Algorithm and Software to Generate Code for Wendland Functions in Factorized Form

Hjortur Bjornsson and Sigurdur Hafstein

Science Institute, University of Iceland, Dunhagi 3, 107 Reykjavík, Iceland

Keywords: Wendland Function, Lyapunov Functions, Radial Basis Functions, Code Generation.

Abstract: In this paper we describe an algorithm to determine Wendland’s Radial Basis Functions in a specific factorized form. Additionally, we present a software tool that uses this algorithm to generate a C/C++ library that implements the Wendland functions with arbitrary parameters in factorized form. This library is more efficient and has higher numerical accuracy than previous implementations. The software tool is written in Python and is available for download.

1 INTRODUCTION

Interpolation and collocation using Radial Basis Functions (RBF), in particular compactly supported RBFs, has been the subject of numerous research activities in the past decades (Wu, 1992; Floater and Iske, 1996; Franke and Schaback, 1998; Wendland, 1998; Buhmann, 2003; Buhmann, 2000; Wendland, 2005; Wendland, 2017). They are well suited as kernels of Reproducing Kernel Hilbert Spaces and their mathematical theory is mature. The authors and their collaborators have applied Wendland’s compactly supported RBFs for computing Lyapunov functions for nonlinear systems, both deterministic (Giesl, 2007; Giesl, 2008; Giesl and Hafstein, 2015) and stochastic (Bjornsson et al., 2019). Lyapunov functions are useful tools to analyse stability of various dynamical systems, either deterministic or stochastic, e.g. (Khalil, 2002; Sastry, 1999; Vidyasagar, 2002; Khasminskii, 2012; Mao, 2008). Various numerical methods have been used to find Lyapunov functions for the systems at hand (Giesl and Hafstein, 2015; Hafstein et al., 2018). Meshless collocation using RBFs is one such method and many different families of RBFs have been studied (Wendland, 2005).

In the papers (Giesl and Hafstein, 2015; Bjornsson et al., 2019) meshless collocation with so called Wendland functions is used. The Wendland functions are compactly supported radial functions, that are polynomials on their support. The Wendland function family is defined in such a way that many tedious and error prone calculations have to be done by hand in order to obtain formulas for the functions to be used in the software implementation in (Giesl and Hafstein, 2015; Bjornsson et al., 2019). In (Argaez et al., 2017) an algorithm is proposed that determines the Wendland polynomials in expanded form, that is: for each integer \( l, k \geq 0 \) finds a list of numbers \( a_0, a_1, \ldots, a_d \) such that the Wendland function \( \psi_{l,k}(r) = \sum_{i=0}^{d} a_i r^i \) on its compact support. However, it was shown in (Bjornsson and Hafstein, 2018) that the evaluations of these polynomials in this form using typical schemes, such as Horner’s scheme, can lead to significant numerical errors.

Evaluating the Wendland functions in factorized form (Bjornsson and Hafstein, 2018) is more efficient and numerically accurate, so we propose an alternate algorithm to determine the functions in factorized form. In addition to developing the algorithm, we implemented it and created a software tool that generates a reusable software library, which implements these Wendland polynomials in factorized form in C/C++.

2 BACKGROUND

In the paper (Bjornsson et al., 2019), meshless collocation using RBFs was used to calculate Lyapunov functions for various Stochastic Differential Equations (SDE). This included computing, for a given domain \( \Omega \subset \mathbb{R}^d \) and a boundary \( \Gamma \subset \mathbb{R}^d \), a solution to the Partial Differential Equation (PDE)
where $L$ is a certain second-order differential operator, and $h$ and $c$ are appropriately chosen functions. A numerical solution to the above problem was determined by choosing points $X_1 = \{x_1, \ldots, x_N\} \subset \Omega$ and $X_2 = \{\xi_1, \ldots, \xi_M\} \subset \Gamma$ and solving the interpolation problem

$$\begin{cases}
LV(x_i) = h(x_i) & x_i \in \Omega \\
V(x_i) = c(x_i) & x_i \in \Gamma,
\end{cases}$$

(1)

and

$$\psi_{l,k+1}(r) = C_{l,k+1} \int_r^1 r \psi_{l,k}(t) \, dt,$$

(4)

where $(1-r)_+ := \max\{1-r,0\}$ and $C_{l,k+1} \neq 0$ is a constant. For interpolation and collocation using a particular Wendland function, the value of the constant $C_{l,k+1}$ is not of importance because the Wendland function appears linearly on both sides of a linear equation. Therefore, one can just fix values that are convenient for the problem at hand and we will do this in the following section. These functions satisfy the relation

$$-C_{l,k+1} \psi_{l,k}(r) = \frac{d}{dr} \psi_{l,k+1}(r).$$

(5)

It is not difficult to verify that the functions

$$\Phi_{l,0}(r) = [(1-r)_+]^l$$

(6)

and (6) also satisfy a relation of the form

$$-2(k-1) \Phi_{l,k}(r) = \frac{d}{dr} \Phi_{l,k+1}(r),$$

for all integers $k,l \geq 0$, i.e. a relation identical to equation (5) with $C_{l,k+1} = 2(k-1)$. Just note that

$$\frac{d}{dr} \Phi_{l,0}(r) (t^2 - r^2)^{k-1} dr = -2r(k-1) \int_r^1 \Phi_{l,0}(t)(t^2 - r^2)^{k-2} dt.$$

Therefore (7) delivers an alternative way to define the Wendland functions, see (Wendland, 2017). Note that (Wendland, 2017) uses a different numbering scheme of the functions than we do.

The Wendland functions have several important properties, c.f. e.g. (Giesl, 2007, Prop. 3.10):

1. $\psi_{l,k}(r)$ is a polynomial of degree $l + 2k$ for $r \in [0,1]$ and $\text{supp} (\psi_{l,k}) = [0,1]$.
2. The radial function $\Psi(x) := \psi_{l,k}(||x||)$ is $C^{2k}$ at 0.
3. $\psi_{l,k}$ is $C^{k+l-1}$ at 1.

Frequently we fix the parameter $l := \left\lfloor \frac{n}{2} \right\rfloor + k + 1$, where $n$ is the space dimension we are working in, and a constant $c > 0$ to fix the support. By the properties stated above, the radial function $\Psi(x) := \psi_{l,k}(||x||)$ is then a $C^{2k}$ function with $\text{supp} (\Psi) = B^d(0,c^{-1}) \subset \mathbb{R}^n$, where $B^n(0,c^{-1})$ is the closed $n$-dimensional ball around the origin with radius $c^{-1}$.

3 WENDLAND FUNCTIONS

The Wendland functions are a family of functions, depending on two parameters $l,k \in \mathbb{N}_0$ defined by

$$\psi_{l,0}(r) = [(1-r)_+]^l$$

(3)
4 ALGORITHM

We represent $d$-degree polynomials $\sum_{i=0}^{d} a_i t^i$ as a list of coefficients $(a_0, a_1, \ldots, a_d)$. Our implementation uses Python with List objects. Addition and multiplication of polynomials of this form are easily implemented as:

$$
\sum_{i=0}^{d_1} a_i t^i + \sum_{j=0}^{d_2} b_j t^j = \sum_{i=0}^{\text{max}(d_1,d_2)} (a_i + b_i) t^i,
$$

where $a_i = 0$ for $i > d_1$ and $b_j = 0$ for $j > d_2$. Multiplication is given by

$$
\left( \sum_{i=0}^{d_1} a_i t^i \right) \left( \sum_{j=0}^{d_2} b_j t^j \right) = \sum_{i=0}^{d_1+d_2} c_i t^i,
$$

where

$$
c_i = \sum_{k+i=j} a_k b_j.
$$

An antiderivative of a polynomial is given by

$$(a_0, a_1, a_2, \ldots, a_d) \mapsto (0, a_0, \frac{a_1}{2}, \frac{a_2}{3}, \ldots, \frac{a_d}{d+1}),$$

corresponding to

$$\int \sum_{i=0}^{d} a_i t^i dt = \sum_{i=0}^{d} \frac{a_i}{i+1} t^{i+1},$$

and differentiation by

$$(a_0, a_1, \ldots, a_d) \mapsto (a_1, 2a_2, 3a_3, \ldots, da_d).$$

In order to maximize exact calculations up to computer limitations, we store the coefficients as tuples of Integers, numerator and denominator, avoiding the floating point approximation. Specifically, we used the Rational class provided in Python. We can represent polynomials in two variables as a polynomial in the first variable where each coefficient is a polynomial in the second variable (of which each coefficient is a rational number).

4.1 Construction

To calculate a polynomial representing the Wendland function $\psi_{l,k}$ on the interval $[0, 1]$ we start by fixing the derivative

$$p'(t) = (1-t)^l t (t^2 - r^2)^{k-1},$$

see (7), which we represent as a polynomial in $t$ with each coefficient a polynomial in $r$. We integrate with respect to $t$ and obtain a new polynomial $p(t)$ in $t$, again with coefficients that are polynomials in $r$. We evaluate the polynomial $p$ at $t = 1$ and at $t = r$, which in both cases result in a polynomial in $r$, and we obtain the polynomial $\psi(r) = p(1) - p(r)$. Note that $\psi(r) = C_1 \psi_{l,k}(r)$ for some constant $C_1 \neq 0$.

By using a long division algorithm we factor $\psi$ into the form

$$\psi(r) = C_2 (1-r)^{l+k} p_{l,k}(r)$$

such that $p_{l,k}(r)$ is a polynomial with integer coefficients with no common factors. This is possible since $\psi_{l,0}$ has a zero of order $l$ at 1, and by using the recursive relation in equation (4), we see that $\psi_{l,k}$ has a zero of order $l+k$ at 1. Since $\psi_{l,k}$ is essentially only defined up to a multiplicative non-zero constant, we discard the constant $C_2$ and use $\psi(r) = (1-r)^{l+k} p_{l,k}(r)$, a polynomial with integer coefficients, as a starting point for our recursion.

Using the relation in (5), ignoring the constant $C_{l,k}$, we see that

$$\psi_{l,k}^{-1}(r) = \frac{d}{dr} \left[ (1-r)^{l+k} p_{l,k}(r) \right]$$

$$= \frac{1}{r} (1-r)^{l+k-1} \left( (1-r) p'_{l,k}(r) - (l+k) p_{l,k}(r) \right).$$

Therefore we have

$$p_{l,k}^{-1}(r) \equiv \frac{\psi_{l,k}^{-1}(r)}{(1-r)^{l+k-1}}$$

$$= \frac{1}{r} \left[ (1-r) p'_{l,k}(r) - (l+k) p_{l,k}(r) \right].$$

We know that $\psi_{l,k}^{-1}$ is a polynomial, therefore

$$\frac{d}{dr} \left[ (1-r)^{l+k} p_{l,k}(r) \right]$$

must be divisible by the monomial $r$. Since $(1-r)^{l+k-1}$ is not divisible by $r$, the right-hand side of (10) must be a polynomial in $r$. Therefore $p_{l,k}^{-1}$ is a well defined polynomial.

By pulling out the common factor $b_{k-1} \in \mathbb{Z}$ of the coefficients in $p_{l,k}^{-1}$ we obtain a new polynomial $\tilde{p}_{l,k}^{-1}$ and a constant $b_{k-1}$ such that

$$p_{l,k}^{-1} = b_{k-1} \tilde{p}_{l,k}^{-1}.$$

Repeating this step, until we arrive at $p_{l,0}$, we get a collection of polynomials in the form

$$\psi_i(r) = b_1 \cdots b_i (1-r)^{l+k-i} \tilde{p}_{l,k-i}(r), \ i = 1, 2, \ldots, k$$

(11)

where each of the polynomials $\tilde{p}_{l,k-i}(r)$ has integer coefficients and each of the constants $b_i$ is a negative integer.

The above list follows the notation in (Giel, 2007) were $\psi_0$ is the polynomial given in (8) and is equal to the Wendland function $\psi_{l,k}$, and $\psi_1, \ldots, \psi_l$ are the Wendland functions given by $\psi_{l,k-1}, \ldots, \psi_{l,0}$ respectively, see equation (4). It is important to keep track of the constants $b_1, \ldots, b_l$ in (11) as they are necessary for correct evaluation of formula (2).
4.2 Example

To see how the algorithm works let us consider how it computes the Wendland function $\psi_{l,k}$ for $l = 5$ and $k = 4$. Here $p'(t) = (1 - t)^5 t^{(r^2 - r^2)^3}$ and we obtain

$$\psi(r) = \int_0^1 \frac{1}{1 - t} (1 - t)^5 t^{(r^2 - r^2)^3} dt$$

$$= -\frac{16}{3003} r^{13} + \frac{1}{24} r^{12} - \frac{32}{231} r^{11} + \frac{4}{4} r^{10} - \frac{16}{63} r^9 + \frac{1}{8} r^8 - \frac{1}{42} r^6 + \frac{1}{18} r^4 - \frac{1}{924} r^2 + \frac{1}{10296}$$

and set $\psi_0(r) = (1 - r)^9 (384 r^4 + 453 r^3 + 237 r^2 + 63 r + 7)$ and $\psi_1(r) = (1 - r)^9 (384 r^4 + 453 r^3 + 237 r^2 + 63 r + 7)$. For $r \in [0, 1]$ we have the formulas (recall that $\psi_{l,k}(r) = 0$ if $r \notin [0, 1]$):

$$\psi_{5,4}(r) = \psi_4(r)$$

$$= (1 - r)^9 (384 r^4 + 453 r^3 + 237 r^2 + 63 r + 7)$$

$$\psi_{5,3}(r) = \psi_3(r) \frac{d}{d \psi_0(r)}$$

$$= 156(1 - r)^8 (32 r^3 + 25 r^2 + 8 r + 1)$$

$$\psi_{5,2}(r) = \psi_2(r) \frac{d}{d \psi_0(r)}$$

$$= (1 - r)^7 (16 r^2 + 7 r + 1)$$

$$\psi_{5,1}(r) = \psi_1(r) \frac{d}{d \psi_0(r)}$$

$$= -82,368(1 - r)^6 (6 r + 1)$$

$$\psi_{5,0}(r) = \psi_0(r) \frac{d}{d \psi_0(r)}$$

$$= 3,459,456(1 - r)^5$$

Note that we have, indeed, computed a lot more useful information than just a family of Wendland functions $\psi_{5,i}, i = 0, 1, \ldots, 4$. In our algorithm, for a fixed $l, k$, we have

$$\psi_{l,k-j} = \psi_{j}(r) \frac{d}{d \psi_{j-1}(r)}$$

$$= \frac{d}{d \psi_{l,k-j+1}(r)}$$

and we have thus delivered all the radial basis functions needed for a collocation problem. This corresponds to computing a whole table as in (Giesl, 2007, Table 3.1), but for a collocation problem with arbitrary high derivatives. In the software tool, discussed in the next section, also the constant $c > 0$ used to fix the support of the Wendland function, is included in these computations.

5 SOFTWARE LIBRARY

We have implemented the above algorithm in a software tool that generates C/C++ code versions of the Wendland functions in factorized form. In a previous work (Bjornsson and Hafstein, 2018), we determined that the most efficient and accurate way to evaluate these Wendland functions was to use this factorized form. Evaluating these polynomials in fully expanded format using Horner’s scheme (Burrus et al., 2003), can lead to very large numerical errors as shown in (Bjornsson and Hafstein, 2018). Below is a part of the library generated by our tool, which shows the family of Wendland functions obtained when starting with $\Psi_0(x) = \psi_{5,4}(c|x|)$, where $c > 0$ is the constant that controls the support of the radial function $\Psi$.

```
1 double _wendlandpsi_5_4_0(double x, .... double c){
2 double t=_ipow((1.0-x), 9);
3 t=1.0+t*(((384)*x + 453)*x + ....
4 237)*x + 63)*x + 7);
5 return t;
6 }
7 double _wendlandpsi_5_4_1(double x, .... double c){
8 double t=_ipow((1.0-x), 8);
9 t=156.0+t*(_ipow(c, 2) *(((32)*x ....
10 + 25)*x + 8)*x + 1);
11 return t;
12 }
13 double _wendlandpsi_5_4_2(double x, .... double c){
14 double t=_ipow((1.0-x), 7);
15 t=3432.0+t*(_ipow(c, 4) *(((16)*x ....
16 + 7)*x + 1);
17 return t;
18 }
19 double _wendlandpsi_5_4_3(double x, .... double c){
20 double t=_ipow((1.0-x), 6);
21 t=82368.0+t*(_ipow(c, 6) *(((6)*x ....
22 + 1);
23 return t;
24 }
25 double _wendlandpsi_5_4_4(double x, .... double c){
26 double t=_ipow((1.0-x), 5);
27 t=3459456.0+t*(_ipow(c, 8) *((1);}
28 return t;
29 }
```

Note that _wendlandpsi_5_4_0 corresponds to $\psi_0$ in the example, but with $x = cr$ as argument.

When starting with $\Psi_0(x) = \psi_{5,3}(c|x|)$ instead, the relevant definitions are:

1The tool is available at https://gitlab.com/hjortur/ wendland-function-generator/ with example outputs.
Listing 2: Generated code for the $\psi_{5,3}$ family.

```c
1 double _wendlandpsi_5_3_0(double x, double c) {  
2   double t = _ipow((1.0 - x), 8);  
3   t = 1.0*t + (((32)*x + 25)*x + ... 
4     8)*x + 1);  
5   return t;  
6 }
7 double _wendlandpsi_5_3_1(double x, double c) {  
8   double t = _ipow((1.0 - x), 7);  
9   t = -22.0*t + _ipow(c, 2)*((16)*x + 
10     7)*x + 1);  
11   return t;  
12 }
13 double _wendlandpsi_5_3_2(double x, double c) {  
14   double t = _ipow((1.0 - x), 6);  
15   t = 528.0*t + _ipow(c, 4)*((6)*x + 
16     4)*x + 1);  
17   return t;  
18 }
19 double _wendlandpsi_5_3_3(double x, double c) {  
20   double t = _ipow((1.0 - x), 5);  
21   t = -22176.0*t + _ipow(c, 6);  
22   return t;  
23 }
```

Note that the polynomials _wendlandpsi_5_3_1 and _wendlandpsi_5_3_2 differ only by a multiplication of a constant and a power of $c$, and both polynomials are a representative of the Wendland function $\psi_{5,2}$.

The function _ipow(x,i) evaluates $x^i$ where $x$ is a double and $i$ is a positive integer. A possible efficient implementation is given by:

```c
// Function for fast squaring to ...  
// integer power  
static double _ipow(double base, ...  
int exp)  
{  
  double result = 1.0;  
  for (;;)  
  {  
    if (exp & 1)  
    {  
      exp >>= 1;  
      result *= base;  
    }  
    if (!exp)  
    {  
      break;  
    }  
    base *= base;  
  }  
  return result;  
}
```

The functions _wendlandpsi_x_y_z have been “flattened” in the sense that their domain is $[0,1]$. They require the user to premultiply the $x$ value with the chosen RBF-constant $c > 0$, that is for $\Psi(x) = \psi_{5,2}(c|x|)$, the user needs to pass in the value $c|x|$ and $c$ after ensuring that $c|x| \in [0,1]$. A possible implementation using the Armadillo library (Sanderson, 2010) could, for example, be:

```c
Listing 4: Example usage.
1 double psi3(const arma::vec &x, ...  
2 double c)  
3 {  
4   double cx=c*arma::norm(x,2);  
5   return ( cx < 1.0 ) ? ...  
6     _wendlandpsi_5_4_3(cx, c) ...  
7     0.0 ;  
8 }
```

The tool is a simple Python script named wendlandfunctions.py. When the script is run it outputs text for code- and header-files, which contain the Wendland function definitions. The user can supply the script with a parameter $-1$ and an integer value $m \geq 2$, in order to output code for Wendland functions from $\psi_{2,1}$ up to $\psi_{m,i}$ for all $0 \leq i < m$.

### 5.1 Performance

In previous work (Bjornsson and Hafstein, 2018) we have compared different methods to evaluate these Wendland functions. The methods used for point evaluation were:

- Having them in factorized form, as our software tool provides, see Listing 1.
- Fully expanded polynomials and evaluated using Horner Scheme, as in (Argaez et al., 2017).
- Precomputing the function in high precision (see below) at $10^7$ evenly spaced points on the interval $[0,1]$ and using them as a lookup table. That is, look up the closes value.
- Using the same lookup table but additionally linearly interpolate between two nearest neighbours to improve accuracy.

Figure 1 shows the performance of evaluating $\psi_{7,2}$ at $10^7$ different points on the interval $[0,1]$, and Figure 2 shows the highest relative error at these points, with these four different methods. To estimate the relative error we calculated the value of the polynomial $\psi_{7,2}$ in Matlab using variable precision arithmetic (VPA), up to 32 significant digits, then rounded the value to the closest double precision floating point number. Note that the factorized form and Lookup-table are close in speed, but the factorized form gives much greater accuracy.
Algorithm and Software to Generate Code for Wendland Functions in Factorized Form

ACKNOWLEDGEMENT

This research was supported by the Icelandic Research Fund (Rannís) in grant number 152429-051, Lyapunov Methods and Stochastic Stability.

REFERENCES


6 CONCLUSION

We have developed an algorithm and created a tool to generate a C/C++ library for Wendland’s compactly supported Radial Basis Functions with arbitrary parameters in a factorized form. This allows for the efficient and numerically accurate evaluation of these functions. This is desirable since previously they were generated in a non-optimal form or had to be evaluated by hand, which is a tedious and error prone process. Additionally, the software generates a whole family of Wendland functions suitable for solving collocation problems for each initial Wendland function $\psi_{l,k}$ and its support radius $c^{-1}$. The tool takes less than a second to output the .c and .h files for all the Wendland function families up to and including order 8.