WendlandXool: Simplified C++ code to compute Wendland functions

Carlos Argáez[1], Peter Giesl[2], and Sigurdur Freyr Hafstein[3]

[1] Science Institute, University of Iceland, Tæknigarður, Dunhagi 5, 107 Reykjavík, Iceland, carlos@hi.is

[2] Department of Mathematics, University of Sussex, Falmer, BN1 9QH, UK, p.a.giesl@sussex.ac.uk

[3] Science Institute, University of Iceland, Tæknigarður, Dunhagi 5, 107 Reykjavík, Iceland, shafstein.is

Abstract. Radial basis functions are certain real-valued functions that depend only on the distance of their argument to a fixed point, i.e. $\mathbf{x} \mapsto \Psi(\|\mathbf{x} - \mathbf{x}_0\|)$. Among them, one can find Gaussians, multiquadrics and, the subject of this paper, Wendland functions that are compactly supported and positive definite functions [20], constructed as polynomials on their compact support. They find a great amount of applications, in particular, in algorithms to construct complete Lyapunov functions. In this paper, we present a new code to construct Wendland functions of any order, as well as their derived auxiliary functions used for numerically solving PDEs. This new code simplifies the structure and the ease of use compared to the code presented in [4]. Further, it optimises the routines to evaluate the functions and presents a new feature: a compilable LaTeX report with all the instructions and steps to construct them.

Keywords: Wendland functions, C++, scientific computing, applied mathematics, interpolating functions, compactly supported functions

1 INTRODUCTION

Wendland functions are compactly supported Radial Basis Functions (RBFs) widely used in (generalised) interpolation problems, including solving linear Partial Differential Equations (PDEs).

Let N points $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{R}^n$ (collocation points), $\mathbf{x}_i \neq \mathbf{x}_j$ if $i \neq j$, and associated values $f_1, \dots, f_N \in \mathbb{R}$ be given. A classical interpolation problem consists of finding a function $f\colon \mathbb{R}^n \to \mathbb{R}$ satisfying $f(\mathbf{x}_j) = f_j$ for all $j = 1, \dots, N$. In a generalised interpolation problem one prescribes the values of linear functionals applied to the function f. By prescribing the values of a linear differential operator, generalised interpolation problems can be used to numerically solve linear PDEs.

RBFs as kernels for Reproducing Kernel Hilbert Spaces (RKHS) provide a mighty machinery for solving interpolation and generalised interpolation problems in arbitrary dimensions [8,10,18]. Further, the error between the numerically

computed interpolants and the true solution can be bounded above in terms of the fill distance, which is the proper measure of the density of the collocation points.

Another advantage of using RBFs and generalised interpolation problems for solving PDEs is that the collocation points do not have to be evenly distributed and no triangulation of the phase-space is needed. Such methods have been used in numerous different settings in areas as different as geography, image processing, various engineering applications, numerical integration [9], machine learning and neural networks, cf. e.g. [9,22]. The authors are mainly interested in RBFs methods for numerically solving PDEs [11], in particular in solving Zubov-like PDEs for the computation of (complete) Lyapunov functions in dynamical systems [12,1,2,3].

A (generalised) interpolation problem aims to select a function in a given function space, typically a RKHS, that optimally fulfills certain conditions. The Wendland functions are polynomials on their compact support and are well suited as kernels for RKHS. The structure of a particular Wendland function depends on two parameters and they can be defined in a recursive manner.

In this paper we present a major revision of our first open-source code to compute Wendland functions [4] and give a new simplified algorithm implemented in C++ to explicitly compute any Wendland function. We have mentioned before in [4] the existence of different code, written in MAPLE [24,17] as well as the C++ libraries libMesh [13] and FOAM-FSI [7] that can be used to evaluate and provide Wendland functions. Further, there is a code written in R that can be used to evaluate Wendland functions for given radial values [14]. However, these tools are limited to a predefined selection of Wendland functions or to the use of commercial computational packages. In [6] a program written in Python was presented, that generates C++ header files and code for Wendland functions, that can be included in C++ projects. In our code the use is somewhat simpler using classes and the evaluation of the Wendland functions and their auxiliary functions uses the factorised form, that was shown in [5] to give much more exact results than when they are not factorised.

We present a C++ code that includes all necessary operations to compute any Wendland function and uses the *Armadillo C++ library* for linear algebra and scientific computing [15,16], which is also distributed free of charge.

Further, as a new tool, our code produces a LaTeX report in which all the operations to construct a Wendland function and auxiliary functions are summarised. This tool was introduced because the computation of Wendland functions and their auxiliaries requires the repeated application of several operations, that can be error prone in practice:

- Polynomial integration,
- Polynomial derivation,
- Polynomial factorisation.

Such a report provides a useful debugging tool and delivers the constructed Wendland functions and auxiliary functions in LaTeX format, both factorised and expanded.

The new algorithm provides the coefficients of the polynomials of the Wendland functions as integers, which is the form in which Wendland functions are usually presented [12]. When Wendland functions are used to solve generalised interpolation problems, they appear linearly on both sides of linear equations and therefore they are essentially only defined up to a non-zero multiplicative constant. If the integer coefficients cannot be represented by built in integer types then the program reports this. In fact, for practical computations Wendland functions are commonly used for quite low parameters l, k, because otherwise the condition number of the collocation matrix is often very large.

2 GENERALISED INTERPOLATION USING WENDLAND FUNCTIONS

In this section we sum up the most important aspects of generalised interpolation using Wendland functions. Consider a Hilbert space $H \subset C(\mathbb{R}^n, \mathbb{R})$ of continuous functions $f : \mathbb{R}^n \to \mathbb{R}$ and its dual H^*, i.e. the set of all linear and continuous functionals $\lambda : H \to \mathbb{R}$. In a RKHS the point evaluation functional $\delta_{\mathbf{x}_0}(f) = f(\mathbf{x}_0)$, evaluating the function at a point $\mathbf{x}_0 \in \mathbb{R}^n$, is in H^*. Given more regularity of the functions in H, differential operators evaluated at a point, e.g., $\lambda = \delta_{\mathbf{x}_0} \circ \frac{\partial}{\partial x_j}$, $j \in \{1, \ldots, n\}$, are also in H^*.

Definition 1 (Generalised interpolation problem). *Given N linearly independent functionals $\lambda_1, \ldots, \lambda_N \in H^*$ and corresponding values $f_1, \ldots, f_N \in \mathbb{R}$, a generalised interpolant $f \in H$ satisfies $\lambda_j(f) = f_j$ for all $j = 1, \ldots, N$.*

Note that the classical interpolation problem is a special case with $\lambda_j = \delta_{\mathbf{x}_j}$.

A norm-minimal generalised interpolant is an interpolant that is minimal in the norm of the Hilbert space H, i.e.

$$\arg\min_{f \in H} \{\|f\|_H : \lambda_j(f) = f_j, 1 \le j \le N\}.$$

The norm-minimal interpolant is unique and can be written as a linear combination of the Riesz representers $v_j \in H$ of the functionals, cf. e.g. [22], and if H is a RKHS the Riesz representers have a simple formula.

Recall that a RKHS is a Hilbert space H with a reproducing kernel $\Phi : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ such that

1. $\Phi(\cdot, \mathbf{x}) \in H$ for all $\mathbf{x} \in \mathbb{R}^n$
2. $g(\mathbf{x}) = \langle g, \Phi(\cdot, \mathbf{x}) \rangle_H$ for all $g \in H$ and $\mathbf{x} \in \mathbb{R}^n$

Here $\langle \cdot, \cdot \rangle_H$ is the inner product of H.

The Riesz representer of $\lambda_j \in H^*$ has the formula $v_j = \lambda_j^{\mathbf{y}} \Phi(\cdot, \mathbf{y})$, i.e. λ_j applied to $\mathbf{x} \to \Phi(\mathbf{x}, \mathbf{y}) \in H$. Thus we can write the norm-minimal interpolant as $f(\mathbf{x}) = \sum_{j=1}^{N} \beta_j \lambda_j^{\mathbf{y}} \Phi(\mathbf{x}, \mathbf{y})$, where the interpolation conditions $\lambda_j(f) = f_j$, $1 \le j \le N$, are used to fix the coefficients β_j.

RBF kernels are such that $\Phi(\mathbf{x},\mathbf{y}) := \Psi(\|\mathbf{x}-\mathbf{y}\|)$ for a function $\Psi : [0,\infty) \to [0,\infty)$. There are numerous RBFs that can serve as kernels for RKHS and different RBFs will lead to different RKHSs.

The so-called Wendland functions [23,19,20,21,24] have a compact support and are polynomials on their support. The corresponding RKHSs are norm-equivalent to Sobolev spaces, which together with the simple form of the Wendland functions make them well suited as kernels to solve linear PDEs.

The Wendland functions $\mathbf{\Psi}^0_{l,k}$, where $l \in \mathbb{N}$ and $k \in \mathbb{N}_0$, can be defined recursively as follows, cf. [20].

Definition 2 (Wendland function). *The Wendland function $\mathbf{\Psi}^0_{l,k}$, where $l \in \mathbb{N}$ and $k \in \mathbb{N}_0$, is defined recursively by*

$$\begin{aligned} \mathbf{\Psi}^0_{l,0}(r) &= (1-r)^l_+ \\ \mathbf{\Psi}^0_{l,j+1}(r) &= \int_r^1 t\mathbf{\Psi}^0_{l,j}(t)\,dt \text{ for } j = 0,1,\ldots,k-1, \end{aligned} \tag{1}$$

where $x_+ = \max\{x,0\}$ and $x^l_+ := (x_+)^l$.

Hence

$$\mathbf{\Psi}^0_{l,k}(r) = \underbrace{\int_r^1 t_k \int_{t_k}^1 t_{k-1} \cdots \int_{t_2}^1}_{k \text{ integrations}} t_1 \mathbf{\Psi}^0_{l,0}(t_1) \overbrace{\mathrm{d}t_1 \cdots \mathrm{d}t_k}^{k \text{ differentials}} . \tag{2}$$

Note that the support of the Wendland function $\mathbf{\Psi}^0_{l,k}(r)$ is the interval $[0,1]$ and thus, for a constant $c > 0$ the support of the function $\mathbf{x} \mapsto \mathbf{\Psi}^0_{l,k}(c\|\mathbf{x}-\mathbf{x}_0\|)$ is a ball of radius c^{-1}, centered at $\mathbf{x}_0$. In applications for dynamical systems we deal with certain differential operators that require the following auxiliary functions, derived from the Wendland functions. They are also the reason, why it is advantageous to include a constant $c > 0$ in the definition of a Wendland function, i.e. consider the function $r \mapsto \mathbf{\Psi}^0_{l,k}(cr)$ rather than $\mathbf{\Psi}^0_{l,k}(r)$.

Definition 3 (Auxiliary functions $\mathbf{\Psi}^1_{l,k}$ and $\mathbf{\Psi}^2_{l,k}$). *For a fixed $c > 0$ and a given Wendland function $r \mapsto \mathbf{\Psi}^0_{l,k}(cr)$ the auxiliary functions $r \mapsto \mathbf{\Psi}^1_{l,k}(cr)$ and $r \mapsto \mathbf{\Psi}^2_{l,k}(cr)$ are defined as follows for $r > 0$,*

$$\mathbf{\Psi}^1_{l,k}(cr) = r^{-1}\frac{d}{dr}\mathbf{\Psi}^0_{l,k}(cr) \quad \text{and} \quad \mathbf{\Psi}^2_{l,k}(cr) = r^{-1}\frac{d}{dr}\mathbf{\Psi}^1_{l,k}(cr).$$

Remark 1. Auxiliary functions of higher order can be defined equivalently. The presented code can easily be adapted to compute them as well.

Remark 2. In case $\mathbf{\Psi}^j_{l,k}(cr)$ can be continuously extended to $r = 0$, this is done and the function is also defined for $r = 0$.

3 ALGORITHM

As seen in (1), the parameters that define the polynomial's degree are k and l, where l denotes the degree of the initial polynomial and k is the number of iterative integrations.

An outline of the algorithm is as follows:

1. Compute the binomial expansion of $(1 - t_1)^l$. We do this by computing Pascal's triangle and expand the polynomial accordingly. The coefficients are stored in an array, starting with the coefficient for $t_1^0 := 1$ (left), then the coefficient for t_1^1, etc.
2. After this, the iterative procedure starts. Remember that, according to (2), k is the number of integrations that will be performed. The operations performed by the algorithm are to multiply a polynomial with the free variable and to integrate, cf. (1). Both correspond to simple manipulations on the vector of coefficients of the polynomial.
   (a) Multiplying with the free variable corresponds to shifting all elements to the right and put zero in the first position. After multiplying by t_j an integration is performed.
   (b) Integration: We compute the following integral, with f as the polynomial from the last step, cf. (2).

$$\int_{t_{j+1}}^{1} f(t_j)\mathrm{d}t_j = F(1) - F(t_{j+1}). \tag{3}$$

   Again this corresponds to shifting all elements to the right, but now the coefficient a_s of t_{j+1}^s in $F(t_{j+1})$ is multiplied with $1/s$. The first position, the coefficient for $t_{j+1}^0 := 1$, becomes the sum of the other coefficients. All other coefficients must then be multiplied by -1, corresponding to $F(1) - F(t_{j+1})$. Notice that the new coefficients will in general not be integer numbers.
3. Once the for-loop ends at k, the remaining step is to compute the factor to convert all coefficients to integers. For this the algorithm keeps track of the least common denominator `lcd` and the greatest common factor `hcf` of the coefficients of the polynomials. If no overflow occurs the coefficients are multiplied by the factor `round(lcd/hcf)`, which makes all coefficients integers. If there is an overflow the execution is stopped and the user is warned.
4. The final polynomial $\mathbf{\Psi}_{l,k}^0(cr)$ is of order $l+1+2\cdot k$ in $x = cr$ and is used to compute the auxiliary functions $\Psi_{l,k}^1(cr)$ and $\Psi_{l,k}^2(cr)$ as in Definition 3. For the efficient implementation, it is of use to note that in general for $j > 0$ the function $\Psi_{l,k}^j(cr)/c^{2j}$ can be written as a rational function of $x = cr$, with the polynomial x^u, $u \in \mathbb{N}_0$, in the denominator. That is

$$\Psi_{l,k}^j(cr) = \sum_{i=-u}^{v} a_i(cr)^i c^{2j} = (cr)^{-u} c^{2j} \sum_{i=0}^{v+u} a_i(cr)^i$$

and the transformation from $\Psi^j_{l,k}(cr)$ to $\Psi^{j+1}_{l,k}(cr)$ is, once again, essentially just a simple manipulation of a vector of coefficients.

5. Finally, $\Psi^j_{l,k}(cr)$, $j = 0, 1, 2$, are factorised in the form

$$\mathbf{\Psi}^j_{l,k}(cr) = c^{2j}(1-cr)^s \frac{f_t(cr)}{(cr)^u},$$

where $f_t(cr) = \sum_{i=0}^{t} b_i (cr)^i$ is a polynomial in $x = cr$ of maximal degree. For this we use the fact that the coefficients in the polynomial, or polynomial numerator, of $\mathbf{\Psi}^j_{l,k}$ has integer coefficients and thus, iterated polynomial synthetic division gives exact results and the polynomial $f_t(cr)$ also has integer coefficients.

6. Along the process the report in LaTeX is written.

Remark 3. Recall that when Wendland functions are used in generalised interpolation, they can be multiplied with a non-zero constant because they appear on both sides of linear equation. Therefore, they are most commonly presented with integer coefficients as in [12].

3.1 EVALUATION

To evaluate the functions we use Horner's scheme. A few comments are in order: The evaluation routine is the same for all the functions, the Wendland function and all its auxiliary functions. In general we evaluate for $u, v \in \mathbb{N}_0$ and for $x = cr$, $0 < x < 1$, the expression

$$\Psi^j_{l,k}(x) = \sum_{k=-u}^{v} a_k(x)^k c^{2j} = c^{2j}(1-x)^s \frac{f_t(x)}{x^u} = c^{2j}(1-x)^s x^{-u} \sum_{i=0}^{t} b_i x^i$$

using Horner's scheme for the polynomial term $\sum_{i=0}^{t} b_i x^i$. For $x \geq 1$ the routine returns zero as expected. If $u = 0$ the function can be continuously extended to $x = 0$ and is evaluated as above, cf. e.g. Proposition 3.5 in [12].

4 HOW TO USE / EXAMPLES

The program is found in https://github.com/LyapXool/WendlandXool-V2 and consists of the files `wendland.cpp` and `wendland.hpp`.
Further, the file `WendlandExample.cpp` included, contains an example of its use. The class `WendRBF` delivers the interface to the Wendland functions to the user. To construct the Wendland function $\mathbf{\Psi}^0_{l,k}(cr)$, for some constants $l \in \mathbb{N}$, $k \in \mathbb{N}_0$, and $c > 0$, together with its auxiliaries $\mathbf{\Psi}^1_{l,k}(cr)$ and $\mathbf{\Psi}^2_{l,k}(cr)$, the user can simply make it as an object using the constructor with the corresponding parameters:

```
WendRBF(int l, int k, double c, bool printreport)
```

If `printreport=true` then a detailed LaTeX compilable report on the construction of the Wendland function and its auxiliaries is written in `wendlandreport.tex`. The default value is `false` and no report is written. After its initialisation/creation the user can evaluate $\mathbf{\Psi}^0_{l,k}(cr)$ at any $r \geq 0$ using

```
double WendRBF::operator()(double r)
```

Note that the argument should be r, not $x = cr$. To evaluate $\mathbf{\Psi}^1_{l,k}(cr)$ at r one uses the member function `double WendRBF::aux1(double r)` and to evaluate $\mathbf{\Psi}^2_{l,k}(cr)$ the member function `double WendRBF::aux2(double r)`. The routines that do the actual computations are in `namespace wendland`. An excerpt from the report when `WendRBF psi31(3,1,1.0,true)` is called follows:

Construction and all steps for the construction of the Wendland function. Wendland function $\Psi^0_{3,1}$

First binomial $(1 - t_1)^3$

$t_1^0 - 3t_1^1 + 3t_1^2 - t_1^3$
Multiplying by t_1:

$0t_1^0 + t_1^1 - 3t_1^2 + 3t_1^3 - t_1^4$
Computing integration: 1
Integrating from r **to** 1:

$5 \times 10^{-2} r^0 + 0r^1 - 5 \times 10^{-1} r^2 + r^3 - 7.5 \times 10^{-1} r^4 + 2 \times 10^{-1} r^5$
Using the factor 20 **the Wendland function becomes for** $0 \leq cr \leq 1$:

$\Psi^0_{3,1}(cr) = 1 + 0c^1 r^1 - 10c^2 r^2 + 20c^3 r^3 - 15c^4 r^4 + 4c^5 r^5$

Construction and all steps for the construction of the auxiliar function.

Order: 1 Wendland function derivative $\Psi^1_{3,1}$.
Derive $\Psi^0_{3,1}(cr)$ by r and divide the result by r:

$\Psi^1_{3,1}(cr) = -20c^2 + 60c^3 r^1 - 60c^4 r^2 + 20c^5 r^3$
For $0 < cr \leq 1$.

Construction and all steps for the construction of the auxiliar function.

Order: 2 Wendland function derivative $\Psi^2_{3,1}$.
Derive $\Psi^1_{3,1}(cr)$ by r and divide the result by r:

$$\Psi^2_{3,1}(cr) = 60c^3r^{-1} - 120c^4r^0 + 60c^5r^1$$

For $0 < cr \leq 1$.

Functions presented in a factorised form.

Next, we present the factorised version of the Wendland function.

$$\Psi^0_{3,1}(cr) = (1-cr)^4_+ \left(1 + 4c^1r^1\right)$$

$$\Psi^1_{3,1}(cr) = (1-cr)^3_+ c^2 (-20)$$

$$\Psi^2_{3,1}(cr) = (1-cr)^2_+ c^3 \left(60r^{-1}\right)$$

For a graphical presentation of the function $\Psi^0_{3,1}(cr)$, $c = 1$, and its two auxiliary functions $\Psi^1_{3,1}(cr)$ and $\Psi^2_{3,1}(cr)$ can be seen in Figure 1. For a table with the coefficients of the polynomials see Table 1.

	Wendland function $\Psi^0_{3,1}(cr)$						Function $\Psi^1_{3,1}(r)$				Function $\Psi^2_{3,1}(r)$		
Exponent k of r^k	0	1	2	3	4	5	0	1	2	3	-1	0	1
Coefficient a_k of $a_k r^k$	1	0	-10	20	-15	4	-20	60	-60	20	60	-120	60

Table 1. Arrays presenting the results and their storage.

5 CONCLUSIONS

We have upgraded our previous contribution [4] to compute Wendland's compactly supported Radial Basis Functions. The new code is more user friendly, delivering the functionality though a class. Additionally, the evaluation of the

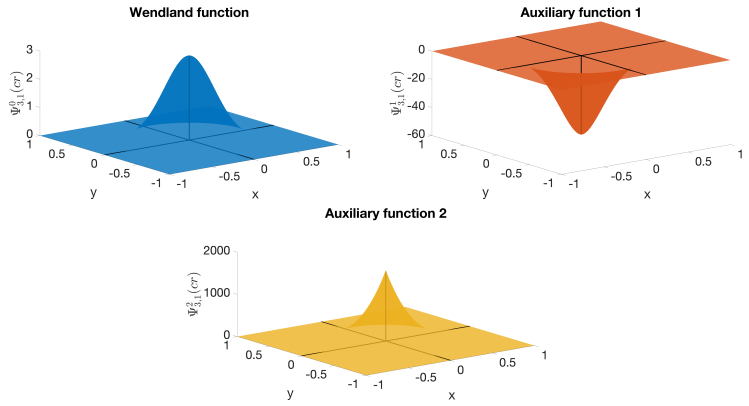

Fig. 1. Upper left: Wendland function $\Psi^0_{3,1}(\|(x,y)\|)$. Upper right: Auxiliary function 1. Lower: Auxiliary function 2.

Wendland functions and their auxiliaries has been optimised for numerical accuracy and, optionally, a detailed LaTeX compilable report on the generation of the Wendland functions and their auxiliaries can be generated.

References

1. C. Argáez, P. Giesl, and S. Hafstein. Analysing dynamical systems towards computing complete Lyapunov functions. In *Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications, Madrid, Spain*, pages 323–330, 2017.
2. C. Argáez, P. Giesl, and S. Hafstein. Computation of complete Lyapunov functions for three-dimensional systems. In *Proceedings of the 57rd IEEE Conference on Decision and Control (CDC)*, pages 4059–4064, 2018.
3. C. Argáez, P. Giesl, and S. Hafstein. *Computational approach for complete Lyapunov functions. In* Dynamical Systems in Theoretical Perspective. Springer Proceedings in Mathematics & Statistics. ed. Awrejcewicz J. (eds)*., volume 248, pages 1–11, 2018.*
4. *S. Argáez, C. Hafstein and P. Giesl. Wendland functions - a C++ code to compute them. In* Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, pages 323–330, 2017.*
5. *H. Bjornsson and S. Hafstein. Verification of a numerical solution to a collocation problem. In* Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pages 587–594, 2018.*
6. *H. Bjornsson and S. Hafstein. Algorithm and software to generate code for wendland functions in factorized form. In* Proceedings of the 16th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pages 156–162, 2019.*
7. *D. Blom, P. Cardiff, T. Gillebaart, E. t. Hofstede, and V. Kazemi-Kamyab. FOAM-FSI project. Github.*

8. *M. D. Buhmann.* Radial basis functions: theory and implementations*, volume 12 of* Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2003.*
9. *J. Dick, F. Kuo, and I. Sloan. High-dimensional integration: the quasi-Monte Carlo way.* Acta Numer.*, 22:133–288, 2013.*
10. *G. Fasshauer.* Meshfree approximation methods with MATLAB*. Number 6 in Interdisciplinary Mathematical Sciences. World Scientific Publishing, 2007.*
11. *B. Fornberg and N. Flyer. Solving PDEs with radial basis functions.* Acta Numer.*, 24:215–258, 2015.*
12. *P. Giesl.* Construction of Global Lyapunov Functions Using Radial Basis Functions*. Lecture Notes in Math. 1904, Springer, 2007.*
13. *B. Kirk, J. Peterson, R. Stogner, and G. Carey.* `libMesh`*: A C++ library for parallel adaptive mesh refinement/coarsening simulations.* Eng. Comput.*, 22(3-4):237–254, 2006.*
14. *D. Nychka. Wendland family of covariance functions and supporting numerical functions. Technical report. https://www.rdocumentation.org/packages/fields/versions/11.6/topics/Wendland.*
15. *C. Sanderson and R. Curtin. Armadillo: a template-based C++ library for linear algebra.* J. Open Source Softw*, 1(2):26, 2016.*
16. *C. Sanderson and R. Curtin.* Mathematical Software - ICMS 2018*, volume 10931 of* LNCS*, chapter A User-Friendly Hybrid Sparse Matrix Class in C++, pages 422–430. Springer, 2018.*
17. *R. Schaback. The missing Wendland functions.* Adv. Comput. Math.*, (34):67–81, 2011.*
18. *R. Schaback and H. Wendland. Kernel techniques: from machine learning to meshless methods.* Acta Numer.*, 15:543–639, 2006.*
19. *R. Schaback and Z. Wu. Operators on radial functions.* J. Comput. Appl. Math.*, 73(1-2):257–270, 1996.*
20. *H. Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree.* Adv. Comput. Math.*, 4(4):389–396, 1995.*
21. *H. Wendland. Error estimates for interpolation by compactly supported Radial Basis Functions of minimal degree.* J. Approx. Theory*, 93:258–272, 1998.*
22. *H. Wendland.* Scattered data approximation*, volume 17 of* Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2005.*
23. *Z. Wu. Compactly supported positive definite radial functions.* Adv. Comput. Math.*, 4(3):283–292, 1995.*
24. *S. Zhu. Compactly supported radial basis functions: how and why? OCCAM Preprint Number 12/57, University of Oxford, 2012.*