A SYMMETRIC COLLOCATION METHOD WITH FAST EVALUATION

Michael J. Johnson

Department of Mathematics and Computer Science
Kuwait University
P.O. Box: 5969 Safat 13060 Kuwait
johnson@mcs.sci.kuniv.edu.kw

In memory of Georg Heinig

Abstract. Symmetric collocation, which can be used to numerically solve linear partial differential equations, is a natural generalization of the well-established scattered data interpolation method known as radial basis function (rbf) interpolation. As with rbf interpolation, a major shortcoming of symmetric collocation is the high cost, in terms of floating point operations, of evaluating the obtained function. When solving a linear partial differential equation, one usually has some freedom in choosing the collocation points. We explain how this freedom can be exploited to allow the fast evaluation of the obtained function provided the basic function is chosen as a tensor product of compactly supported piecewise polynomials. Our proposed fast evaluation method, which is exact in exact arithmetic, is initially designed and analyzed in the univariate case. The multivariate case is then reduced, recursively, to multiple univariate evaluations. Along with the theoretical development of the method, we report the results of selected numerical experiments which help to clarify expectations.

1. Introduction

A well-known result from linear algebra (see [7, p. 344]) states that if $\nu_1^r, \nu_2^r, \dots, \nu_n^r$ are linearly independent elements of a real inner-product space H and f is a given element of H, then there exists a unique element $s \in H$ which has minimal norm subject to the conditions

$$\langle s, \nu_i^r \rangle = \langle f, \nu_i^r \rangle, \quad 1 \le i \le n. \tag{1.1}$$

Typeset by $\mathcal{AMS}$-TEX

This result is well liked because s can be easily found as the unique function of the form $s = \lambda_1\nu_1^r + \lambda_2\nu_2^r + \cdots + \lambda_n\nu_n^r$ which satisfies conditions (1.1). In matrix form, these conditions can be expressed as $A\lambda = F$, where $\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_n]^t$, $F = [\langle f, \nu_1^r\rangle, \langle f, \nu_2^r\rangle, \ldots, \langle f, \nu_n^r\rangle]^t$ and A is the $n \times n$ symmetric positive definite Grammian matrix $A(i,j) = \langle \nu_i^r, \nu_j^r\rangle$. If we think of ν_i as the linear functional on H defined by $\nu_i(g) := \langle g, \nu_i^r\rangle$, then conditions (1.1) are *interpolation* conditions and consequently s is the *minimal norm interpolant* to f at the linear functionals $\nu_1, \nu_2, \ldots, \nu_n$.

We wish to employ this result in a rather specific context. Let d be a positive integer and let ϕ be an even, real-valued function in $L_1(\mathbb{R}^d)$ satisfying

$$\widehat{\phi}(w) > 0 \text{ for all } w \in \mathbb{R}^d, \tag{1.2}$$

where $\widehat{\phi}$ denotes the Fourier transform of ϕ defined by $\widehat{\phi}(w) := \int_{\mathbb{R}^d} \phi(x)e^{-\imath x\cdot w}\,dx$. Given our *basic function* ϕ, we let H_ϕ denote the subspace of real-valued functions in $L_2(\mathbb{R}^d)$ for which

$$\|f\|_\phi^2 := (2\pi)^{-d}\int_{\mathbb{R}^d} \left|\widehat{f}(w)\right|^2 / \widehat{\phi}(w)\,dw < \infty.$$

It can be easily shown that H_ϕ is a Hilbert space with inner product

$$\langle f, g\rangle_\phi := (2\pi)^{-d}\int_{\mathbb{R}^d} \widehat{f}(w)\overline{\widehat{g}(w)}/\widehat{\phi}(w)\,dw;$$

however, we make no use of the completeness of H_ϕ; for our purpose it suffices that H_ϕ be an inner product space.

Definition 1.3. A compactly supported real distribution ν is *admissable* if

$$\int_{\mathbb{R}^d} |\widehat{\nu}(w)|^2\,\widehat{\phi}(w)\,dw < \infty. \tag{1.4}$$

We show in section 2 that if ν is admissible, then $\nu^r := \nu * \phi$ belongs to H_ϕ and represents ν in the sense that $\nu(f) = \langle f, \nu^r\rangle_\phi$ for all $f \in H_\phi$. In this context, the minimal norm interpolation result becomes the following:

Minimal Norm Interpolation Theorem. *Let $\nu_1, \nu_2, \dots, \nu_n$ be admissible distributions and let $f \in H_\phi$. Then there exists a unique $s \in H_\phi$ which minimizes $\|s\|_\phi$ subject to the interpolation conditions $\nu_i(s) = \nu_i(f)$ for $1 \le i \le n$. Moreover, s can be written as*

$$s = \lambda_1 \nu_1^r + \lambda_2 \nu_2^r + \cdots + \lambda_n \nu_n^r,$$

for some scalars $\lambda_1, \lambda_2, \dots, \lambda_n$.

For $\xi \in \mathbb{R}^d$, let δ_ξ denote the Dirac delta distribution defined by

$$\delta_\xi(f) := f(\xi).$$

We note that δ_ξ is admissible if and only if $\widehat{\phi} \in L_1(\mathbb{R}^d)$, and in this case, the representor of δ_ξ is $\delta_\xi^r = \delta_\xi * \phi = \phi(\cdot - \xi)$. Furthermore, the Minimal Norm Interpolation Theorem says that if $f \in H_\phi$ and $\xi_1, \xi_2, \dots, \xi_n$ are points in $\mathbb{R}^d$, then the function $s \in H_\phi$, of minimal norm, which satisfies $\delta_{\xi_i}(s) = \delta_{\xi_i}(f)$ (ie. $s(\xi_i) = f(\xi_i)$), $1 \le i \le n$, has the form $s = \sum_{i=1}^n \lambda_i \phi(\cdot - \xi_i)$. In the literature, this method of interpolation is called *radial basis function interpolation* (see [11] and [3]).

For a multi-integer α, the distribution $D^\alpha \delta_\xi$ is defined by

$$D^\alpha \delta_\xi(g) := (-1)^{|\alpha|} D^\alpha g(\xi).$$

It is easy to check that the distributions $D^\alpha \delta_0$, $|\alpha| \le k$, are all admissible if and only if $\int_{\mathbb{R}^d} (1 + |w|^{2k}) \widehat{\phi}(w)\, dw < \infty$ (which, incidentally, implies that $\phi \in C^{2k}(\mathbb{R}^d)$). In this case, the representor of $\nu := D^\alpha \delta_\xi$ is

$$\nu^r = (D^\alpha \delta_\xi) * \phi = D^\alpha \phi(\cdot - \xi).$$

In the literature, this approach, which can be used to solve linear partial differential equations, is called *symmetric collocation* (see [5], [9], and [14]). A general framework for deriving error estimates is given in [6]. In order to illustrate how symmetric collocation can be used to approximately solve a linear partial differential equation, we consider an example using Poisson's equation.

Example 1.5. Let Ω be a bounded domain in $\mathbb{R}^2$ with a piecewise smooth boundary $\partial\Omega$, and consider the problem of finding u satisfying

$$\begin{aligned} \Delta u &= f \text{ in } \Omega, \\ u &= g \text{ on } \partial\Omega, \end{aligned}$$

where $\Delta := D^{(2,0)} + D^{(0,2)}$ denotes the Laplacian operator. Since our problem involves second order derivatives, we assume that our basic function ϕ satisfies $\int_{\mathbb{R}^2}(1+|w|^4)\widehat{\phi}(w)\,dw < \infty$. One first chooses points $\xi_1, \xi_2, \ldots, \xi_n$ in Ω and points $\eta_1, \eta_2, \ldots, \eta_\kappa$ on $\partial\Omega$, and we take, as an approximation to the solution u, the function $s \in H_\phi$ of minimal norm which interpolates u at the linear functionals $\Delta\delta_{\xi_j}$, $1 \le j \le n$, and δ_{η_j}, $1 \le j \le \kappa$. The minimal norm interpolation theorem tells us that s has the form

$$s(x) = \sum_{j=1}^{n} \lambda_j \Delta\phi(x - \xi_j) + \sum_{j=1}^{\kappa} \mu_j \phi(x - \eta_j), \tag{1.6}$$

where the coefficients $\{\lambda_j\}$ and $\{\mu_j\}$ are determined by the interpolation equations $\Delta\delta_{\xi_j}(s) = \Delta\delta_{\xi_j}(u)$ (ie. $\Delta s(\xi_j) = f(\xi_j)$) and $\delta_{\eta_j}(s) = \delta_{\eta_j}(u)$ (ie. $s(\eta_j) = g(\eta_j)$).

A major drawback to the symmetric collocation method is the high numerical cost of evaluating the obtained function s. In the above example, each evaluation of s requires $O(n + \kappa)$ floating point operations (flops); whereas in most of the standard numerical methods for solving this equation (eg. the finite element method), each evaluation of the obtained solution would require $O(1)$ flop. If the linear system of equations which determine the unknown coefficients are solved with an iterative method, then one is also faced with the high cost of evaluating the residual. In the above example, to evaluate the residual, ie to evaluate $\Delta s(\xi_j)$, $1 \le j \le n$, and $s(\eta_j)$, $1 \le j \le \kappa$, requires $O(n+\kappa)^2$ flops.

The primary aim of this paper is to show that the evaluation costs can be substantially reduced if the basic function ϕ is chosen as a tensor product of piecewise polynomials; that is, if ϕ has the form

$$\phi(x) = \psi_1(x_1)\psi_2(x_2)\cdots\psi_d(x_d), \quad x = (x_1, x_2, \dots, x_d) \in \mathbb{R}^d, \tag{1.7}$$

where the functions ψ_i are univariate compactly supported piecewise polynomials. This improvement in efficiency is obtained as certain assumptions begin to be realized. The nature and significance of these assumptions will become clear as the results develop, but we can at least explain how these assumptions would be realized in the above example. For Example 1.5, assume that the points $\{\xi_j\}$ lie on a grid $X \times Y$, where X and Y are subsets of $\mathbb{R}$, each having cardinality $O(\sqrt{n})$. As for the points $\{\eta_j\}$, we assume that their number satisfies $\kappa = O(\sqrt{n})$. This assumption should be reasonable since if one has covered a two dimensional domain with n points, then one should be able to cover the one dimensional piecewise smooth boundary curve with the same density using $O(\sqrt{n})$ points. Under these assumptions it is then possible to evaluate the residual using $O(n)$ flops.

The motivation for our proposed fast evaluation method is largely the same as that of the methods proposed by Beatson and his collaborators (see [1], [12], [4]). Our method differs from those in that we provide exact evaluation in exact arithmetic; whereas these other methods provide approximate evaluation (they incorporate an adjustable trade-off between efficiency and accuracy). Furthermore, special properties of the basic function, which would not be valid when the basic function is a tensor product piecewise polynomial, are required by these previous methods.

An outline of the paper is as follows. In section 2, we prove the abovementioned representation in H_ϕ of admissible distributions; while in section 3 we lay the groundwork for our fast evaluation method. The univariate case of this method is then addressed in section 4, and the multivariate case is treated in section 5. Regarding notation, we mention that when working with vectors in $\mathbb{R}^d$, we prefer that subscripts (as in $\xi_1, \xi_2, \xi_3, \dots$) be used to label a sequence of vectors rather than the components of a single vector. In order to access the components of a vector $\xi \in \mathbb{R}^d$, we employ the standard orthonormal basis $e_1, e_2, \dots, e_d$ for $\mathbb{R}^d$ to write $\xi = (\xi \cdot e_1, \xi \cdot e_2, \dots, \xi \cdot e_d)$. With this notation, the function ϕ in (1.7) will be written $\phi(x) = \psi_1(x \cdot e_1)\psi_2(x \cdot e_2) \cdots \psi_d(x \cdot e_d)$.

2. Representors in H_ϕ of Admissible Distributions

Let ν be a real compactly supported distribution. For test functions $g \in C_c^\infty(\mathbb{R}^d)$, the convolution $\nu * g$ is again a test function and is defined by $(\nu * g)(x) := \nu(\widetilde{g}(\cdot - x))$, where $\widetilde{g} := g(-\cdot)$ (see [13, p.155]). In particular, we obtain

$$\nu(g) = (\nu * \widetilde{g})(0), \quad g \in C_c^\infty(\mathbb{R}^d).$$

We wish to employ this equality as our definition of $\nu(f)$ for $f \in H_\phi$, but first of all, we must define $\nu * g$, say for $g \in L_1 \cup L_2$. Recall that $\widehat{g} \in L_\infty \cup L_2$ by the Plancherel Theorem. Since $\widehat{\nu}$ and all of its derivatives have at most polynomial growth, it follows that $\widehat{\nu}\widehat{g}$ is a tempered distribution, and so we define the tempered distribution $\nu * g$ by

$$(\nu * g)\widehat{\ } := \widehat{\nu}\widehat{g}$$

Proposition 2.1. *Let ν be admissible and let $f \in H_\phi$. Then $\nu * \phi \in H_\phi$ and $(\nu * f)\widehat{\ }$ is integrable.*

Proof. We first mention that the assumptions on ϕ ensure that $\varepsilon := \min_{w \in \mathbb{R}^d} \dfrac{1}{\widehat{\phi}(w)} > 0$; it follows that

$$\varepsilon \, \|\widehat{g}\|^2_{L_2(\mathbb{R}^d)} \le (2\pi)^d \, \|g\|^2_\phi$$

whenever g is a tempered distribution with $\widehat{g}$ locally integrable; and consequently $\|g\|_\phi < \infty$ implies $g \in L_2(\mathbb{R}^d)$. Thus, in order to show that $\nu * \phi \in H_\phi$, it suffices to show that $\|\nu * \phi\|_\phi < \infty$:

$$\|\nu * \phi\|^2_\phi = (2\pi)^{-d} \int_{\mathbb{R}^d} \left|\widehat{\nu}(w)\widehat{\phi}(w)\right|^2 / \widehat{\phi}(w) \, dw = (2\pi)^{-d} \int_{\mathbb{R}^d} |\widehat{\nu}(w)|^2 \, \widehat{\phi}(w) \, dw < \infty.$$

Regarding $(\nu * f)\widehat{\ }$, we see that

$$\|(\nu * f)\widehat{\ }\|_{L_1(\mathbb{R}^d)} = \left\|\widehat{\nu}\widehat{f}\right\|_{L_1(\mathbb{R}^d)} \le \left\|\widehat{\nu}\sqrt{\widehat{\phi}}\right\|_{L_2(\mathbb{R}^d)} \left\|\widehat{f}/\sqrt{\widehat{\phi}}\right\|_{L_2(\mathbb{R}^d)} < \infty,$$

where we have used the Cauchy-Schwarz inequality. $\square$

Since $(\nu * f)\widehat{\ }$ is integrable, it follows that $\nu * f$ is continuous (in particular, $(\nu * f)(0)$ is well-defined) and $(\nu * f)(x) = (2\pi)^{-d} \int_{\mathbb{R}^d} e^{\imath x \cdot w} \, (\nu * f)\widehat{\ }(w) \, dw$ for all $x \in \mathbb{R}^d$. With this in mind, we make our

Definition 2.2. For admissible ν and $f \in H_\phi$, we define

$$\nu(f) := (\nu * \widetilde{f})(0), \qquad \text{where } \widetilde{f} := f(-\cdot).$$

We can now prove that $\nu * \phi$ is the representor of ν in H_ϕ.

Theorem. *Let ν be admissible and put $\nu^r := \nu * \phi$. Then $\nu(f) = \langle f, \nu^r\rangle_\phi$ for all $f \in H_\phi$.*

Proof. Since H_ϕ is a real inner product space, we have, on the one hand,

$$(2\pi)^d\nu(f) = (2\pi)^d(\nu * \widetilde{f})(0) = \int_{\mathbb{R}^d} \left(\nu * \widetilde{f}\right)^\wedge = \int_{\mathbb{R}^d} \widehat{\nu}\widehat{\widetilde{f}} = \int_{\mathbb{R}^d} \widehat{\nu}\overline{\widehat{f}},$$

while on the other hand,

$$(2\pi)^d\langle f, \nu^r\rangle_\phi = (2\pi)^d\langle \nu^r, f\rangle_\phi = \int_{\mathbb{R}^d} \widehat{\nu^r}\overline{\widehat{f}}/\widehat{\phi} = \int_{\mathbb{R}^d} \widehat{\nu}\overline{\widehat{f}}.$$

□

The above representation is well-known in the theory of reproducing kernel Hilbert spaces (see [8], [10]). The above proof is a slight generalization, however, in that the space H_ϕ is not necessarily a reproducing kernel Hilbert space (H_ϕ is not necessarily a subspace of $C(\mathbb{R}^d)$).

3. Fast Evaluation

We are concerned with the fast evaluation of the function s, or of its derivatives, when s is the function of minimal norm obtained when a linear partial differential equation is solved using the symmetric collocation method as described in the introduction. If the basic function ϕ is chosen as a tensor product of univariate piecewise polynomials, then any standard evaluation task can be reduced to several basic evaluation tasks of the following form:

Basic Evaluation Task 3.1. Let ψ be the tensor product of univariate polynomials

$$\psi(x) := \psi_1(e_1 \cdot x)\psi_2(e_2 \cdot x)\cdots\psi_d(e_d \cdot x), \quad x \in \mathbb{R}^d,$$

where ψ_i is a compactly supported piecewise polynomial of degree k_i having m_i nodes. Given distinct translation points $\xi_1, \xi_2, \ldots, \xi_n$ in $\mathbb{R}^d$, real scalars $\lambda_1, \lambda_2, \ldots, \lambda_n$ and evaluation points $Z = \{z_1, z_2, \ldots, z_N\} \subset \mathbb{R}^d$, the *basic evaluation task* is the task of evaluating the function

$$f(x) := \sum_{j=1}^{n} \lambda_j \psi(x - \xi_j)$$

at the points in Z.

To illustrate how such basic evaluation tasks arise, let us consider the context of Example 1.5. If $\phi(x) = \psi_1(e_1 \cdot x)\psi_2(e_2 \cdot x)$, then we can write (1.6) as

$$\begin{aligned} s(x) = \sum_{j=1}^{n} \lambda_j \psi_1''(e_1 \cdot (x - \xi_j))\psi_2(e_2 \cdot (x - \xi_j)) + \sum_{j=1}^{n} \lambda_j \psi_1(e_1 \cdot (x - \xi_j))\psi_2''(e_2 \cdot (x - \xi_j)) \\ + \sum_{j=1}^{\kappa} \mu_j \psi_1(e_1 \cdot (x - \eta_j))\psi_2(e_2 \cdot (x - \eta_j)); \end{aligned}$$

hence the task of evaluating s at Z reduces to three basic evaluation tasks. Similarly, evaluating $D^\alpha s$ also reduces to three basic evaluation tasks, and consequently, evaluating Δs reduces to six basic evaluation tasks.

Our fast evaluation algorithm hinges on the univariate case $d = 1$ which we consider in section 4, leaving the general d-variate case to section 5. However, before that we discuss a few basic ideas regarding polynomials and piecewise polynomials.

Let us adopt the point of view that a polynomial p, of degree k, is numerically represented by the vector $[p_0, p_1, \ldots, p_k]$ in the sense that

$$p(t) = \sum_{\ell=0}^{k} p_\ell t^\ell.$$

It is well-known that p can be evaluated at a real number t using $2k$ flops. For a real scalar λ, the polynomial λp can be rendered[1] using $k + 1$ flops, and if q is another polynomial

[1]The term *render* is used to refer to the task of computing an object's numerical representaton.

of degree k, then the sum $p+q$ (or difference $p-q$) can be rendered using $k+1$ flops. If τ is a real number and q is the translate of p defined by $q := p(\cdot + \tau)$, then we can write $q(t) = \sum_{\ell=0}^{k} q_\ell t^\ell$, where the vector $[q_0, q_1, \dots, q_k]$ can be found using a standard algorithm associated with the Newton form of a polynomial [2, pp. 13–16]:

```
Step 1: Set $q_i = p_i$ for $i = 0, 1, \dots, k$, and set $s = \tau p_k$.

Step 2: For $i = 0, 1, \dots, k-1$ do {
            set $q_{k-1} = q_{k-1} + s$
            set $q_j = q_j + \tau q_{j+1}$ for $j = k-2, k-3, \dots, i$ }
```

It is a simple matter to show that this algorithm uses $k + (k-1) + \cdots + 1$ additions and $1 + (k-1) + (k-2) + \cdots + 1$ products; hence $q = p(\cdot + \tau)$ can be rendered using $k^2 + 1$ flops.

A function $g : \mathbb{R} \to \mathbb{R}$ is said to be a *compactly supported piecewise polynomial* if there exist real numbers $t_1 < t_2 < \cdots < t_m$ and polynomials $g_1, g_2, \dots, g_{m-1}$ such that

$$g(t) = \begin{cases} 0 & \text{if } t < t_1 \text{ or } t \geq t_m, \\ g_j(t - t_j) & \text{if } t_j \leq t < t_{j+1}. \end{cases}$$

We denote the set of all compactly supported piecewise polynomials by $\mathbb{P}$, and we note that it is a translation invariant linear space, all of whose members are right-continuous. The numbers $t_1 < t_2 < \cdots < t_m$ form a *node system* for g with *polynomial pieces* $g_1, g_2, \dots g_{m-1}$. Since there are infinitely many node systems for a given $g \in \mathbb{P}$, we will say that a real number t is an *essential node* for g if it is contained in every node system for g. It is easy to see that the essential nodes of any non-trivial $g \in \mathbb{P}$ form a node system for g; however, in numerical algorithms one usually does not care whether or not the node system in hand contains only essential nodes.

In order to discuss the computational cost of some basic operations with piecewise polynomials, let us assume that the polynomial pieces of g are given as polynomials of degree at most k. It is easy to see that the task of evaluating g at a real number t requires at most $1 + 2k$ flops and to multiply g by a real scalar requires at most $(m-1)(k+1)$ flops. The translate $g(\cdot + \tau)$ can be rendered simply by subtracting τ from each node, an operation which requires m flops. The piecewise polynomials in $\mathbb{P}$ have the interesting property that for any $g \in \mathbb{P}$ and $y \in \mathbb{R}$, there exists a unique polynomial $P_y g$ such that

$$(P_y g)(t) = g(y+t) \text{ for all } t \in [0, \varepsilon),$$

provided $\varepsilon > 0$ is sufficiently small. The linear operator $P_y : \mathbb{P} \to \Pi$ will play an important role in the following section. If $y < t_1$ or $y \geq t_m$, then $P_y g = 0$, and if $y = t_i$, for some $1 \leq i \leq m-1$, then $P_y g = g_i$. On the other hand, if $t_i < y < t_{i+1}$, then $P_y g = g_i(\cdot + y - t_i)$. Hence, the task of rendering the polynomial $P_y g$ requires $k^2 + 2$ flops if $y \in [t_1, t_m] \backslash \{t_1, t_2, \ldots, t_m\}$ and requires 0 flops otherwise.

4. Fast evaluation in the univariate case

We consider the Basic Evaluation Task in the univariate case $d = 1$. If the function

$$f(t) := \sum_{j=1}^{n} \lambda_j \psi(t - \xi_j), \quad t \in \mathbb{R}, \tag{4.1}$$

is evaluated directly, then each evaluation requires n subtractions, n evaluations of ψ, n products and $n-1$ additions for a potential total of $2n(k+2)-1$ flops. Hence, performing the Basic Evaluation Task directly may require $N(2n(k+2)-1)$ flops. An alternate approach is to first render f as a piecewise polynomial (of degree k), and then evaluate it at Z. The efficiency of this alternate approach depends on an efficient means of rendering f, which we now present.

We assume that ψ has been rendered with nodes $t_1 < t_2 < \cdots < t_m$ and polynomial pieces $s_1, s_2, \ldots, s_{m-1}$, each of degree k, and we assume that the translation points $\{\xi_j\}$ have been sorted as $\xi_1 < \xi_2 < \cdots < \xi_n$. Noting that the translate $\psi(\cdot - \xi_j)$ has nodes $t_1 + \xi_j, t_2 + \xi_j, \ldots, t_m + \xi_j$, we see that a node system for f can be obtained as the distinct entries in the list

$$(t_i + \xi_j : 1 \le i \le m,\ 1 \le j \le n), \tag{4.2}$$

which we will denote $x_1 < x_2 < \cdots < x_M$, and the polynomial pieces of f will be denoted $f_1, f_2, \ldots, f_{M-1}$. The idea behind our fast rendering method is that once a particular polynomial piece f_r is known, the adjacent piece f_{r+1} can be computed very efficiently. Once f_{r+1} has been computed, we can then efficiently compute f_{r+2}, and so on...
To see how this is done, let us assume that $f_r = P_{x_r} f$ has been rendered and we consider the task of rendering $f_{r+1} = P_{x_{r+1}} f$. For the sake of simplicity, let us assume (for the moment) that the node x_{r+1} appears only once in the above list as $x_{r+1} = t_{i_0} + \xi_{j_0}$. Then the node x_{r+1} is *caused* by the term $\lambda_{j_0}\psi(\cdot - \xi_{j_0})$ in (4.1), and thus $f_0(t) := f(t) - \lambda_{j_0}\psi(t - \xi_{j_0})$ does not have an essential node at x_{r+1}. Consequently, $P_{x_{r+1}} f_0$ can be obtained from $P_{x_r} f_0$ simply by polynomial translation:

$$P_{x_{r+1}} f_0 = [P_{x_r} f_0](\cdot + x_{r+1} - x_r).$$

But since f and f_0 differ by only one term, it is then possible to express $P_{x_{r+1}} f$ as the sum of $[P_{x_r} f](\cdot + x_{r+1} - x_r)$ and some remainder which is *caused* by the outstanding term $\lambda_{j_0}\psi(\cdot - \xi_{j_0})$. The resulting relation is

$$f_{r+1} = f_r(\cdot + x_{r+1} - x_r) + \lambda_{j_0}(s_{i_0} - s_{i_0 - 1}(\cdot + t_{i_0} - t_{i_0 - 1})),$$

from which we conclude that f_{r+1} can be rendered with only $O(k^2)$ flops. In order to eliminate repeated computations, it is advisable to render and save, in advance, the polynomials $Q_1, Q_2, \ldots, Q_m$ defined by

$$Q_i := s_i - s_{i-1}(\cdot + t_i - t_{i-1}), \quad i = 1, 2, \ldots, m,$$

where $s_0 = s_m = 0$.

In order to address the general case, we assume that the list (4.2) has been sorted as $y_1 \leq y_2 \leq \cdots \leq y_{mn}$ with pointers $\mathbf{i} : \{1, 2, \ldots, mn\} \to \{1, 2, \ldots, m\}$ and $\mathbf{j} : \{1, 2, \ldots, mn\} \to \{1, 2, \ldots, n\}$ satisfying $y_\ell = t_{\mathbf{i}(\ell)} + \xi_{\mathbf{j}(\ell)}$ for $\ell = 1, 2, \ldots, mn$.

Proposition 4.3. *Let $r \in \{1, 2, \ldots, M-2\}$. If ℓ and u are defined by*
$x_r = y_\ell < y_{\ell+1} = x_{r+1} = y_{\ell+u} < y_{\ell+u+1} = x_{r+2}$, then

$$f_{r+1} = f_r(\cdot + x_{r+1} - x_r) + \sum_{i=1}^{u} \lambda_{\mathbf{j}(\ell+i)} Q_{\mathbf{i}(\ell+i)}.$$

Proof. Put $x = x_r$ and $\tau = x_{r+1} - x_r$, and define the linear operator L by $Lg := P_{x+\tau} g - T_\tau P_x g$, where T_τ is the translation operator defined by $T_\tau g := g(\cdot + \tau)$. Note that $Lg = 0$ whenever $g \in \mathbb{P}$ is a piecewise polynomial having no essential nodes in the interval $(x, x+\tau]$. The assumptions in force ensure that only the translates $\psi(\cdot - \xi_j)$, for $j = \mathbf{j}(\ell+1)$, $\mathbf{j}(\ell+2), \ldots, \mathbf{j}(\ell+u)$, have nodes at x_{r+1}, and hence $L[\psi(\cdot - \xi_j)] = 0$ for all other indices j. We can thus write

$$f_{r+1} - f_r(\cdot + x_{r+1} - x_r) = Lf = \sum_{j=1}^{n} \lambda_j L[\psi(\cdot - \xi_j)] = \sum_{i=1}^{u} \lambda_{\mathbf{j}(\ell+i)} L[\psi(\cdot - \xi_{\mathbf{j}(\ell+i)})].$$

In order to complete the proof, it suffices to show that $L[\psi(\cdot - \xi_{\mathbf{j}(\ell+i)})] = Q_{\mathbf{i}(\ell+i)}$ for $i = 1, 2, \ldots, u$. For this, assume $1 \leq i \leq u$ and let us write $\mathbf{i} = \mathbf{i}(\ell+i)$ and $\mathbf{j} = \mathbf{j}(\ell+i)$, for the sake of brevity. Since $x_{r+1} = y_{\ell+i} = t_{\mathbf{i}} + \xi_{\mathbf{j}}$, it follows that $x_{r+1} - \xi_{\mathbf{j}} = t_{\mathbf{i}}$, and hence

$$t_{\mathbf{i}-1} \leq x - \xi_{\mathbf{j}} < t_{\mathbf{i}},$$

with the understanding that $t_0 = x - \xi_{\mathbf{j}}$ (in case $\mathbf{i} = 1$). Noting that $P_x[\psi(\cdot - \xi_{\mathbf{j}})] = P_y\psi$, with $y := x - \xi_{\mathbf{j}}$, and with the above inequality in view, we see that $P_y\psi = s_{\mathbf{i}-1}(\cdot + y - t_{\mathbf{i}-1})$. Therefore,

$$T_\tau P_x[\psi(\cdot - \xi_{\mathbf{j}})] = T_\tau P_y\psi = s_{\mathbf{i}-1}(\cdot + y - t_{\mathbf{i}-1} + \tau) = s_{\mathbf{i}-1}(\cdot + t_{\mathbf{i}} - t_{\mathbf{i}-1}).$$

On the other hand, noting that $(x + \tau) - \xi_{\mathbf{j}} = x_{r+1} - \xi_{\mathbf{j}} = t_{\mathbf{i}}$, we see that

$$P_{x+\tau}[\psi(\cdot - \xi_{\mathbf{j}})] = P_{t_{\mathbf{i}}}\psi = s_{\mathbf{i}}.$$

Hence $L[\psi(\cdot - \xi_{\mathbf{j}})] = s_{\mathbf{i}} - s_{\mathbf{i}-1}(\cdot + t_{\mathbf{i}} - t_{\mathbf{i}-1}) = Q_{\mathbf{i}}$ which completes the proof. $\square$

Proposition 4.3 leads immediately to the following

Fast Rendering Method. The nodes of f will be denoted $x_1, x_2, \dots, x_M$ and the polynomial pieces of f will be denoted $f_1, f_2, \dots, f_{M-1}$.

Step 1: Set $Q_i := s_i - s_{i-1}(\cdot + t_i - t_{i-1}), \quad \text{for} \quad i = 1, 2, 3, \dots, m.$

Step 2: Form and sort the list $(t_i + \xi_j : 1 \le i \le m,\ 1 \le j \le n)$ as

$$y_1 \le y_2 \le \cdots \le y_{mn} \quad \text{with} \quad y_\ell = t_{\mathbf{i}(\ell)} + \xi_{\mathbf{j}(\ell)},$$

and let $x_1 < x_2 < \cdots < x_M$ denote the distinct values in $\{y_1, y_2, \dots, y_{mn}\}$.

Step 3: Set $f_1 = \lambda_{\mathbf{j}(1)} Q_{\mathbf{i}(1)}$ and $r_0 = r = 1$.

```
Step 4: For $\ell = 2, 3, \dots, mn - 1$ do {
        if $y_\ell = y_{\ell-1}$
            set $f_r = f_r + \lambda_{\mathbf{j}(\ell)} Q_{\mathbf{i}(\ell)}$.
        otherwise
            set $r = r + 1$ and $f_r = f_{r-1}(\cdot + x_r - x_{r-1}) + \lambda_{\mathbf{j}(\ell)} Q_{\mathbf{i}(\ell)}$.
        }
```

Theorem 4.4. *The rendering of the piecewise polynomial f using the Fast Rendering Method requires no more than $m(k^2+2k+5)(n+1)$ flops.*

Proof. Since $s_0 = s_m = 0$, step 1 requires $m-1$ subtractions, $m-1$ polynomial translations, and $m-2$ polynomial subtractions which amounts to $(m-1)(k^2+2)+(m-2)(k+1)$ flops. Step 2 requires mn sums and step 3 requires $k+1$ products. For step 4, we see that for each value $\ell \in \{2,3,\dots,mn-1\}$, at most one subtraction, one polynomial translation, one multiplication of a polynomial by a scalar, and one polynomial addition are performed. Hence, step 4 requires at most $(mn-2)(k^2+2+2(k+1))$ flops. Adding these flop counts shows that the Fast Rendering Method requires at most $m(k^2+2k+5)(n+1)-3k^2-k(m+5)-2m-11$ flops. $\square$

Note that if m and k are constant, then the Fast Rendering Method requires $O(n)$ flops. Although the Fast Rendering Method is exact, in exact arithmetic, numerical experiments using floating point arithmetic show that the rendering loses accuracy as one marches away from the starting point x_1. To counter this loss of accuracy, it is advisable to periodically refresh the computation by directly computing a polynomial piece f_r, rather than rely on Proposition 4.3. We note that a particular polynomial piece f_r can be computed directly as

$$f_r = P_{x_r} f = \sum_{j=1}^{n} \lambda_j P_{x_r}[\psi(\cdot - \xi_j)] = \sum_{j=1}^{n} \lambda_j P_{x_r - \xi_j}\psi. \tag{4.5}$$

Proposition 4.6. *The direct computation of a particular polynomial piece f_r via (4.5) requires no more than*

$$n_r\,(k^2+2k+4) \text{ flops, where } n_r := \#\{j : t_1 \le x_r - \xi_j < t_m\}.$$

Proof. Since $P_{x_r-\xi_j}\psi = 0$ whenever $x_r - \xi_j \notin [t_1, t_m)$, we see that the rightmost sum in (4.5) contains at most n_r nonzero terms. In case $n_r = 0$, the conclusion is clear since f_r would equal 0; so let us assume $n_r > 0$. As mentioned at the end of the previous section, these nonzero terms $\{P_{x_r-\xi_j}\psi\}$ can be computed using at most $n_r(k^2+2)$ flops, and multiplying them by the scalars $\{\lambda_j\}$ requires an additional $n_r(k+1)$ flops. Finally, the sum of the n_r nonzero polynomials $\{\lambda_j P_{x_r-\xi_j}\psi\}$ requires $(n_r-1)(k+1)$ flops. Adding these flop counts shows that the computation of f_r requires at most $n_r(k^2+2k+4)-(k+1)$ flops. $\square$

In order to decide when to use (4.5) rather than Proposition 4.3, the author suggests employing the notion of a *trust radius* R_ψ with the following understanding: Suppose f_{r_0-1} has been computed directly using (4.5) and then subsequent pieces $f_{r_0}, f_{r_0+1}, \ldots, f_r$ have been computed using Proposition 4.3. The piece f_r will be *trusted* (considered accurate) if $x_{r+1} - x_{r_0} \le R_\psi$. In order to illustrate how a suitable choice of a trust radius can be found experimentally we consider the following

Example 4.7. Let ψ be the multiple of Wendland's function $\phi_{1,3}$ (see [15]), given by

$$\psi(t) := \begin{cases} (1-|t|)^7(21\,|t|^3 + 19t^2 + 7\,|t| + 1) & \text{if } |t| \le 1 \\ 0 & \text{if } |t| > 1 \end{cases},$$

which is an even piecewise polynomial of degree 10, is supported on $[-1,1]$, and has three nodes $\{-1,0,1\}$. Wendland has shown that the Fourier transform of ψ satisfies

$$K_1(1+|w|)^{-8} \le \widehat{\psi}(w) \le K_2(1+|w|)^{-8}, \quad w \in \mathbb{R}, \tag{4.8}$$

for some positive constants K_1, K_2. Incidentally, it follows from (4.8) that $\int_{\mathbb{R}}(1+|w|^6)\widehat{\psi}(w)\,dw < \infty$, which implies that $\psi \in C^6(\mathbb{R})$. We randomly choose translation points $-1 = \xi_1 \le \xi_2 \le$

$\cdots \le \xi_n \le 2$ and coefficients $\lambda_j \in [-1,1]$, and then render f over the interval $[x_{r_0-1}, x_M]$, where $x_{r_0} = 0$. The rendering is obtained by directly computing f_{r_0-1} using (4.5) and computing subsequent pieces using Proposition 4.3. Denoting the resultant piecewise polynomial by $\mathbf{f}$, which is not exactly equal to f due to round-off errors, we compute the largest interval $[0,b]$ for which

$$\|f - \mathbf{f}\|_{L_\infty([0,b])} \le (3 \times 10^{-13}) \left\|\overline{f}\right\|_{L_\infty(\mathbb{R})},$$

where $\overline{f}(x) := \sum_{j=1}^n |\lambda_j \psi(x - \xi_j)|$. The computation of $\mathbf{f}$ and its evaluation is performed using double precision arithmetic as defined by IEEE's *Binary Floating Point Arithmetic Standard 754-1985.* The computation of the 'exact' values of f is performed using GNU's multiple precision library gmp-3.1.1. After 1021 independent runs, using values $n = 4, 5, \ldots, 1024$, we find that the intersection of all obtained intervals $[0,b]$ is $[0, 0.52]$. Based on this, we choose the trust radius to be $R_\psi = 0.52$. It is encouraging to note that the obtained interval $[0, 0.52]$ is fairly independent of n; for example, if the same experiment is run with $n = 32$ (still 1021 independent runs), the smallest interval is $[0, 0.61]$.

Using the trust radius R_ψ to decide when to refresh the rendering computation in the Fast Rendering Method leads to the following variant.

Stabilized Fast Rendering Method. Steps 1,2,3 are the same as in the Fast Rendering Method, but step 4 becomes

```
Step 4': For $\ell = 2, 3, \ldots, mn-1$ do {
            if $y_\ell = y_{\ell-1}$
                set $f_r = f_r + \lambda_{\mathbf{j}(\ell)} Q_{\mathbf{i}(\ell)}$.
otherwise {
    if $x_{r+1} - x_{r_0} > R_\psi$, set $r_0 = r+1$ and directly compute $f_r$ using (4.5).
    set $r = r+1$ and $f_r = f_{r-1}(\cdot + x_r - x_{r-1}) + \lambda_{\mathbf{j}(\ell)} Q_{\mathbf{i}(\ell)}$. }
}
```

Theorem 4.9. *The rendering of the piecewise polynomial f using the Stabilized Fast Rendering Method requires no more than*

$$m(k^2+2k+5)(n+1) + (m + \lceil L/R_\psi \rceil (k^2+2k+4))n \text{ flops,} \tag{4.10}$$

where $L = t_m - t_1$ denotes the length of the support interval of ψ.

Proof. The required cost (in terms of flops) is the same as the cost of the Fast Rendering Method, except for the additional cost in step $4'$ of directly computing f_r, say for $r \in \{r_1 < r_2 < \cdots < r_{\mathcal{N}}\}$, and of computing the differences $x_{r+1} - x_{r_0}$, of which there are less than mn. By Proposition 4.6, the cost of directly computing these polynomial pieces is at most $\sum_{i=1}^{\mathcal{N}} n_{r_i}(k^2+2k+4)$ flops. We first note that

$$\sum_{i=1}^{\mathcal{N}} n_{r_i} = \sum_{j=1}^{n} \#\{i : t_1 \le x_{r_i} - \xi_j < t_m\} = \sum_{j=1}^{n} \#\{i : t_1 < x_{1+r_i} - \xi_j \le t_m\},$$

where the last equality holds since f has no nodes in the open interval (x_{r_i}, x_{1+r_i}). Since $x_{1+r_{i+1}} - x_{1+r_i} > R_\psi$, it follows that $\#\{i : t_1 < x_{1+r_i} - \xi_j \le t_m\} \le \lceil L/R_\psi \rceil$, and hence that $\sum_{i=1}^{\mathcal{N}} n_{r_i}(k^2+3k+3) \le n\lceil L/R_\psi \rceil (k^2+2k+4)$. $\square$

We mention that the first term in (4.10) estimates the cost of the Fast Rendering Method and the second term estimates the additional costs which arise in step $4'$.

Corollary 4.11. *If the Basic Evaluation Task is performed by first rendering f using the Stabilized Fast Rendering Method and then evaluating f at the points in Z, then this requires no more than*

$$(m + \lceil L/R_\psi \rceil)(k^2 + 2k + 6)(n + 1) + (2k + 1)N \textit{ flops.}$$

To illustrate the potential improvement in efficiency, we mention that if N is proportional to n (and ψ is fixed), then the above flop count is $O(n)$; whereas directly performing the Basic Evaluation Task (as mentioned at the beginning of this section) requires $O(n^2)$ flops. In practice it is often the case that one has settled on the choice of a particular function ψ, but has left the choice of *scale* open. In other words, one intends to use the dilate $\psi(\sigma\cdot)$, where the dilation parameter σ is left as a tuning parameter. Note that the length of the support interval for the dilate $\psi(\sigma\cdot)$ is $L_\sigma = L/\sigma$. If R has been chosen as the trust radius for the function ψ, we suggest that the trust radius for the dilate $\psi(\sigma\cdot)$ be chosen as $R_\sigma = R/\sigma$. In this case we will have $L_\sigma/R_\sigma = L/R$ and hence the cost estimates in Theorem 4.9 and Corollary 4.11 are independent of the dilation parameter σ.

Example 4.12. With $n = 1024$, let f, $\overline{f}$, ψ and $\{\xi_j\}$ be as in Example 4.7 except that we will employ the dilate $\psi(\sigma\cdot)$ in place of ψ, and the translation points $\{\xi_j\}$ are chosen randomly in the interval $[-6, 6]$. Let $\mathbf{f}$ denote the rendering of f obtained using the Stabilized Fast Rendering Method with trust radius $R = 0.52/\sigma$. For each dilation value $\sigma \in \{\frac{1}{4}, \frac{1}{2}, 1, 2\}$, we make 1024 independent runs and record the average (over 1024 runs) number of flops used to obtain $\mathbf{f}$ and the maximum (over 1024 runs) of the normalized error: $\|f - \mathbf{f}\|_{L_\infty(\mathbb{R})} \,/\, \|\overline{f}\|_{L_\infty(\mathbb{R})}$.

σ	1/4	1/2	1	2
average flops	$852n$	$851n$	$849n$	$844n$
normalized error	3.6×10^{-14}	4.9×10^{-14}	7.1×10^{-14}	9.0×10^{-14}

Running the same experiments using the second derivative ψ'' in place of ψ and trust radius $R = 0.40/\sigma$, we find that

σ	$1/4$	$1/2$	1	2
average flops	$676n$	$675n$	$672n$	$667n$
normalized error	3.1×10^{-14}	3.8×10^{-14}	5.5×10^{-14}	7.4×10^{-14}

Running the same experiments using the fourth derivative ψ^{iv} in place of ψ and trust radius $R = 0.54/\sigma$, we find that

σ	$1/4$	$1/2$	1	2
average flops	$353n$	$353n$	$352n$	$350n$
normalized error	2.4×10^{-14}	2.9×10^{-14}	3.0×10^{-14}	4.6×10^{-14}

5. Fast evaluation in the multivariate case

Let $\mathcal{M}_1$ be a method for performing the Basic Evaluation Task in the univariate case $d = 1$. We will show that one can then obtain, recursively, methods $\mathcal{M}_d$, $d = 2, 3, 4, \ldots$, for the general case. For this, we consider the Basic Evaluation Task assuming that methods $\mathcal{M}_1, \mathcal{M}_2, \ldots \mathcal{M}_{d-1}$ have been defined.

For vectors $x \in \mathbb{R}^d$, we define $x' := (e_1 \cdot x, e_2 \cdot x, \ldots, e_{d-1} \cdot x) \in \mathbb{R}^{d-1}$, and for subsets $X \subseteq \mathbb{R}^d$, we define $X' := \{x' : x \in X\} \subseteq \mathbb{R}^{d-1}$. For $x \in \mathbb{R}^d$, let us write

$$\psi(x) = \psi_{<d}(x')\, \psi_d(e_d \cdot x), \qquad \text{where } \psi_{<d}(y) := \psi_1(e_1 \cdot y)\psi_2(e_2 \cdot y) \cdots \psi_{d-1}(e_{d-1} \cdot y).$$

Put $\Xi := \{\xi_1, \xi_2, \ldots, \xi_n\}$, and define

$$n_i := \#\, e_i \cdot \Xi \quad \text{and} \quad N_i := \#\, e_i \cdot Z, \quad \text{for } i = 1, 2, \ldots, d,$$

where $e_i \cdot X$ denotes the set $\{e_i \cdot x : x \in X\}$. With $\{x_1, x_2, \cdots, x_{n_d}\} := e_d \cdot \Xi$ and $\Xi_\ell := \{\xi \in \Xi : e_d \cdot \xi = x_\ell\}$, we see that $\Xi_1, \Xi_2, \ldots, \Xi_{n_d}$ is a partition of Ξ, and hence, for $z \in \mathbb{R}^d$, we can write

$$f(z) = \sum_{\ell=1}^{n_d} G_\ell(z')\psi_d(e_d \cdot z - x_\ell), \qquad \text{where} \qquad G_\ell(z') := \sum_{\xi \in \Xi_\ell} \lambda_\xi \psi_{<d}(z' - \xi').$$

Note that, for each ℓ, G_ℓ is a $(d-1)$-variate function which can be evaluated using method $\mathcal{M}_{d-1}$. Using method $\mathcal{M}_{d-1}$, we compute $G_\ell(w)$ for all $w \in Z'$, $1 \le \ell \le n_d$. Then, for each $w \in Z'$, we use method $\mathcal{M}_1$ to evaluate the univariate function $\sum_{\ell=1}^{n_d} G_\ell(w)\psi_d(\cdot - x_\ell)$ at $\{e_d \cdot z : z \in Z \text{ with } z' = w\}$; thus obtaining the values of $f(z)$ for $z \in Z$ with $z' = w$. As w ranges over Z', we obtain all values of $f(z)$ for $z \in Z$. We summarize this algorithm as follows:

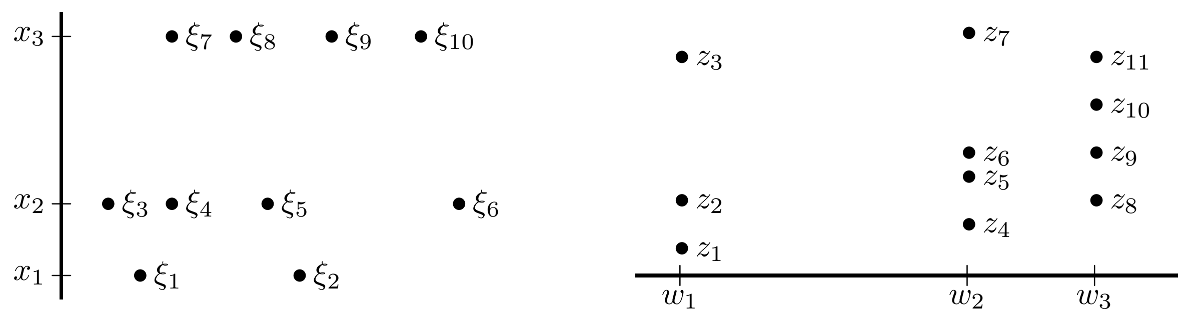
Recursive Evaluation Algorithm.

Step 1: For $\ell = 1, 2, \ldots, n_d$, use method $\mathcal{M}_{d-1}$ to evaluate the $(d-1)$-variate function G_ℓ at all points in Z'; thus obtaining the values $G_\ell(w)$ for all $w \in Z'$, $1 \le \ell \le n_d$.

Step 2: For each $w \in Z'$, use method $\mathcal{M}_1$ to evaluate the univariate function $\sum_{\ell=1}^{n_d} G_\ell(w)\psi_d(\cdot - x_\ell)$ at $\{e_d \cdot z : z \in Z \text{ with } z' = w\}$; thus obtaining the values $f(z)$ for all $z \in Z$.

We illustrate the Recursive Evaluation Algorithm with a simple example in two dimensions.

Example. Suppose $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{10}\}$ and $Z = \{z_1, z_2, \ldots, z_{11}\}$ are as indicated:



With $z = (w, x)$, we write $f(z) = \sum_{i=1}^{10} \lambda_i \psi_1(w - e_1 \cdot \xi_i)\psi_2(x - e_2 \cdot \xi_i)$ in the form

$$f(z) = G_1(w)\psi_2(x - x_1) + G_2(w)\psi_2(x - x_2) + G_3(w)\psi_2(x - x_3), \text{ where}$$

$$G_1(w) = \lambda_1\psi_1(w - e_1 \cdot \xi_1) + \lambda_1\psi_1(w - e_1 \cdot \xi_2),$$

$$G_2(w) = \lambda_3\psi_1(w - e_1 \cdot \xi_3) + \lambda_4\psi_1(w - e_1 \cdot \xi_4) + \lambda_5\psi_1(w - e_1 \cdot \xi_5) + \lambda_6\psi_1(w - e_1 \cdot \xi_6),$$

$$G_3(w) = \lambda_7\psi_1(w - e_1 \cdot \xi_7) + \lambda_8\psi_1(w - e_1 \cdot \xi_8) + \lambda_9\psi_1(w - e_1 \cdot \xi_9) + \lambda_{10}\psi_1(w - e_1 \cdot \xi_{10}).$$

In step 1, method $\mathcal{M}_1$ is used to evaluate G_ℓ at $Z' = \{w_1, w_2, w_3\}$ for $\ell = 1, 2, 3$, thus obtaining the values $\begin{pmatrix} G_1(w_1) & G_2(w_1) & G_3(w_1) \\ G_1(w_2) & G_2(w_2) & G_3(w_2) \\ G_1(w_3) & G_2(w_3) & G_3(w_3) \end{pmatrix}$
Then in step 2, method $\mathcal{M}_1$ is used to evaluate:

$f(w_1, x) = G_1(w_1)\psi_2(x - x_1) + G_2(w_1)\psi_2(x - x_2) + G_3(w_1)\psi_2(x - x_3)$ at $x \in e_2 \cdot \{z_1, z_2, z_3\}$ (obtaining $f(z_1), f(z_2), f(z_3)$),

$f(w_2, x) = G_1(w_2)\psi_2(x - x_1) + G_2(w_2)\psi_2(x - x_2) + G_3(w_2)\psi_2(x - x_3)$ at $x \in e_2 \cdot \{z_4, z_5, z_6, z_7\}$ (obtaining $f(z_4), f(z_5), f(z_6), f(z_7)$),

$f(w_3, x) = G_1(w_3)\psi_2(x - x_1) + G_2(w_3)\psi_2(x - x_2) + G_3(w_3)\psi_2(x - x_3)$ at $x \in e_2 \cdot \{z_8, z_9, z_{10}, z_{11}\}$ (obtaining $f(z_8), f(z_9), f(z_{10}), f(z_{11})$).

We now estimate the number of flops required by the Recursive Evaluation Algorithm. Let us suppose, in the univariate case, that the basic evaluation task can be performed by method $\mathcal{M}_1$ using at most $F_1(\psi, \Xi, Z)$ flops. For $d > 1$, we define $F_d(\psi, \Xi, Z)$ recursively by

$$F_d(\psi, \Xi, Z) := F_{d-1}(\psi_{<d}, \Xi', Z')n_d + F_1(\psi_d, e_d \cdot \Xi, e_d \cdot Z)\,\#Z'.$$

It is fairly easy to see (by induction) that the Recursive Evaluation Algorithm can be performed using at most $F_d(\psi, \Xi, Z)$ flops: Step 1 requires $F_{d-1}(\psi_{<d}, \Xi', Z')n_d$ flops (as the mapping $\xi \mapsto \xi'$, from Ξ_ℓ to Ξ', is injective) and Step 2 requires at most $F_1(\psi_d, e_d \cdot \Xi, e_d \cdot Z)\,\#Z'$

flops, where we have employed the inclusion $\{e_d \cdot z : z \in Z \text{ with } z' = w\} \subset e_d \cdot Z$.

The inequality $\#Z' \leq N_1 N_2 \cdots N_{d-1}$ leads to the following

Theorem 5.1.

$$F_d(\psi, \Xi, Z) \leq \sum_{i=1}^{d} \left(\prod_{1 \leq j < i} N_j \right) F_1(\psi_i, e_i \cdot \Xi, e_i \cdot Z) \left(\prod_{i < j \leq d} n_j \right).$$

Proof. The case $d = 1$ holds with equality. Proceeding by induction, we assume the inequality for $d - 1$ and consider d. Then

$$\begin{aligned} F_d(\psi, \Xi, Z) &= F_{d-1}(\psi_{<d}, \Xi', Z') n_d + F_1(\psi_d, e_d \cdot \Xi, e_d \cdot Z)\, \#Z' \\ &\leq \sum_{i=1}^{d-1} \left(\prod_{1 \leq j < i} N_j \right) F_1(\psi_i, e_i \cdot \Xi', e_i \cdot Z') \left(\prod_{i < j \leq d-1} n_j \right) n_d \\ &\qquad + F_1(\psi_d, e_d \cdot \Xi, e_d \cdot Z)\, N_1 N_2 \cdots N_{d-1} \\ &= \sum_{i=1}^{d} \left(\prod_{1 \leq j < i} N_j \right) F_1(\psi_i, e_i \cdot \Xi, e_i \cdot Z) \left(\prod_{i < j \leq d} n_j \right), \end{aligned}$$

which completes the induction. □

In order to better appreciate Theorem 5.1, we mention that the right side of the estimate is a sum of d terms, where each term is the product of d factors. Specifically, the j-th factor of the i-th term equals N_j if $j < i$, equals $F_1(\psi_i, e_i \cdot \Xi, e_i \cdot Z)$ if $j = i$ and equals n_j if $j > i$.

If the method $\mathcal{M}_1$, which until now has been left unspecified, is taken as described in Corollary 4.11, then we can take

$$F_1(\psi_i, e_i \cdot \Xi, e_i \cdot Z) = (m_i + \lceil L_i / R_i \rceil)(k_i^2 + 2k_i + 6)(1 + \#\, e_i \cdot \Xi) + (2k_i + 1)\, \#\, e_i \cdot Z, \tag{5.2}$$

where k_i and m_i denote, respectively, the degree and number of nodes of ψ_i, and L_i/R_i is the ratio between the length of the support interval and the trust radius for ψ_i.

Example 5.3. Let us return to example 1.5 in the specific case when Ω is the region bounded by the cardioid given in polar coordinates by $r = 2 + 2\cos\theta$, and ϕ is the tensor product function $\phi(x) = \psi(e_1 \cdot x)\psi(e_2 \cdot x)$, where ψ is the piecewise polynomial used in examples 4.7 and 4.12. Given $h > 0$, with $\frac{1}{2h} \in \mathbb{N}$ assumed for simplicity, we choose $\Xi := \{\xi_1, \xi_2, \ldots, \xi_n\} := h\mathbb{Z}^2 \cap \Omega$ and let $\Gamma := \{\eta_1, \eta_2, \ldots, \eta_\kappa\}$ be points around $\partial\Omega$ (with $\eta_1 = (4, 0)$) which are equispaced by a distance h with respect to arclength along the cardioid. Since Ω has area 6π, we can say $n \approx 6\pi/h^2$, and since the length of the cardioid is 16 we have $\kappa = 16/h$. With the function s written as in (1.6), we will concern ourselves with the tasks of evaluating s at the points in Γ and evaluating Δs at the points in Ξ (this amounts to evaluating the residual for the collocation equations). Writing s as

$$s(x) = \sum_{j=1}^{\kappa} \mu_j \phi(x - \eta_j) + \sum_{j=1}^{n} \lambda_j D^{(2,0)}\phi(x - \xi_j) + \sum_{j=1}^{n} \lambda_j D^{(0,2)}\phi(x - \xi_j),$$

we see that the task of evaluating s at the points in Γ comprises three basic evaluation tasks (see 3.1), which we label I, II and III, and writing Δs as

$$\begin{aligned}\Delta s(x) = \sum_{j=1}^{\kappa} \mu_j D^{(2,0)}\phi(x - \eta_j) + \sum_{j=1}^{\kappa} \mu_j D^{(0,2)}\phi(x - \eta_j)\\ + \sum_{j=1}^{n} \lambda_j D^{(4,0)}\phi(x - \xi_j) + 2\sum_{j=1}^{n} \lambda_j D^{(2,2)}\phi(x - \xi_j) + \sum_{j=1}^{n} \lambda_j D^{(0,4)}\phi(x - \xi_j),\end{aligned}$$

we see that the task of evaluating Δs at the points in Ξ comprises five basic evaluation tasks, which we label IV, V, VI, VII and VIII. We wish to use Theorem 5.1 along with (5.2) to estimate the number of flops needed for each of these basic evaluation tasks assuming method $\mathcal{M}_1$ is as described in Corollary 4.11. Toward this end, we mention that

$$(5.4) \quad \# e_1 \cdot \Xi = \frac{9}{2h}, \quad \# e_2 \cdot \Xi = 2\lfloor \frac{3\sqrt{3}}{2h} \rfloor + 1, \quad \# e_1 \cdot \Gamma = \frac{8}{h} + 1, \quad \# e_2 \cdot \Gamma = \frac{16}{h} - 1.$$

Furthermore, we list the following details regarding the functions ψ, ψ'' and ψ^{iv}:

	degree	#nodes	R	L	$\lceil L/R \rceil$
ψ	10	3	0.52	2	4
ψ''	8	3	0.40	2	5
ψ^{iv}	6	3	0.54	2	4

In order to illustrate the use of Theorem 5.1 and (5.2), let us consider basic evaluation task IV, which employs the tensor product function $D^{(2,0)}\phi(x) = \psi''(e_1 \cdot x)\psi(e_2 \cdot x)$. By Theorem 5.1 (and the observation preceding it), we have

$$\text{flops(IV)} \leq F_1(\psi'', e_1 \cdot \Gamma, e_1 \cdot \Xi)(\# e_2 \cdot \Gamma) + (\# e_1 \cdot \Xi)F_1(\psi, e_2 \cdot \Gamma, e_2 \cdot \Xi),$$

and then (5.2) yields $F_1(\psi'', e_1 \cdot \Gamma, e_1 \cdot \Xi) \leq 688(1 + \# e_1 \cdot \Gamma) + 17(\# e_1 \cdot \Xi)$ and $F_1(\psi, e_2 \cdot \Gamma, e_2 \cdot \Xi) \leq 882(1 + \# e_2 \cdot \Gamma) + 21(\# e_2 \cdot \Xi)$. After substituting the values in (5.4) and employing a simplifying estimate, we see that flops(IV) $\leq 153284 h^{-2} + O(h^{-1})$. The cost (in terms of flops) of the remaining basic evaluation tasks can be estimated in a similar manner to obtain

flops(I)	$\leq$	$231168\, h^{-2} + O(h^{-1})$	flops(V)	$\leq$	$164342\, h^{-2} + O(h^{-1})$
flops(II)	$\leq$	$56147\, h^{-2} + O(h^{-1})$	flops(VI)	$\leq$	$30258\, h^{-2} + O(h^{-1})$
flops(III)	$\leq$	$52273\, h^{-2} + O(h^{-1})$	flops(VII)	$\leq$	$32970\, h^{-2} + O(h^{-1})$
flops(IV)	$\leq$	$153284 h^{-2} + O(h^{-1})$	flops(VIII)	$\leq$	$164342\, h^{-2} + O(h^{-1})$

Since the number of collocation points $n + \kappa$ is bounded above and below by a constant multiple of h^{-2}, we conclude that the number of flops needed to evaluate s at Γ and to evaluate Δs at Ξ is bounded by a constant multiple of $n + \kappa$. In the following table, we display the actual cost of each basic evaluation task for the case $h = 1/32$, where $n + \kappa = 19807 \approx 19.3 h^{-2}$. In order to improve the efficiency, we have implemented method $\mathcal{M}_1$ using direct evaluation whenever it is expected to be more efficient than the method described in Corollary 4.11 (typically when $N << n$). We also report a normalized error,

$\|f - \mathbf{f}\|_{\ell_\infty(Z)} \,/\, \|f\|_{\ell_\infty(Z)}$, where the coefficients $\{\lambda_j\}$ and $\{\mu_j\}$ have been chosen randomly in $[-1, 1]$.

task	flops ($\times h^{-2}$)	normalized error
I	3245	7.77×10^{-15}
II	11990	6.02×10^{-15}
III	14253	3.63×10^{-14}
IV	60402	1.08×10^{-13}
V	48613	8.52×10^{-14}
VI	20742	7.76×10^{-15}
VII	23467	3.18×10^{-15}
VIII	19331	1.94×10^{-14}

Although solving the collocation equations is beyond the scope of the present contribution, we mention that these equations are notoriously ill-conditioned and are usually solved using a pre-conditioned iterative method. The author employs quad-precision along with a domain-decomposition preconditioner which is similar to that suggested in [2]. Essentially this amounts to a multilevel implementation of Von Neumann's method of alternating projections, along with a GMRES-like subspace development on the outer-most level. The following table details the experimental results for the problem

$$\Delta u = 0 \text{ in } \Omega; \qquad u = g \text{ on } \partial\Omega,$$

$$\text{where } g(x,y) = \Re\left(\sin \frac{4z^2}{1+z}\right) \text{ (with } z = x + \imath y).$$

h	# col pts	# iter.	$\|u-g\|_{L_\infty(\partial\Omega)}$	$\|\Delta u\|_{L_2(\Omega)}$	$\|u-g\|_{L_\infty(\Omega)}$
1/4	1325	8	3.4×10^{-10}	27256	92.804
1/8	5077	18	2.4×10^{-11}	3953.6	6.8975
1/32	19807	37	1.3×10^{-11}	261.6	0.46348

The exact solution is of course $u = g$; incidentally, $\|g\|_{L_\infty(\Omega)} \approx 6853$ and $\|g_{xx}\|_{L_2(\Omega)} \approx 48767$. We remark that each iteration requires $O(N)$ flops, where $N \sim h^{-2}$ denotes the total number of collocation points. It appears from this trial that the required number of iterations for convergence is roughly $O(h^{-1}) = O(\sqrt{N})$, and thus it appears that the collocation equations are solved using $O(N^{3/2})$ flops.

References

1. R.K. Beatson & W.A. Light, *Fast evaluation of radial basis functions; Methods for two-dimensional polyharmonic splines*, IMA J. Numer. Anal. **17** (1997), 343–372.
2. C. de Boor, *A practical guide to splines*, Applied Mathematical Sciences **27**, Springer-Verlag, New York, 1978.
3. M.D. Buhmann, *New developments in the theory of radial basis function interpolation*, Multivariate Approximation: From CAGD to Wavelets (K. Jetter, F.I. Utreras, eds.), World Scientific, Singapore, 1993, pp. 35–75.
4. J.B. Cherrie, R.K. Beatson & G.N. Newsam, *Fast evaluation of radial basis functions: Methods for generalized multiquadrics in* $\mathbb{R}^n$, SIAM J. Sci. Comput. **23** (2002), 1549–1571.
5. G. Fasshauer, *Solving Partial Differential Equations by Collocation with Radial Basis Functions*, Surface Fitting and Multiresolution Methods (A. Le Mehaute, C. Rabut, and L. L. Schumaker, eds.), Vanderbilt University Press, 1997, pp. 131-138.
6. C. Franke & R. Schaback, *Convergence Order Estimates of Meshless Collocation Methods using Radial Basis Functions*, Advances in Computational Mathematics **8** (1998), 381–399.
7. S.H. Friedberg, A.J. Insel, & L.E. Spence, *Linear Algebra, 3rd ed.*, Prentice Hall, New Jersey, 1997.
8. M. Golomb & H.F. Weinberger, *Optimal approximation and error bounds*, On numerical approximation (R.E. Langer, ed.), Univ. Wisconsin Press, Madison, Wisconsin, 1959, pp. 117–190.
9. 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)** (1990), 127–145.
10. W. Light and H. Wayne, *Spaces of distributions, interpolation by translates of a basis function and error estimates*, Numer. Math. **81** (1999), 415–450.
11. M.J.D. Powell, *The theory of radial basis function approximation in 1990*, Advances in Numerical Analysis II: Wavelets, Subdivision, and Radial Functions (W.A. Light, ed.), Oxford University Press, Oxford, 1992, pp. 105–210.
12. M.J.D. Powell, *Truncated Laurent expansions for the fast evaluation of thin plate splines*, Numer. Algorithms **5** (1993), 99-120.
13. W. Rudin, *Functional Analysis*, McGraw-Hill, New York, 1973.
14. R. Schaback & H. Wendland, *Using compactly supported radial basis functions to solve partial differential equations*, Boundary Element Technology XIII (C.S. Chen, C.A. Brebbia and D.W. Pepper, eds.), WitPress, Southampton, Boston, 1999, pp. 311–324.
15. H. Wendland, *Error estimates for interpolation by radial basis functions of minimal degree*, J. Approx. Th. **93** (1998), 258–272.