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

Hjortur Bjornsson and Sigurdur Hafstein[a]
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 a useful tool to analyse stability of various dynamical systems, either deterministic or stochastic, cf. 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 $a_0, a_1, \dots a_d$ 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 $\Omega \subset \mathbb{R}^n$ and a boundary $\Gamma \subset \mathbb{R}^d$, a solution to the Partial Differential Equation (PDE)

[a] https://orcid.org/0000-0003-0073-2765

Bjornsson, H. and Hafstein, S.
Algorithm and Software to Generate Code for Wendland Functions in Factorized Form.
DOI: 10.5220/0008191901560162
In *Proceedings of the 16th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2019)*, pages 156-162
ISBN: 978-989-758-380-3
Copyright © 2019 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

$$\begin{cases} LV(x) = h(x) & x \in \Omega \\ V(x) = c(x) & x \in \Gamma, \end{cases} \tag{1}$$

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 $X_1 = \{x_1, \dots, x_N\} \subset \Omega$ and $X_2 = \{\xi_1, \dots, \xi_M\} \subset \Gamma$ and solving the interpolation problem

$$\begin{cases} LV(x_i) = h(x_i) & \text{for all } i = 1, \dots, N \\ V(\xi_i) = c(\xi_i) & \text{for all } i = 1, \dots, M. \end{cases}$$

The solution to this interpolation problem is given by

$$\begin{aligned} V(x) = & \sum_{k=1}^{N} \alpha_k (\delta_{x_k} \circ L)^y \psi(\|x-y\|) \\ & + \sum_{k=1}^{M} \alpha_{N+k} (\delta_{\xi_k} \circ L^0)^y \psi(\|x-y\|), \end{aligned} \tag{2}$$

where $\delta_y V(x) = V(y)$, the superscript y denotes that the operator is applied with respect to the variably y, and the operator L^0 is the identity operator. Here the function ψ is a compactly supported RBF (Wendland, 2017). The constants α_i are determined as a solution to the linear system $A\alpha = \gamma$, where A is the symmetric and positive definite matrix

$$A = \begin{bmatrix} B & C \\ C^T & D \end{bmatrix}$$

and the matrices $B = (b_{jk})_{j,k=1,\dots,N}$, $C = (c_{jk})_{j=1,\dots,N,k=1,\dots,M}$ and $D = (d_{jk})_{j,k=1,\dots,M}$ have elements:

$$\begin{aligned} b_{jk} &= (\delta_{x_j} \circ L)^x (\delta_{x_k} \circ L)^y \psi(x-y) \\ c_{jk} &= (\delta_{x_j} \circ L)^x (\delta_{\xi_k} \circ L^0)^y \psi(x-y) \\ d_{jk} &= (\delta_{\xi_j} \circ L^0)^x (\delta_{\xi_k} \circ L^0)^y \psi(x-y). \end{aligned}$$

To compute such a Lyapunov function a large number of evaluations of the function ψ and its derivatives is necessary. To verify the properties of a Lyapunov function for the function computed, even more evaluations are necessary. Therefore, it turned out to be essential that these evaluations could be carried out in an efficient and accurate way.

3 WENDLAND FUNCTIONS

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

$$\psi_{l,0}(r) = [(1-r)_+]^l \tag{3}$$

and

$$\psi_{l,k+1}(r) = C_{l,k+1} \int_r^1 t\psi_{l,k}(t)\mathrm{d}t, \tag{4}$$

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 $C_{l,k+1} \neq 0$ 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

$$-C_{l,k+1}\psi_{l,k}(r) = \frac{\frac{d}{dr}\psi_{l,k+1}(r)}{r}. \tag{5}$$

It is not difficult to verify that the functions

$$\Phi_{l,0}(r) = [(1-r)_+]^l \quad \text{and} \tag{6}$$

$$\Phi_{l,k}(r) = \int_r^1 \Phi_{l,0}(t)t(t^2-r^2)^{k-1}\mathrm{d}t \quad \text{for } k > 0 \tag{7}$$

also satisfy a relation of the form

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

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

$$\begin{aligned} & \frac{d}{dr} \int_r^1 \Phi_{l,0}(t)t(t^2-r^2)^{k-1}\mathrm{d}t \\ & = -2r(k-1) \int_r^1 \Phi_{l,0}(t)t(t^2-r^2)^{k-2}\mathrm{d}t. \end{aligned}$$

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, cf. 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 $\mathrm{supp}(\psi_{l,k}) = [0,1]$.
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 $l := \lfloor \frac{n}{2} \rfloor + k + 1$, 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 $\Psi(x) := \psi_{l,k}(c\|x\|)$ is then a C^{2k} function with $\mathrm{supp}(\Psi) = \mathcal{B}^d(0, c^{-1}) \subset \mathbb{R}^n$, where $\mathcal{B}^n(0, c^{-1})$ is the closed n-dimensional ball around the origin with radius c^{-1}.

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:

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

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

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

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 ψ into the form

$$\psi(r) = C_2 (1-r)^{l+k} p_{l,k}(r) \tag{8}$$

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

$$\begin{aligned} \psi_{l,k-1}(r) &= \frac{\frac{d}{dr}\left[(1-r)^{l+k} p_{l,k}(r)\right]}{r} \\ &= \frac{1}{r}(1-r)^{l+k-1}((1-r)p'_{l,k}(r) - (l+k)p_{l,k}(r)). \end{aligned} \tag{9}$$

Therefore we have

$$\begin{aligned} p_{l,k-1}(r) &:= \frac{\psi_{l,k-1}(r)}{(1-r)^{l+k-1}} \\ &= \frac{1}{r}\left[(1-r)p'_{l,k}(r) - (l+k)p_{l,k}(r)\right]. \end{aligned} \tag{10}$$

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 by 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 $\hat{p}_{l,k-1}$ and a constant b_{k-1} such that

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

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

$$\psi_i(r) = b_1 \cdots b_i (1-r)^{l+k-i} \hat{p}_{l,k-i}(r), \; i = 1, 2, \ldots, k \tag{11}$$

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

The above list follows the notation in (Giesl, 2007) were ψ_0 is the polynomial given in (8) and is equal to the Wendland function $\psi_{l,k}$, and $\psi_1, \ldots, \psi_i$ are the Wendland functions given by $\psi_{l,k-1}, \ldots, \psi_{l,k-i}$ respectively, see equation (4). It is important to keep track of the constants $b_1, \ldots, b_i$ 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 $p'(t) = (1-t)^5 t(t^2 - r^2)^3$ and we obtain

$$\begin{aligned}\psi(r) &= \int_r^1 (1-t)^{5+4} t(t^2-r^2)^3 dt \\ &= -\frac{16}{3003}r^{13} + \frac{1}{24}r^{12} - \frac{32}{231}r^{11} + \frac{1}{4}r^{10} - \frac{16}{63}r^9 \\ &+ \frac{1}{8}r^8 - \frac{1}{42}r^6 + \frac{1}{168}r^4 - \frac{1}{924}r^2 + \frac{1}{10296} \\ &= \frac{1}{72072}(1-r)^9(384r^4 + 453r^3 + 237r^2 + 63r + 7)\end{aligned}$$

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

$$\begin{aligned}\psi_{5,4}(r) &= \psi_0(r) \\ &= (1-r)^9(384r^4 + 453r^3 + 237r^2 + 63r + 7) \\ \psi_{5,3}(r) = \psi_1(r) &= \frac{\frac{d}{dr}\psi_0(r)}{r} \\ &= -156(1-r)^8(32r^3 + 25r^2 + 8r + 1) \\ \psi_{5,2}(r) = \psi_2(r) &= \frac{\frac{d}{dr}\psi_1(r)}{r} \\ &= 3{,}432(1-r)^7(16r^2 + 7r + 1) \\ \psi_{5,1}(r) = \psi_3(r) &= \frac{\frac{d}{dr}\psi_2(r)}{r} \\ &= -82{,}368(1-r)^6(6r + 1) \\ \psi_{5,0}(r) = \psi_4(r) &= \frac{\frac{d}{dr}\psi_3(r)}{r} \\ &= 3{,}459{,}456(1-r)^5\end{aligned}$$

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

$$\begin{aligned}\psi_{l,k-j} = \psi_j(r) &= \frac{\frac{d}{dr}\psi_{j-1}(r)}{r} \\ &= \frac{\frac{d}{dr}\psi_{l,k-j+1}(r)}{r}, \quad \text{for } j = 1, \ldots, k,\end{aligned}$$

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[1] 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 Ψ.

Listing 1: Generated code for the $\psi_{5,4}$ family.

```
double __wendlandpsi_5_4_0(double x, ...
     double c){
    double t=__ipow((1.0-x),9);
    t=1.0*t*(((((384)*x + 453)*x + ...
         237)*x + 63)*x + 7);
    return t;
}
double __wendlandpsi_5_4_1(double x, ...
     double c){
    double t=__ipow((1.0-x),8);
    t=-156.0*t*__ipow(c,2)*((((32)*x ...
         + 25)*x + 8)*x + 1);
    return t;
}
double __wendlandpsi_5_4_2(double x, ...
     double c){
    double t=__ipow((1.0-x),7);
    t=3432.0*t*__ipow(c,4)*(((16)*x ...
         + 7)*x + 1);
    return t;
}
double __wendlandpsi_5_4_3(double x, ...
     double c){
    double t=__ipow((1.0-x),6);
    t=-82368.0*t*__ipow(c,6)*((6)*x ...
         + 1);
    return t;
}
double __wendlandpsi_5_4_4(double x, ...
     double c){
    double t=__ipow((1.0-x),5);
    t=3459456.0*t*__ipow(c,8)*(1);
    return t;
}
```

Note that `__wendlandpsi_5_4_j` corresponds to ψ_j 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:

[1]The tool is available at https://gitlab.com/hjortur/wendland-function-generator/ with example outputs.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Listing 2: Generated code for the $\psi_{5,3}$ family.

```
double __wendlandpsi_5_3_0(double ...
    x, double c){
    double t=__ipow((1.0-x),8);
    t=1.0*t*((((32)*x + 25)*x + ...
        8)*x + 1);
    return t;
}
double __wendlandpsi_5_3_1(double ...
    x, double c){
    double t=__ipow((1.0-x),7);
    t=-22.0*t*__ipow(c,2)*(((16)*x ...
        + 7)*x + 1);
    return t;
}
double __wendlandpsi_5_3_2(double ...
    x, double c){
    double t=__ipow((1.0-x),6);
    t=528.0*t*__ipow(c,4)*((6)*x ...
        + 1);
    return t;
}
double __wendlandpsi_5_3_3(double ...
    x, double c){
    double t=__ipow((1.0-x),5);
    t=-22176.0*t*__ipow(c,6)*(1);
    return t;
}
```

Note that the polynomials `__wendlandpsi_5_3_1` and `__wendlandpsi_5_4_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:

Listing 3: Exponentiation routine.

```
//Function for fast squaring to ...
    integer power
//Posted by user Elias Yarrkov on ...
    Stackoverflow.
static double __ipow(double base, ...
    int exp){
    double result = 1.0;
    for (;;){
        if (exp & 1)
            result *= base;
        exp >>= 1;
        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 $\Psi(x) = \psi_{l,k}(c\|x\|)$, 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:

Listing 4: Example usage.

```
double psi3(const arma::vec &x, ...
    double c){
    double cx=c*arma::norm(x,2);
    return ( cx < 1.0 ) ? ...
        __wendlandpsi_5_4_3(cx,c) ...
        : 0.0;
}
```

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 `--l` 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 method to evalute 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 Horners 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.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

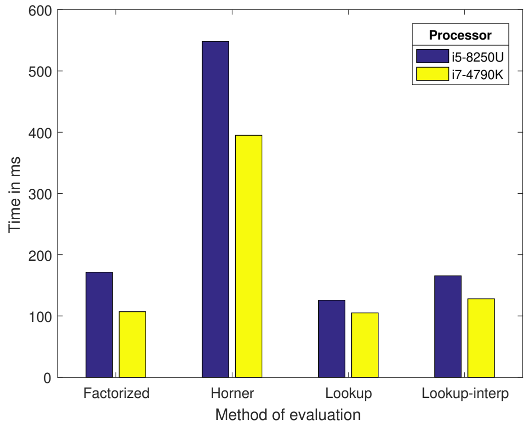

Figure 1: Running time of evaluating $\psi_{7,2}$ at 10^7 points.

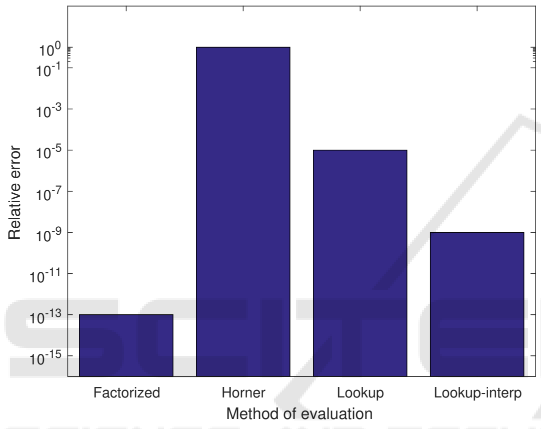

Figure 2: Relative error of evaluations of $\psi_{7,2}$. Note the logarithmic scale of the y-axis.

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.

ACKNOWLEDGEMENT

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

REFERENCES

Argaez, C., Hafstein, S., and Giesl, P. (2017). Wendland functions - a C++ code to compute them. In *Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - Volume 1: SIMULTECH,*, pages 323–330. INSTICC, SciTePress.

Bjornsson, H. and Hafstein, S. (2018). Verification of a numerical solution to a collocation problem. In *Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics - Volume 1: CTDE,*, pages 587–594. INSTICC, SciTePress.

Bjornsson, H., Hafstein, S., Giesl, P., Gudmundsson, S., and Scalas, E. (2019). Computation of the stochastic basin of attraction by rigorous construction of a Lyapunov function. *Discrete and Continuous Dynamical Systems-Series B (to appear)*.

Buhmann, M. (2000). Radial basis functions. In *Acta numerica, 2000*, volume 9 of *Acta Numer.*, pages 1–38. Cambridge Univ. Press, Cambridge.

Buhmann, M. (2003). *Radial basis functions: theory and implementations*, volume 12 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge.

Burrus, C. S., Fox, J. W., Sitton, G. A., and Treitel, S. (2003). Horner's method for evaluating and deflating polynomials. *DSP Software Notes, Rice University, Nov*, 26.

Floater, M. and Iske, A. (1996). Multistep scattered data interpolation using compactly supported Radial Basis Functions. *J. Comput. Appl. Math.*, 73(1-2):65–78.

Franke, C. and Schaback, R. (1998). Solving partial differential equations by collocation using radial basis functions. *Appl. Math. Comput.*, 93(1):73–82.

Giesl, P. (2007). *Construction of Global Lyapunov Functions Using Radial Basis Functions*, volume 1904 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin.

Giesl, P. (2008). Construction of a local and global Lyapunov function using radial basis functions. *IMA J. Appl. Math.*, 73(5):782–802.

Giesl, P. and Hafstein, S. (2015). Computation and verification of Lyapunov functions. *SIAM Journal on Applied Dynamical Systems*, 14(4):1663–1698.

Hafstein, S., Gudmundsson, S., Giesl, P., and Scalas, E. (2018). Lyapunov function computation for autonomous linear stochastic differential equations using sum-of-squares programming. *Discrete and Continuous Dynamical Systems - Series B*, 23(2):939–950.

Khalil, H. (2002). *Nonlinear systems*. Pear, 3. edition.

Khasminskii, R. (2012). *Stochastic stability of differential equations*. Springer, 2nd edition.

Mao, X. (2008). *Stochastic Differential Equations and Applications*. Woodhead Publishing, 2nd edition.

Sanderson, C. (2010). Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA.

Sastry, S. (1999). *Nonlinear Systems: Analysis, Stability, and Control*. Springer.

Vidyasagar, M. (2002). *Nonlinear System Analysis*. Classics in applied mathematics. SIAM, 2. edition.

Wendland, H. (1998). Error estimates for interpolation by compactly supported Radial Basis Functions of minimal degree. *J. Approx. Theory*, 93:258–272.

Wendland, H. (2005). *Scattered data approximation*, volume 17 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge.

Wendland, H. (2017). Multiscale radial basis functions. In *Frames and other bases in abstract and function spaces*, Appl. Numer. Harmon. Anal., pages 265–299. Birkhäuser/Springer, Cham.

Wu, Z. (1992). Hermite-Birkhoff interpolation of scattered data by radial basis functions. *Approx. Theory Appl.*, 8(2):1–10.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS