for recent surveys, see for example [4, 7, 8]). Table 2 lists four common choices of radial functions $\phi(r)$. An RBF centered at a location $\mathbf{x}_k$ (in a space of any dimensionality) will take the form $\phi(||\mathbf{x} - \mathbf{x}_k||)$, where the norm is the standard Euclidean distance function. With the data value f_k at node $\mathbf{x}_k,\ k = 1, 2, \ldots, n$, a 'basic' RBF interpolant to $f(\mathbf{x})$ then becomes

$$s(\mathbf{x}) = \sum_{k=1}^{n} \lambda_k \phi(||\mathbf{x} - \mathbf{x}_k||) \quad (18)$$

(with different enhancements available, as discussed in the references above). The expansion coefficients λ_k in (18) can be obtained by c*ollocation* - enforcing the exact result at the node points:

$$\begin{bmatrix} \phi(||\mathbf{x}_1 - \mathbf{x}_1||) & \phi(||\mathbf{x}_1 - \mathbf{x}_2||) & \cdots & \phi(||\mathbf{x}_1 - \mathbf{x}_n||) \\ \phi(||\mathbf{x}_2 - \mathbf{x}_1||) & \phi(||\mathbf{x}_2 - \mathbf{x}_2||) & \cdots & \phi(||\mathbf{x}_2 - \mathbf{x}_n||) \\ \vdots & \vdots & & \vdots \\ \phi(||\mathbf{x}_n - \mathbf{x}_1||) & \phi(||\mathbf{x}_n - \mathbf{x}_2||) & \cdots & \phi(||\mathbf{x}_n - \mathbf{x}_n||) \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}, \quad (19)$$

or briefer, $A\boldsymbol{\lambda} = \mathbf{f}$. It has been shown repeatedly that particularly high accuracy usually is obtained when ε (the shape parameter appearing in $\phi(r)$, cf. Table 2) is small, making that parameter regime of near-flat RBFs particularly interesting. After the variable change $\varepsilon^2 = z$, we recognize the matrix (10) in Example 2 as corresponding to 1-D MQ RBF interpolation at the five node points

Type of basis function	**Radial function $\phi(r)$**
Gaussian (GA)	$e^{-(\varepsilon r)^2}$
Multiquadric (MQ)	$\sqrt{1+(\varepsilon r)^2}$
Inverse Quadratic (IQ)	$1/(1+(\varepsilon r)^2)$
Inverse Multiquadric (IMQ)	$1/\sqrt{1+(\varepsilon r)^2}$

Table 2: For common choices for infinitely differentiable radial functions.

$x_k = \{-2, -1, 0, 1, 2\}$. Using n nodes in 1-D, the order of the singularity of $A(z)^{-1}$ becomes $p = n - 1$, agreeing with $p = 4$ observed in Example 2. In higher-D, the corresponding relations between n and p become more complicated, and will depend on the character of the node layouts, with grid-based nodes often giving higher orders (p-values) that scattered node cases [15, 26]. For scattered nodes in 2-D and 3-D, see the last two columns in Table 1. RBF applications more often than not lead to matrices with singularity orders well beyond the previous upper limit for practical Laurent expansions.

Situations with high p-values in conjunction with additional matrix singularities very near to $z = 0$ (as in Example 1) occur frequently. For example, when solving PDEs over surfaces, RBF-generated finite differences (RBF-FD) approximate derivatives using local 'patches' of nodes [5, 10, 23]. If such a 'patch' is near-flat (a 3-D case) vs. completely flat (a 2-D case), Table 1 shows that the p-values will differ by some integer. Just as for complex roots of polynomials with continuously changing coefficients, the present singularities need to move continuously between these two cases, i.e. there must in the transition be singularities arbitrarily close to but not quite at a high-order one at $z = 0$. Near-flat cases arise not only when a curved surface has a near-flat region, but also when solving PDEs over, say, a spherical surface, and node densities are increased while keeping the number of nodes in each RBF-FD stencil fixed.

4.1 Example 3: A larger RBF-based test example

As a concluding test case, we consider the 2-D node distribution illustrated in Figure 4, again using MQ RBFs. The singularity of the $n = 37$ size $A(z)^{-1}$ matrix is in this case of order $p = 8$, placing this test case outside the reach of previous algorithms. We start this time with an expansion for $A(z)$ up through $O(z^{12})$ and will thus be able to compute X_{-8}, $X_{-7}, \ldots, X_{-4}$ (recalling that for the RBF error analysis described in [17, 21], the two leading X-matrices X_{-8}, X_{-7} suffice). The error test indicated in Section 3.2.3 (cf. equation (16)) will in this case show that virtually all significant digits have been lost, if using double precision. Therefore, we run this test case in quad precision (128 bit floating point; about 34 significant decimal digits), using version 3.8.9 of the MATLAB extended precision toolbox developed by Advanpix.

Straightforward tests (as described in Section 3.2.3) will reveal that $r = 0.005$, $n_b = 16$, $n_c = 40$, $n_z = 70$ represent good parameter choices. Similarly to how Figure 3 suggested error levels better than 10^{-11} for all entries in that case (observing by how much the results varied when parameters were changed), we find in the present case (without known exact solution) variations less than 10^{-16} in all entries of X_{-8}, $X_{-7}, \ldots, X_{-4}$ telling that, when converted to double precision, all entries will be accurate to the machine rounding level.

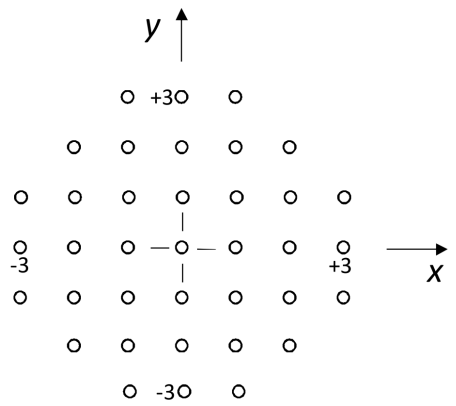

Figure 4: The grid-based $n = 37$ node 2-D stencil used in Example 2.

4.2 Some remarks on computer times

The arithmetic operation count for the algorithm is dominated by the $O(n_b \cdot {n_z}^2 \cdot n^2)$ operations needed for changing the system structure from that shown in Figure 2 part (a) to that in part (b). The value of p enters only indirectly in that n_b may need to be increased in proportion to p (with the need for adjustments in n_z less likely). In regular (hardware supported) double precision, computer times for Examples 2 and 3 become about 0.07 and 0.56 seconds, respectively, on a notebook with a 2 core 2.8 GHz Intel i5 processor. The latter case requires however extended precision for acceptable accuracy, which increases the cost by a factor around 200, to about 2 minutes. However, once the cost penalty of invoking software implemented extended precision is accounted for, it turns out to make only relatively little further difference if one uses quad precision (as for Example 3) or even several hundred digits of precision. Some larger computer systems in the past (such as IBM System 370) had quad precision implemented in hardware, but that is unfortunately rare nowadays.

4.3 Some remarks on stable RBF algorithms

Although both Examples 2 and 3 were motivated by RBF interpolation, it should be noted that Laurent expansion algorithms are quite different from 'stable algorithms' in the sense this phrase has been used to previously describe the Contour-Padé [13] and RBF-QR [9, 12] algorithms for interpolation / derivative approximations and the RBF-QR [22] and RBF-GA [11] algorithms for calculating weights in RBF-FD approximations. Focusing on the former application (interpolation), the key point is that (19) (inverting the A-matrix) by itself is an ill-conditioned problem, whereas calculating the interpolant $s(\mathbf{x})$, mathematically defined by (18) and (19), can be shown to (usually) depend in a well conditioned way on the data $f_k,\ k = 1, 2, \ldots, n$ even in the flat basis function limit of $\varepsilon^2 = z \to 0$ [3, 13, 14]. Designed to bypass a direct use of (19), these stable RBF algorithms arrive at $s(\mathbf{x})$ using intermediate steps that are all well conditioned. They are also computationally much faster than any present Laurent expansion algorithm; see for ex. Figure 8 in [11] for timing information.

5 Conclusions

Previous algorithms for calculating Laurent expansions of matrix valued analytic functions have used exact recursions, and have consequently been severely limited in terms of the orders of the singularities that could be handled - roughly up to $p = 5$ or $p = 6$, due to their need for over $O(28^p)$ matrix multiplications. The present rational approximation-based approach overcomes this growth in cost with p. It has however not yet been developed to the point of offering a completely robust 'black box' code that can be used blindly. The goal of the present study has been limited to demonstrating that this approximate algorithm can overcome this cost barrier and still be close to optimally accurate for a given floating point precision level. However, no code based on floating point calculations can overcome the fact that numerical inversion of near-singular matrices is a genuinely ill-conditioned problem. For increasing problem sizes, it seems therefore very likely that extended precision arithmetic is unavoidable.

In a forthcoming joint study with Grady Wright [27], we apply the present rational approximation approach directly to RBF interpolants $s(\mathbf{x})$ (as defined in (18)) and to RBF-FD weight calculations rather than, as here, to $A(z)^{-1}$, obtaining a new stable RBF algorithm.

6 Appendix

The following is a listing of the MATLAB code for Example 2:

```
warning('off','all'); clear;
% Parameters for test Example 2, used when calling the L_exp routine
% @A_z      % External function evaluating a truncated matrix expansion
m  = 5;     % Use Taylor coefficients up through power m
r  = 0.03;  % Radius in complex z-plane
nb = 10;    % b-coefficients indices 0,1,2,...,nb-1
nc = 32;    % C-matrices     indices 0,1,2,...,nc-1
nz = 36;    % Number of z_i-values around circle radius r; use nb < nc < nd
bc = 1e-12; % Cutoff value in the b-vector, for deciding the order p
[X,b] = L_exp(@A_z,m,r,nb,nc,nz,bc) % Call Laurent expansion routine and
                                    % display its results, and the b-vector



function A = A_z(z,m)
% Function which returns a matrix A, obtained when the Taylor expansion of
% the A(z) matrix has been evaluated at z, using powers up through z^m.
% Implementation here for the Example 2 test case
persistent cf D
if isempty(cf)                  % Execute three lines below once only
   cf = [1,cumprod(cumsum([1/2,-ones(1,m-1)]))./cumprod(1:m)];
   x = -2:2;   n = length(x);  % The 1-D nodes used for Example 2
   D = bsxfun(@minus,x,x').^2; % 'Node distance table' (squared entries)
end
A = ones(size(D));  DD = z*D;
for k = 1:m                     % Add up the Taylor expansion that
   A = A+cf(k+1)*DD;            % provides the A(z) matrix
   DD = z*D.*DD;
end



function [X,bv] = L_exp(A_z,m,r,nb,nc,nz,bc)
```

```
% Input parameters described in main script
% Outpt parameters
%   X   Computed Laurent expansion matrices
%  bv   b-vector

ang = linspace(0,2*pi,nz+1);     % Create z_i-values equispaced around a
z_i  = r*exp(1i*ang(1:nz));      % circle of radius r centered at the origin
E = bsxfun(@power,z_i.',0:nc-1);% Create the E-matrix

for i = 1:nz                     % Loop over z_i-values
   A_i = inv(A_z(z_i(i),m));     % Calculate inverses of the A(z_i) matrices
   if i==1; [n,~] = size(A_i); AI  = zeros(n,n,nz); end
   AI(:,:,i) = A_i;
   E(i,:) = E(i,:)/max(max(abs(AI(:,:,i))));  % Scale the equations
end
AI = permute(AI,[3,1,2]);    % Re-arrange so first index runs over the z_i's
[Q,R] = qr(E);               % qr factorize E
F = zeros(nz,nb,n,n);
for j = 1:n                  % Loop over all the block rows in Fig. 2 (a,b);
   for i = 1:n               % then build up FU and FL (cf. Fig. 2 (c))
      F(:,:,i,j) = Q'*bsxfun(@times,AI(:,i,j),E(:,1:nb));
   end
end
FU = reshape(permute(F(1:nc,    :,:,:),[1,3,4,2]),n^2*nc,      nb);
FL = reshape(permute(F(nc+1:nz,:,:,:),[1,3,4,2]),n^2*(nz-nc),nb);

[~,~,V] = svd(FL,'econ');    % Apply svd to FL; The last column of V will
bv = V(:,nb);                % contain the b-vector. Then find value of p
p = find(abs(bv)>bc,1,'first')-1;
FU = FU(:,p+1:nb); FL = FL(:,p+1:nb);
[~,~,V] = svd(FL,'econ'); b = V(:,nb-p);% Calculate an updated b-vector

C = R(1:nc,:)\reshape(FU*b,nc,n^2);      % Create the C-matrices (eq. (7))
b2 = [b;zeros(nc-nb+p,1)];               % Convert to X-matrices (eq. (2))
X = toeplitz(b2,[b2(1),zeros(1,nc-1)])\C;
X = reshape(X,nc,n,n);  X = permute(X,[2,3,1]);
X(:,:,m-p+2:nc) = [];                    % Display the computed X-matrices
```

References

[1] K. E. Avrachenkov, M. Haviv, and P. G. Howlett, *Inversion of analytic matrix functions that are singular at the origin*, SIAM J. Matrix Anal. **22(4)** (2001), 1175–1189.

[2] K. E. Avrachenkov and J. B. Lasserre, *Analytic perturbation of generalized inverses*, Linear Alg. Applic. **438** (2013), 1793–1813.

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

[4] G. E. Fasshauer, *Meshfree Approximation Methods with MATLAB*, Interdisciplinary Mathematical Sciences - Vol. 6, World Scientific Publishers, Singapore, 2007.

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

[6] B. Fornberg and N. Flyer, *A numerical implementation of Fokas boundary integral approach: Laplace's equation on a polygonal domain*, Proc. R. Soc. A. **467** (2011), 2983–3003.

[7] ———, *A Primer on Radial Basis Functions with Applications to the Geosciences*, SIAM, Philadelphia, 2015.

[8] ———, *Solving PDEs with radial basis functions*, Acta Numerica **24** (2015), 215–258.

[9] B. Fornberg, E. Larsson, and N. Flyer, *Stable computations with Gaussian radial basis functions*, SIAM J. Sci. Comput. **33(2)** (2011), 869–892.

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

[11] B. Fornberg, E. Lehto, and C. Powell, *Stable calculation of Gaussian-based RBF-FD stencils*, Comp. Math. Applic. **65** (2013), 627–637.

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

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

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

[15] B. Fornberg and J. Zuev, *The Runge phenomenon and spatially variable shape parameters in RBF interpolation*, Comput. Math. Appl. **54** (2007), 379–398.

[16] J. Gilewicz and Y. Kryakin, *Froissart doublets in Padé approximation in the case of polynomial noise*, J. Comput. Appl. Math. **153** (2003), 235–242.

[17] P. Gonzalez-Rodriguez, V. Bayona, M. Moscoso, and M. Kindelan, *Laurent series based RBF-FD method to avoid ill-conditioning*, Eng. Anal. Bound. Elem. **52** (2015), 24–31.

[18] P. Gonzalez-Rodriguez, M. Moscoso, and M. Kindelan, *Laurent expansion of the inverse of perturbed, singular matrices*, J. Comput. Phys. **299** (2015), 307–319.

[19] P. G. Howlett, *Input retrieval in finite dimensional linear systems*, J. Austral. Math. Soc. **23** (1982), 357–382.

[20] O. L. Ibryaeva and V. M. Adukov, *An algorithm for computing a Padé approximant with minimal degree denominator*, J. Comput. Appl. Math. **237** (2013), 529–541.

[21] M. Kindelan, M. Moscoso, and P. Gonzalez-Rodriguez, *Radial basis function interpolation in the limit of increasingly flat basis functions*, J. Comput. Phys. **307** (2016), 225–242.

[22] E. Larsson, E. Lehto, A. Heryudono, and B. Fornberg, *Stable computation of differentiation matrices and scattered node stencils based on Gaussian radial basis functions*, SIAM J. Sci. Comput. **35** (2013), A2096–A2119.

[23] V. Shankar, G. B. Wright, R. M. Kirby, and A. L. Fogelson, *A radial basis function (RBF)-finite difference (FD) method for diffusion and reaction-diffusion equations on surfaces*, J. Sci. Comput. **63** (2015), 745–768.

[24] L. N. Trefethen and J. A. C. Weideman, *The exponentially convergent trapezoidal rule*, SIAM Rev. **56(3)** (2014), 385–458.

[25] M. I. Vishik and L. A. Lyusternik, *The solution of some perturbation problems in the case of matrices and self-adjoint differential equations*, Uspechi Mat. Nauk **15** (1960), 3–80.

[26] A. J. Wathen and S. Zhu, *On spectral distribution of kernel matrices related to radial basis functions*, Numer. Alg. **70** (2015), 709–726.

[27] G. B. Wright and B. Fornberg, *Stable computations with flat radial basis functions using vector-valued rational approximations*, submitted (2016).