Dolomites Research Notes on Approximation

Proceedings of DWCAA12, Volume 6 · 2013 · Pages 37–50

Direct discretizations with applications to meshless methods for PDEs

Robert Schaback[*]

Abstract

A central problem of numerical analysis is the approximate evaluation of integrals or derivatives of functions. In more generality, this is the approximate evaluation of a linear functional defined on a space of functions. Users often just have values of a function u at scattered points $x_1,\dots,x_N$ in the domain Ω of u, and then the value $\lambda(u)$ of a linear functional λ must be approximated via *direct approximation formulae*

$$\lambda(u)\approx\sum_{j=1}^{N}a_j u(x_j),$$

i.e. we approximate λ by point evaluation functionals $\delta_{x_j}\ :\ u\mapsto u(x_j)$. Such *direct discretizations* include classical cases like Newton–Cotes integration formulas or divided differences as approximations of derivatives. They are central for many methods solving partial differential equations, and their error analysis has a long–standing history going back to Peano and his kernel theorem. They also have a strong connection to Approximation Theory.

Here, we apply certain optimizations to certain classes of such discretizations, and we evaluate error norms in Beppo–Levi– and Sobolev spaces. This allows to compare discretizations of very different types. including those that are based on exactness on polynomials and those which are by definition optimal on certain function spaces but lack sparsity. Special attention is given to discretizations that are used within current *meshless methods* for solving partial differential equations.

Much of this work is based on recent collaboration with Oleg Davydov of the University of Strathclyde, Scotland, and Davoud Mirzaei of the University of Isfahan, Iran.

2000 AMS subject classification: 65M06, 65N06, 65D15, 65D25, 65D30, 65D32, 65D0, 41A05, 41A10, 41A55, 42A82 .

Keywords: Meshless methods, cubature, derivatives, multivariate polynomials.

1 Introduction and Overview

Meshless Methods formulate PDE problems via *trial functions* parametrized *entirely in terms of nodes* [4]. Partial derivatives at points or local integrals against test functions have to be expressed in terms of function values at neighbouring scattered points. The resulting formulas are of finite–difference type, but they are by no means unique. Various optimization criteria can be applied to produce "optimal" formulas, and this contribution surveys some of these.

The most common idea is to ask for exactness of the formula for multivariate polynomials up to a certain order. This *polynomial consistency* needs some requirements on the scattered points to be satisfied, but if more than a minimal number of local scattered points is admitted, there are multiple solutions that call for optimization under the constraints of exactness on polynomials. This optimization can be carried out in various ways, and we study some of these, summarizing joint work with Oleg Davydov, Univ. of Strathclyde [6]. One can go for optimal sparsity, or apply Moving Least Squares techniques or optimize formulas with general weights in one way or another. A general error analysis is carried out that indicates which criteria for optimization are useful. In contrast to classical Moving Least Squares discretizations of Meshless Methods, we do not take derivatives of shape functions here. This part is based on joint work with Davoud Mirzaei, Univ. of Isfahan [11, 10, 12].

Derivative formulas produced by taking exact derivatives of kernel–based local interpolants usually have no polynomial consistency, but they can be proven to compete favourably with optimized polynomially consistent formulas, since they are optimal estimators of derivatives for all functions in the native Hilbert space of the kernel. Furthermore, all competing direct derivative formulas can be compared by explicitly calculating the norms of their corresponding error functionals on Sobolev or Beppo–Levi spaces.

A final section uses the latter fact to provide results of extensive numerical experiments comparing all of these methods. It turns out that polynomially consistent formulas compete well with all the others as functionals on Sobolev or Beppo–Levi spaces, though they are necessarily (but not too much) inferior to the optimized kernel–based formulas on these spaces. For kernel–based formulas, one can use smooth kernels with no problems caused by excessive smoothness. But one should ensure some form of sparsity by careful point selection, and the selection of nearest neighbours comes out to be a very good choice.

[*]Institut für Numerische und Angewandte Mathematik, University of Göttingen (D).

2 Meshless Methods

By the pioneering survey article [4], *Meshless Methods* formulate PDE problems via *trial functions* parametrized *"entirely in terms of nodes"*. Let $X = \{x_1, \dots, x_N\} \subset \Omega \subset \mathbb{R}^N$ be a set of nodes. Then each trial function u should take the form

$$u(x) = \sum_{j=1}^{N} s_j(x)u(x_j) \tag{1}$$

where usually, but not necessarily, the *shape functions* $s_1, \dots, s_N$ satisfy the *Lagrange conditions*

$$s_j(x_k) = \delta_{jk},\ 1 \le j,k \le N.$$

A very popular way to get shape functions at scattered nodes is to apply Moving Least Squares, while another method uses translates of kernels or Radial Basis Functions. We consider both cases later.

If a linear PDE problem is given in the form

$$\begin{array}{rcll} Lu &=& f & \text{in } \Omega, \\ Bu &=& g & \text{in } \Gamma := \partial\Omega, \end{array} \tag{2}$$

with a linear differential operator L and a linear "boundary operator" B, it can be discretized in strong or weak form as

$$\lambda_m(u) = f_m,\ 1 \le m \le M, \tag{3}$$

with linear *functionals*. Then the problem is reformulated as a linear system

$$\lambda_m(u) = \sum_{j=1}^{N} \lambda_m(s_j)u(x_j) = f_m,\ 1 \le m \le M \tag{4}$$

in terms of values at nodes. This system may be overdetermined, but it will be approximately solvable, if the original problem has a true solution u^* that has a good approximation u from the meshless trial space. In fact,

$$\begin{array}{rcl} f_m &=& \lambda_m(u^*) \\ &\approx& \lambda_m(u) \\ &=& \displaystyle\sum_{j=1}^{N} \lambda_m(s_j)u(x_j),\ 1 \le m \le M. \end{array}$$

The functionals λ_m come in various forms. We explain part of them by considering the standard elliptic problem

$$\begin{array}{rcll} -\nabla \cdot (a(x)\nabla u(x)) &=& f_\Omega(x) & \text{in } \Omega, \\ u(y) &=& f_D(y), & \text{in } \Gamma_D \subset \Gamma = \partial\Omega, \\ \frac{\partial u}{\partial n}(y) &=& f_D(y), & \text{in } \Gamma_N \subset \Gamma, \end{array}$$

Strong PDE formulations will use *collocation* via functionals

$$\begin{array}{rcllll} \lambda_j(u) &:=& -\nabla(a(\cdot)\nabla u(\cdot))(x_j) &=& f_j, & x_j \in \overline{\Omega} \\ \lambda_k(u) &:=& u(y_k) &=& f_k, & y_k \in \Gamma_D, \\ \lambda_\ell(u) &:=& \frac{\partial u}{\partial n}(z_\ell) &=& f_\ell, & z_\ell \in \Gamma_N \subset \Gamma \end{array}$$

that evaluate differential operators at single points. *Global weak* methods rewrite the main equation as

$$\int_\Omega (a(x)\nabla u(x))^T \nabla v(x)dx = \int_\Omega f_\Omega(x)v(x)dx$$

for test functions v vanishing on the boundary. This leads to functionals

$$\lambda_j(u) := \int_\Omega (a(x)\nabla u(x))^T \nabla v_j(x)dx$$

for test functions v_j, and the values f_j for the equations $\lambda_j(u) = f_j$ need integrations

$$f_j = \int_\Omega f_\Omega(x)v_j(x)dx$$

against the same test functions.

Local weak forms rewrite the differential equation on small subdomains Ω_h as

$$\int_{\Omega_h} (a(x)\nabla u(x))^T \nabla v(x)dx - \int_{\partial\Omega_h} \frac{\partial u}{\partial n}(x)v(x)dx = \int_{\Omega_h} f_\Omega(x)v(x)dx \tag{5}$$

for test functions v that vanish not necessarily on the boundary. If the boundary of Ω_h hits the Neumann boundary, the factor $\frac{\partial u}{\partial n}$ can be replaced by f_N there. This gives the additional local functionals

$$\lambda_\ell(u) := \int_{\partial\Omega_h \cap (\Gamma \setminus \Gamma_N)} \frac{\partial u}{\partial n}(x) v_\ell(x) dx \tag{6}$$

and

$$\lambda_j(u) := \int_{\Omega_h} (a(x)\nabla u(x))^T \nabla v_j(x) dx$$

for test functions v_j on Ω_h. This arrangement of localized functionals is the main variant MLPG1 of the *Meshless Local Petrov Galerkin* method of S.N. Atluri and collaborators [3].

A simplified case called MLPG5 arises when simply choosing the test functions to be constant. Then the main PDE discretization takes the form

$$\lambda_{\Omega_h}(u) := -\int_{\partial\Omega_h \cap (\Gamma \setminus \Gamma_N)} \frac{\partial u}{\partial n}(x) dx = \int_{\partial\Omega_h \cap \Gamma_N} f_N(x) dx + \int_{\Omega_h} f_\Omega(x) dx \tag{7}$$

and involves only boundary integrals like (6) of the normal derivative.

In all of these cases, it is necessary to have cheap evaluations of the functionals on the trial space, and this is the main topic of this contribution. Since numerical integration is required in all weak formulations, it will pay off to have explicit formulas for exact integration.

Time–dependent PDEs can in many cases be handled via meshless methods that follow the above strategy in the spatial variables $x \in \Omega$. The representation (1) is replaced by

$$u(x,t) = \sum_{j=1}^N s_j(x) u(x_j,t)$$

using the spatial shape functions. If D is a linear differential operator with respect to time, one can use

$$Du(x,t) = \sum_{j=1}^N s_j(x) Du(x_j,t)$$

to express everything in terms of time–dependent values at nodes. Together with the other parts of the PDE, this connects values $Du(x_j,t)$ with values $u(x_k,t)$, and thus leads to a system of ODEs that can be solved by the Method of Lines or by certain timestepping methods. There are plenty of papers that apply this to various types of time–dependent PDEs. For details, we refer to the paper [10] which implements discretizations that will be described below.

3 Direct Discretizations

No matter which functionals λ come from the PDE problem along the lines of the previous section, meshless methods usually apply them to trial functions (1) as

$$\lambda(u) = \sum_{j=1}^N \lambda(s_j) u(x_j).$$

This requires evaluation of the functional on all shape functions $s_1, \dots, s_N$. This can be a serious problem if the shape functions are implicitly defined, like in all meshless methods that use Moving Least Squares shape functions.

But it is by no means necessary to use shape functions at all at this point. The above formula is just one way of approximating $\lambda(u)$ “in terms of values at nodes”. One can generally go for

$$\lambda(u) \approx \sum_{j=1}^N a_j u(x_j) \tag{8}$$

with certain coefficients a_j. It should be emphasized that the coefficients a_j in (8) are considered as *given* values that are determined by some specific numerical method, e.g. by one of the choices of discretization schemes discussed later in the paper.

If this is done for all functionals λ_m of (3) similarly, one gets the linear system

$$\lambda_m(u) \approx \sum_{j=1}^N a_{mj} u(x_j) = f_m, \ 1 \le m \le M \tag{9}$$

instead of (4). Here, the matrix entries are more generic, replacing the specific values $\lambda_m(s_j)$ in (4) that depended on shape functions. Being “entirely in terms of values at nodes”, this still follows the philosophy of meshless methods, but without using any shape functions. Once the system is approximately solved, values $u(x_j)$ are known and can be used by any interpolation or approximation method to calculate values at other locations. We call (8) a *direct discretization* of the functional λ, and the rest of this contribution will deal with these. We use the term *direct* to emphasize the fact that these approximations avoid shape functions.

In particular, if functionals contain derivatives, direct discretizations need not evaluate derivatives of shape functions. The literature has the term *diffuse derivatives* [13, 14] for direct discretizations of derivative functionals. We strictly avoid the term "diffuse" because there is nothing uncertain or diffuse there. Instead, the "diffuse" derivatives are direct discretizations of derivatives. The papers [11, 12] prove that there is no loss of accuracy to use direct discretizations replacing derivatives of shape functions, and they used the notion *direct derivatives* for direct approximations of derivatives.

For Dirichlet data, it makes sense [10] to couple the known nodal value $u(y_k)$ at a point y_k on the Dirichlet boundary to neighbouring unknown nodal values $u(x_j)$ at points x_j inside the domain via an additional formula of the type

$$u(y_k) \approx \sum_j a_j u(x_j),$$

which is another direct discretization. In should be interpreted as a formula providing extrapolation to the boundary, and it generates an equation in (9) that connects given Dirichlet data $u(y_k)$ to unknown nodal values $u(x_j)$.

The following sections will deal with several techniques for calculating direct discretizations (8) that determine useful coefficients in (9). In particular, we shall have a close look at the error functional

$$\varepsilon_{\lambda,X,\mathbf{a}} := \lambda - \sum_{j=1}^{N} a_j \delta_{x_j} \tag{10}$$

and try to make it small in one way or another, and we want a cheap calculation of the discretization. We can focus on single functionals for this purpose. In the context of classical theory for numerical solution of PDEs, this deals with *consistency* only, not with *stability*. Stability will depend on which and how many functionals are used for the whole setup of equations (9) in order to let the coefficient matrix have rank N and a stable pseudoinverse. However, readers should keep in mind that the notion of stability of methods for solving time–dependent PDEs is different.

Note that in our setting (3) we always assume that the given data values f_j are exactly the values $\lambda_j(u)$ of the true solution u of the problem. This is a *noiseless situation* and allows us to interpret the approximation error in (9) as a value of a continuous linear functional. If data are polluted by additive noise, e.g. $f_j = \lambda_j(u) + \delta_j$ with nonzero δ_j, this approach fails and requires a completely new error analysis leading to regularization techniques. See e.g. [8] for regularized discretizations of derivatives that can deal with noise.

4 Direct Discretizations via Polynomials

We start with methods for (8) that use polynomials. This is well–known from the univariate case. There, the standard technique is to require *exactness* of (8) for a fixed given functional on a finite–dimensional space of polynomials, and error bounds are obtained via Peano kernels. We shall come back to this in section 4.3.

In multivariate meshless methods, polynomially exact discretizations usually come via Moving Least Squares, but with evaluation of the functionals on the shape functions, which in turn need pointwise calculation. If the functionals contain some integration, this means that one has to evaluate values or derivatives of shape functions on the integration points. Here, we shall avoid the use of shape functions, and we generalize the setting of Moving Least Squares.

We assume a polynomial order (= total degree plus one) m for which a formula (8) for a single functional λ should be exact, and we denote by $\mathcal{P}_m^d$ the space of these polynomials in d variables. Its dimension $\binom{m-1+d}{d}$ will be abbreviated by Q, and we choose a basis $p_1, \dots, p_Q$. The set of (usually local) nodes will be $X = \{x_1, \dots, x_N\}$. In what follows, we focus on a single functional λ in the sense of (8), but we keep in mind that this deals only with a row of the system (9).

With the $Q \times N$ matrix $\mathbf{P} = \mathbf{P}(X, m, d)$ with values $p_i(x_j)$ and the vector

$$\mathbf{p} := (\lambda(p_1), \dots, \lambda(p_Q))^T \in \mathbb{R}^Q,$$

exactness of (8) up to order m means solvability of the linear system

$$\mathbf{Pa} = \mathbf{p}, \;\text{ i.e. } \sum_{j=1}^{N} a_j p_i(x_j) = \lambda(p_i),\; 1 \le i \le Q. \tag{11}$$

This is satisfied if $\operatorname{rank}(\mathbf{P}) = Q \le N$, but this "*unisolvency*" condition is not necessary. For example, take $\lambda(u) = \Delta u(x)$ for some point $x \in \mathbb{R}^2$ and the five–point star discretization. It has $N = 5$ points, is exact for all polynomials in $d = 2$ variables up to order $m = 4$, and thus has $Q = 10$.

For what follows, we always assume solvability of (11), but we will often have additional degrees of freedom that we can use for some kind of optimization. We use boldface notation as soon as we are in Linear Algebra, but nowhere else.

But we emphasize at this point that the system (11) requires only polynomials, no shape functions, and the most expensive part will be the evaluation of $\lambda(p_i)$ in case of integrations of derivatives. But the integrands will be available in closed form, and the integration error can be easily controlled, in particular if the domain has a regular shape, e.g. a ball, a cube, or a polyhedron. In MLPG5, there is no test function, and then there is no additional error induced by numerical integration, and no "background mesh" for integration.

4.1 Sparse Polynomial Discretizations

If the system (11) is solvable, one can ask for a solution with a minimal number of nonzero coefficients. Papers on sparsity often work by minimizing the number of nonzero coefficients, called the zero–"norm". This is a highly nontrivial task, but it is relatively easy to come up with solutions that have only Q nonzero components, if $N \geq Q$. This can be done by the MATLAB backslash operator, for instance, or by *Orthogonal Matching Pursuit* in a simple implementation. In fact, one only has to project the right–hand side $\mathbf{p}$ of (11) into the column space of the matrix $\mathbf{P}$ and find a linear combination by linear independent columns. Details are in [15], but there is a shortcut. If a pivoted QR decomposition of the matrix of the linear system

$$\begin{pmatrix} \mathbf{P} & -\mathbf{p} \\ 1_N^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{a} \\ \alpha \end{pmatrix} = \begin{pmatrix} 0_Q \\ 1 \end{pmatrix}$$

is performed that starts with the final column, there will be an automatic column selection by a greedy choice of the columns that reduce the ℓ_2 error in an optimal way [15].

However, none of these methods, including minimizing the ℓ_1 norm of the coefficients, yields the sparsest solution in all cases. But, of all the simplified methods, the ℓ_1 norm minimization often performs best with respect to sparsity. We shall include it into our numerical examples, together with the Orthogonal Matching Pursuit (OMP) solution and the simple MATLAB backslash operation $\mathbf{a} = \mathbf{P}\backslash\mathbf{p}$ that also does a pivoted QR decomposition, but not dependent on the right–hand side.

4.2 Moving Least Squares

A very popular case within meshless methods is the approach via Moving Least Squares that we describe now.

In the standard form of Moving Least Squares, there is no connection to direct discretizations. For *given* values $u(x_1), \dots, u(x_N)$ at scattered points $x_1, \dots, x_N$ near a fixed point z, it tries to find a value $u(z)$ at z that matches the data well. It calculates a polynomial $p^* \in \mathcal{P}_m^d$ that minimizes

$$\sum_{j=1}^{N} (p^*(x_j) - u(x_j))^2 w^2(z, x_j)$$

for weights coming from a weight function w that is localized around some fixed point z. If p^* is found, the value $p^*(z)$ is taken as $u(z)$. Numerically, this is the minimization of $\|\mathbf{W}_z(\mathbf{P}^T\mathbf{c} - \mathbf{u})\|_2$ for a coefficient vector $\mathbf{c} \in \mathbb{R}^Q$, given vectors $\mathbf{u} = (u(x_1), \dots, u(x_N))^T \in \mathbb{R}^N$ for the data and putting the positive weights $w(z, x_j)$ into a diagonal matrix $\mathbf{W}_z$. Note that only $\mathbf{W}_z$ depends on z. The standard variational argument then shows that the solution vector $\mathbf{c}_z^* = (c_1^*(z), \dots, c_Q^*(z))^T \in \mathbb{R}^Q$ must satisfy the Gaussian normal equations $\mathbf{P}\mathbf{W}_z^2\mathbf{P}^T\mathbf{c}_z^* = \mathbf{P}\mathbf{W}_z^2\mathbf{u}$. If $\mathbf{P}$ has rank Q, this is solvable, and the resulting value at z is

$$p^*(z) = \sum_{i=1}^{Q} c_i^*(z) p_i(z).$$

The vector $\mathbf{c}_z^*$ can formally be written as $\mathbf{c}_z^* = \mathbf{B}_z\mathbf{u}$ with a $Q \times N$ matrix

$$\mathbf{B}_z = (\mathbf{P}\mathbf{W}_z^2\mathbf{P}^T)^{-1}\mathbf{P}\mathbf{W}_z^2.$$

Then the procedure yields

$$u(z) = p^*(z) = \sum_{i=1}^{Q}\sum_{j=1}^{N} b_{ij}(z) u(x_j) p_i(z) = \sum_{j=1}^{N} u(x_j) \underbrace{\sum_{i=1}^{Q} b_{ij}(z) p_i(z)}_{=:s_j(z)}$$

which is of the form (1) with the shape functions $s_1, \dots, s_N$. These functions will not necessarily be polynomials, unfortunately, unless the weights are chosen independent of z.

Standard applications of this for meshless methods would hit (1) by λ and proceed to calculate the values $\lambda(s_j)$. This will lead to an approximation of the form (8). If integrations are involved, the integrations do not run over polynomials. Derivatives at z in the functional are usually handled by taking derivatives of p^* at z, but derivatives at integration points need recalculation of the whole procedure at each integration point. *Diffuse* derivatives ignore the dependence of the b_{ij} on z and just take derivatives of the s_j via derivatives of the polynomials p_i. This will also lead to an approximation of the form (8), but without taking derivatives of shape functions.

The connection to direct discretizations works as follows. We calculate coefficients c_i^* by solving $\mathbf{P}\mathbf{W}^2\mathbf{P}^T\mathbf{c}^* = \mathbf{P}\mathbf{W}^2\mathbf{u}$, and we define our discretization of λ by

$$\lambda(u) \approx \lambda(p^*) = \sum_{i=1}^{Q} c_i^* \lambda(p_i) = \mathbf{p}^T\mathbf{c}^* = \mathbf{p}^T\mathbf{B}\mathbf{u} = \mathbf{p}^T(\mathbf{P}\mathbf{W}^2\mathbf{P}^T)^{-1}\mathbf{P}\mathbf{W}^2\mathbf{u}$$

which is of the required form. This allows arbitrary weights, but it involves a least–squares approach that is not general enough. By some standard calculations, the coefficient vector

$$\mathbf{a} = \mathbf{W}^2\mathbf{P}^T(\mathbf{P}\mathbf{W}^2\mathbf{P}^T)^{-1}\mathbf{p}$$

minimizes

$$\|\mathbf{a}\|^2_{2,1/w^2} := \sum_{j=1}^N \frac{a_j^2}{w_j^2} = \sum_{j=1}^N \frac{a_j^2}{w(z,x_j)^2}$$

with the reciprocals of the Moving Least Squares weights centered about a point z. Then one can use the identity $\lambda(p^*) = \mathbf{u}^T\mathbf{a}$ for calculating the direct discretization. However, if one has to approximate several functionals from values of functions at the same nodes, it is better to calculate a polynomial interpolant p^* with a coefficient vector $\mathbf{c}^*$ first and then take the exact values $\lambda(p^*)$ as approximations.

4.3 General Weights

We now stick to (8), make it exact on $\mathcal{P}_m^d$ via (11), and use the additional degrees of freedom to minimize some norm of the coefficient vector $\mathbf{a} \in \mathbb{R}^N$. For maximal sparsity, one can use section 4.1, or go for the norm $\|\mathbf{a}\|_1$.

If solvability of (11) is assumed, each optimization of the weights will be feasible, and will yield a possibly useful solution. A comparison should be made within the error analysis. There are at least two sufficiently general approaches to error bounds in the literature. In [12], results on stabilized polynomial reconstruction [17] are used, while [6] derives error bounds in terms of growth functions [5]. With a certain simplified shortcut, we explain these techniques by taking an arbitrary polynomial $p \in \mathcal{P}_m^d$ for bounding the error as

$$\begin{aligned} |\varepsilon_{\lambda,X,\mathbf{a}}(u)| &= \left|\lambda(u) - \sum_{j=1}^N a_j u(x_j)\right| \\ &= \left|\lambda(u-p) - \sum_{j=1}^N a_j (u-p)(x_j)\right| \\ &\le |\lambda(u-p)| + \left|\sum_{j=1}^N \frac{a_j}{w_j}(u-p)(x_j) w_j\right| \\ &\le |\lambda(u-p)| + \|\mathbf{a}\|_{q,1/w}\|\mathbf{u}-\mathbf{p}\|_{r,w} \end{aligned}$$

with $1/q + 1/r = 1$ and arbitrary positive weights $w_1, \dots, w_N$. There are several different views on this bound, and all of them give a specific insight.

First, assume that we restrict ourselves to a subset of Q points of X on which we can perform interpolation as

$$p(x) = \sum_{j=1}^Q p_j(x) p(x_j)$$

like in (1), and we interpolate u on these points by p. Then $\mathbf{u} = \mathbf{p}$, and the error consists just of $\lambda(u-p)$. Exactness then implies $a_j = \lambda(p_j)$, $1 \le j \le Q$ and there is nothing to minimize. This is the standard situation known from 1D, e.g. for numerical quadrature like Newton–Cotes formulae. The error of the discretization is exactly the evaluation of the functional on the error function of the interpolation. Even in 1D this can be fine or disastrous, depending on the locations of the points. Without oversampling, there will always be at least a $\log Q$ growth in case of nicely placed points, while there is an exponential growth with Q in case of regularly distributed data. To overcome this, the techniques summarized in [17] use oversampling, and then they get bounded interpolation processes that make $\lambda(u-p)$ manageable. Consequently, by a logic similar to the above one, and using Taylor polynomials like in the fourth view below, the authors of [12] get useful error bounds for these interpolation–based discretizations.

A second view would not take p as an interpolant, but rather argue with a best polynomial approximant p to u. Then the first part of the error bound is again independent of the discretization formula, but the second tells us that we should minimize $\|\mathbf{a}\|_{q,1/w}$ under the constraint $\mathbf{Pa} = \mathbf{p}$, no matter how good the best polynomial approximation to u actually is. But this leaves open how to choose the weights. Note that this approach leads to "derivative–free" error bounds which were fashionable quite a while ago.

The case $q = 2$ of the above argument already points towards Moving Least Squares, for general weights. More specifically, our third view is to take p as the outcome of Moving Least Squares, minimizing $\|\mathbf{u}-\mathbf{p}\|_{2,w}$ with MLS–specific weights. There are good bounds on $\lambda(u-p)$ in this case [1, 17], and one is left with minimizing $\|\mathbf{a}\|_{2,1/w}$ under the constraint $\mathbf{Pa} = \mathbf{p}$, bringing us back to the previous approach. By the standard duality arguments of Moving Least Squares, the minimal solution coefficients a_j^* are exactly $\lambda(s_j)$ if s_j are the shape functions of MLS. This gives a possible reason why many applications of MLS within meshless methods work this way, but note that here the weights are chosen in a special way and a specific polynomial is constructed first. This is by no means mandatory.

A fourth approach [6] views the bound locally around a fixed point z and inserts a Taylor expansion p_z of u around z, but does not use oversampled and thus uniformly bounded polynomial recovery like in [17] and [12]. In addition, this approach gives an indication of which weights could be useful. For $R_z(x) := u(x) - p_z(x)$, we have

$$|R_z(x)| \le \sum_{|\alpha|=m} \frac{|(x-z)^\alpha|}{\alpha!}\|\partial^\alpha u\|_{C(\Omega_z)}$$

on a local subdomain Ω_z containing z and X. Then we have the two error terms $|\lambda(R_z)|$ and

$$\begin{aligned} &\left|\sum_{j=1}^N a_j R_z(x_j)\right| \\ \le &\left|\sum_{|\alpha|=m} \frac{\|\partial^\alpha u\|_{C(\Omega_z)}}{\alpha!} \sum_{j=1}^N |a_j||(x_j-z)^\alpha|\right|. \end{aligned}$$

If no assumptions on the anisotropy of u and the point locations can be made, this bound suggests to take the weights $w_j := \|x_j - z\|_2^m$ for minimization of $\|\mathbf{a}\|_{1,w}$.

We stop the argument here and refer to [6] for details, and in particular, for the connection between this minimization problem and *growth functions*. In section 6 we shall come back to error bounds. In our examples, we denote the discretizations obtained by exactness of order m and minimization of $\|\mathbf{a}\|_{1,w}$ with $w_j := \|x_j - z\|_2^m$ *optimal m–th order local polynomial discretizations* and write $\|\mathbf{a}\|_{1,m}$ for short.

5 Direct Kernel–based Discretizations

We now go back to (8) and view direct discretizations in the most general way. Clearly, they make sense only if the point evaluations $u \mapsto u(x_j)$ and the functional evaluation $u \mapsto \lambda(u)$ are continuous operations. If we assume the functions u to lie in some Hilbert space $\mathcal{H}$ of functions on Ω with continuous point evaluations $\delta_x : u \mapsto u(x)$, the Riesz representers of the functionals δ_x define a *kernel* $K : \Omega \times \Omega \to \mathbb{R}$ with the properties

$$\begin{array}{rcll} K(x,y) &=& (\delta_x, \delta_y)_{\mathcal{H}^*} & \text{for all } x,y \in \Omega, \\ &=& (K(x,\cdot), K(y,\cdot))_{\mathcal{H}} & \text{for all } x,y \in \Omega, \\ f(x) &=& (f, K(x,\cdot))_{\mathcal{H}} & \text{for all } x \in \Omega,\ f \in \mathcal{H}, \\ (\lambda,\mu)_{\mathcal{H}^*} &=& \lambda^x \mu^y K(x,y) & \text{for all } \lambda,\mu \in \mathcal{H}^*, \end{array}$$

where λ^x acts with respect to the variable x. This is the setting of *Reproducing Kernel Hilbert Spaces* [2, 9], and it provides a general and simple construction of optimal direct discretizations for functionals $\lambda \in \mathcal{H}^*$. In fact,

$$|\epsilon_{\lambda,X,\mathbf{a}}(u)| \quad \le \quad \|\epsilon_{\lambda,X,\mathbf{a}}\|_{\mathcal{H}^*} \| \|u\|_{\mathcal{H}} \tag{12}$$

implies that optimal formulas should minimize

$$\begin{array}{rcl} \left\|\epsilon_{\lambda,X,\mathbf{a}}\right\|_{\mathcal{H}^*}^2 &=& \left\|\lambda - \sum_{j=1}^N a_j \delta_{x_j}\right\|_{\mathcal{H}^*}^2 \\ &=& (\lambda,\lambda)_{\mathcal{H}^*} - 2\sum_{j=1}^N a_j (\lambda, \delta_{x_j})_{\mathcal{H}^*} + \sum_{j,k=1}^N a_j a_k (\delta_{x_k}, \delta_{x_j})_{\mathcal{H}^*} \\ &=& \lambda^x \lambda^y K(x,y) - 2\sum_{j=1}^N a_j \lambda^x K(x,x_j) + \sum_{j,k=1}^N a_j a_k K(x_j,x_k) \end{array} \tag{13}$$

with respect to the coefficient vector $\mathbf{a}$. There are no weights here, no polynomials, and also no shape functions so far. Readers without a background in kernel–based techniques should consult [17] for details behind the arguments in this and the following section.

Equation (13) defines a positive semidefinite quadratic form, and the necessary and sufficient condition for a minimizer is the linear system

$$\sum_{j=1}^N a_j K(x_k, x_j) = \lambda^x K(x, x_k),\ 1 \le k \le N. \tag{14}$$

By Hilbert space projection arguments, this system is always solvable, and we get an *optimal kernel–based* direct discretization this way. If point evaluations for different points are always linearly independent, the form is positive definite and the solution is unique. On the downside, these optimal discretizations are usually non–sparse, and the evaluation of the values $\lambda^x K(x,x_k)$ may be costly in case of weak functionals.

In the Hilbert space setting, these discretizations are by construction optimal, and their error bound comes directly from (12) and (13). Note that (13) can be evaluated explicitly, thus leaving only $\|u\|_{\mathcal{H}}$ open in the error bound. In section 9 we shall evaluate errors this way on Sobolev spaces, and then we shall use the abbreviation $Q_S(\mathbf{a}) := \|\epsilon_{\lambda,X,\mathbf{a}}\|_{\mathcal{H}^*}^2$ to stand for the "Sobolev" quadratic form. Note that the *Whittle–Matérn* kernel

$$K(x,y) := \|x-y\|_2^{m-d/2} K_{m-d/2}(\|x-y\|_2),\ x,y \in \mathbb{R}^d$$

is reproducing in Sobolev space $W_2^m(\mathbb{R}^d)$ for $m > d/2$, where K_ν denotes the modified Bessel function of order ν, and the Sobolev quadratic form uses this kernel in (13).

Clearly, these discretizations are exact on the span of functions $K(\cdot,x_j),\ 1 \le j \le N$. It is well–known [18] that generalized Hermite–Birkhoff interpolation of trial functions u from this space is possible, i.e. one can find shape functions $s_1,\dots,s_N$ satisfying Lagrange conditions $\lambda_k(s_j) = \delta_{jk},\ 1 \le j,k \le N$. This implies that the coefficients $\lambda(s_j)$ also solve the optimality problem, and we get that this form of discretization can be written as the exact evaluation of the functional on the shape functions. However, the shape functions are not needed, but the quantities $\lambda^x K(x,x_k),\ 1 \le k \le N$ have to be calculated. This can be a highly nontrivial task if λ involves integration and if there are no integration formulas for functions of the form $K(\cdot,x)$. Solving PDEs in strong form is still convenient, because the kernels are usually easy to evaluate, but for weak problems it can be more efficient to go back to methods based on evaluation of polynomials. This makes a comparison of these techniques necessary, but we postpone a fully general efficiency analysis to future work. In section 9 we present a special case for MLPG5 [11].

We now show how to get sparsity in kernel–based discretizations. The system (14) implies that the calculation of an optimal discretization for a given functional λ in terms of values at a set X of nodes is equivalent to an interpolation of the function $f_\lambda := \lambda^x K(x,\cdot)$ on X by kernel translates $K(\cdot,x_j)$, $1 \le j \le N$. This can be done stepwise by choosing nodes one by one, by a greedy method from [16]. Another possibility for working locally around a point z is to order the nodes x_j with respect to their distance to z and then work only for the $n \le N$ nearest neighbors to z. We shall compare both approaches in section 9.

If users want discretizations on spaces of functions with a prescribed spectral behavior, one can usually invoke some kind of harmonic analysis and come out with a reproducing kernel Hilbert space that produces optimal discretizations. For instance, if univariate functions are band–limited, the appropriate kernel is a scaled sinc function. In general, $r^{-d/2}J_{d/2}(r)$ is the inverse Fourier transform of the characteristic function on the unit ball in $\mathbb{R}^d$. Spaces with algebraic decay of frequencies towards infinity are norm–equivalent to Sobolev spaces, and the appropriate kernels are of Whittle–Matérn or Wendland form. We shall provide examples in section 9.

6 Direct Discretizations in Beppo–Levi Spaces

Having the above machinery at hand, we consider another way of dealing with error bounds for polynomially exact discretizations with general weights. In case of (8), we can consider the error functional $\varepsilon_{\lambda,X,\mathbf{a}}$ of (10) on various spaces of functions. To deal with exactness on $\mathcal{P}^d_m$ via the standard Bramble–Hilbert technique, we should define a linear differential operator $L_m : \mathcal{H} \to \mathcal{F}$ such that $\mathcal{P}^d_m \subset \ker L_m \subset \mathcal{H}$ and $\lambda,\ \delta_{x_j} \in \mathcal{H}^*$. The standard way to define L_m is

$$L_m(u) = (u^{(\alpha)},\ |\alpha| = m)^T \in L_2(\Omega)^Q =: \mathcal{F}$$

arranged into a vector of functions that has length Q. Clearly, $\mathcal{F}$ is a Hilbert space, and we can define a semi–inner product on $\mathcal{H}$ by

$$(u,v)_{BL_m(\Omega)} := \sum_{|\alpha|=m} \frac{m!}{\alpha!}(u^{(\alpha)},v^{(\alpha)})_{L_2(\Omega)} = (L_m(u),L_m(v))_{L_2(\Omega)^Q}.$$

This yields a Beppo–Levi space [17, Definition 10.37] for $\mathcal{H}$ if $m > d/2$. It will have a conditionally positive semidefinite *polyharmonic* or *thin–plate spline* radial kernel

$$K_{m,d}(r) := \left\{ \begin{array}{cc} \frac{\Gamma(d/2-m)}{2^{2m}\pi^{d/2}(m-1)!} r^{2m-d} & d \text{ odd} \\ \frac{(-1)^{m+(d-2)/2}}{2^{2m-1}\pi^{d/2}(m-1)!(m-d/2)!} r^{2m-d}\log r & d \text{ even} \end{array} \right\} \tag{15}$$

of order m [17, (10.11)], and if the functional $\varepsilon_{\lambda,X,\mathbf{a}}$ is continuous on the Beppo–Levi space (it must be zero on $\mathcal{P}^d_m$ for that), it has an error bound

$$|\varepsilon_{\lambda,X,\mathbf{a}}(u)| \le Q_{BL}(\mathbf{a})|u|_{\mathcal{H}} = Q_{BL}(\mathbf{a})\|L_m u\|_{L_2}$$

with the quadratic form

$$Q^2_{BL}(\mathbf{a}) := \|\varepsilon_{\lambda,X,\mathbf{a}}\|^2_{\mathcal{H}^*} = \varepsilon^x_{\lambda,X,\mathbf{a}}\varepsilon^y_{\lambda,X,\mathbf{a}} K_{m,d}(x,y)$$

on the Beppo–Levi space. The best way to optimize $\mathbf{a}$ would then be the minimization of the above quadratic form under the constraint of polynomial exactness. If other coefficient vectors $\mathbf{a}$ come via other minimizations, they can be compared to the optimal formula in the Beppo–Levi space if they are exact on $\mathcal{P}^d_m$. We just evaluate $Q_{BL}(\mathbf{a})$ for them. Then users can decide whether a computationally effective formula based exclusively on polynomials is still competitive to the optimal formula based on polyharmonic splines. We shall do this in section 9. But since all formulas, not just the polynomially exact ones, can be checked for their performance on the Sobolev space $W^m_2(\mathbb{R}^d)$ instead of the Beppo–Levi space of order m, we shall evaluate $Q_S(\mathbf{a})$ as well, in order to see the performance in (12).

7 Approximation Orders

But before we turn to numerical examples, we should explain what happens if direct discretizations are used under scaling $h \to 0$ in the standard way of looking at finer and finer samplings. In case of polynomial exactness on $\mathcal{P}^d_m$ and the Beppo–Levi space error bounds, we can use a scaling argument of Bramble–Hilbert type for this purpose.

Assume that we work in a domain containing a ball around zero, and we scale the set X into hX with some $h \in (0,1]$. For fixed functions u in the Beppo–Levi–space of order m, we then consider the functions $u_h(x) := u(hx)$ and assume the functional λ to scale as

$$\lambda(u_h) = h^k \lambda(u).$$

This works for differential operators of order k, and weighted integrals over them.

If we take a function u on the full Beppo–Levi space of order m on $\Omega \subset \mathbb{R}^d$, we get

$$\begin{array}{rcl} |u_h|^2_{BL_m(\Omega)} & = & \|L_m u_h\|^2_{L_2(\Omega)^Q} \\ & = & h^{2m}\|(L_m u)(h\cdot)\|^2_{L_2(\Omega)^Q} \\ & = & h^{2m-d}\|L_m u\|^2_{L_2(h\Omega)^Q} \\ & \le & h^{2m-d}\|L_m u\|^2_{L_2(\Omega)^Q} \\ & = & h^{2m-d}|u|^2_{BL_m(\Omega)} \end{array} \tag{16}$$

which does a poor job of localization that we shall comment on later.

The scaling of the weights for working on hX should be $h^{-k}\mathbf{a}$ and we get

$$\begin{array}{rcl} |\varepsilon_{\lambda,hX,h^{-k}\mathbf{a}}(u)| & = & \left|\lambda(u)-\sum_{k=1}^{N} h^{-k}a_k u(hx_k)\right| \\ & = & h^{-k}\left|\lambda(u_h)-\sum_{k=1}^{N} a_k u_h(x_k)\right| \\ & = & h^{-k}|\varepsilon_{\lambda,X,\mathbf{a}}(u_h)| \\ & \leq & h^{-k}\|\varepsilon_{\lambda,X,\mathbf{a}}\|_{\mathcal{H}^*}|u_h|_{BL_m(\Omega)} \\ & \leq & h^{m-k-d/2}\|\varepsilon_{\lambda,X,\mathbf{a}}\|_{\mathcal{H}^*}|u|_{BL_m(\Omega)} \end{array} \tag{17}$$

proving less than the expected approximation order h^{m-k}. All discretizations that are exact on $\mathcal{P}_m^d$ will have at least this asymptotic behavior, no matter if they are based on polynomials or not. Comparison of such formulae can be done via $Q^2_{BL}(\mathbf{a})=\|\varepsilon_{\lambda,X,\mathbf{a}}\|^2_{\mathcal{H}^*}$ on a *fixed* scale.

Readers will expect an order h^{m-k} in (17). First, our numerical experiments in section 9 will show that $\|\varepsilon_{\lambda,X,\mathbf{a}}\|^2_{\mathcal{H}^*}$ actually behaves not better than $\mathcal{O}(h^{m-k-d/2})$. Second, the well–known arguments along local Taylor formulas show that proofs of $\mathcal{O}(h^{m-k})$ convergence need a strongly localized norm of the functions to be discretized, or simply a condition like $u\in C^m$ in a neighborhood of z. Looking at (16) tells us that we should have used

$$\|(L_m u)(h\cdot)\|^2_{L_2(\Omega)^Q} = \sum_{|\alpha|=m}\frac{m!}{\alpha!}\int_\Omega (u^\alpha(hx))^2dx \tag{18}$$

which is bounded independent of h as soon as u has bounded derivatives of order m locally around zero. This proves that we would see $\mathcal{O}(h^{m-k})$ convergence of $|\varepsilon_{\lambda,hX,h^{-k}\mathbf{a}}(u)|$ for $u\in C^m(\Omega)$.

But we want to compare errors in Hilbert or Beppo–Levi spaces, not in $C^m(\Omega)$. Therefore we look for a properly scaled and localized version that shows the expected convergence. The associated inner product to (18) is

$$(u,v)_{m,h}:=\sum_{|\alpha|=m}\frac{m!}{\alpha!}\int_\Omega u^{(\alpha)}(hx)v^{(\alpha)}(hx)dx=h^{-d}(u,v)_{BL_m(h\Omega)}$$

which suggests that we have a scaled version of a Beppo–Levi space on Ω_h here, and we want to construct the reproducing kernel. Note that the factor h^{-d} above seems to care for the volume of $h\Omega$.

We start from the polyharmonic kernel $K_{m,d}$ for order m on $\Omega\subset\mathbb{R}^d$. With a projector Π_m onto the polynomials on $\mathcal{P}_m^d$ we have

$$f(x)-(\Pi_m(f))(x)=(f,G_{m,d}(\cdot,x))_{BL_m(\Omega)} \text{ for all } f\in\mathcal{H}$$

due to [17, Theorem 10.17, p. 144], where

$$G_{m,d}(\cdot,x)=K_{m,d}(\cdot,x)-(\Pi_m^y K_{m,d}(\cdot,y))(x).$$

We apply this for $f:=u(h\cdot)$ to get

$$\begin{array}{rcl} u(hx)-\Pi_m(u(h\cdot))(x) & = & (u(h\cdot),G_{m,d}(\cdot,x))_{BL_m(\Omega)} \\ & = & h^m\sum_{|\alpha|=m}\frac{m!}{\alpha!}\int_\Omega u^{(\alpha)}(hy)G^{\alpha,y}_{m,d}(y,x)dy \\ & = & \sum_{|\alpha|=m}\frac{m!}{\alpha!}\int_\Omega u^{(\alpha)}(hy)G^{\alpha,y}_{m,d.h}(hy,hx)dy \\ & = & (u,G_{m,d,h}(\cdot,hx))_{m,h} \\ & = & u(hx)-(\Pi_{m,h}(u))(hx) \end{array}$$

if we have

$$h^mG^{\alpha,y}_{m,d}(y,x)=G^{\alpha,y}_{m,d.h}(hy,hx)$$

and define the projector

$$(\Pi_{m,h}(u))(y):=\Pi_m(u(h\cdot))(y/h).$$

Altogether, this is a localized reproduction equation

$$u(z)-(\Pi_{m,h}u)(z)=(u,G_{m,d,h}(\cdot,z))_{m,h}$$

that we were looking for. To find the kernel, we start from

$$h^mG^{\alpha,y}_{m,d}(y,x) = h^mK^{\alpha,y}_{m,d}(y,x)-h^m(\Pi_m^zK^{\alpha,z}_{m,d}(y,z))(x)$$

and define

$$K_{m,d,h}(y,x) = h^{2m}K_{m,d}(y/h,x/h)$$

to get

$$\begin{array}{rcl}
K^{\alpha,y}_{m,d,h}(y,x) & = & h^m K^{\alpha,y}_{m,d}(y/h,x/h)\\
K^{\alpha,y}_{m,d,h}(hy,hx) & = & h^m K^{\alpha,y}_{m,d}(y,x)\\
h^m G^{\alpha,y}_{m,d}(y,x) & = & K^{\alpha,y}_{m,d,h}(hy,hx) - h^m(\Pi^z_m K^{\alpha,y}_{m,d}(y,z))(x),\\
 & = & K^{\alpha,y}_{m,d,h}(hy,hx) - (\Pi^z_m K^{\alpha,y}_{m,d,h}(hy,hz))(x)\\
 & = & K^{\alpha,y}_{m,d,h}(hy,hx) - (\Pi^w_{m,h} K^{\alpha,y}_{m,d,h}(hy,w))(hx)\\
 & = & G^{\alpha,y}_{m,d.h}(hy,hx).
\end{array}$$

This means the we can take

$$G_{m,d.h}(y,x) \quad = \quad h^{2m}K_{m,d}(y/h,x/h) - (\Pi^w_{m,h}K_{m,d,h}(y,w))(x)$$

modulo a polynomial in y with coefficients in x. Looking back at (15), we see that in odd dimensions we just have to multiply the kernel with h^d. In even dimensions, the kernel $r^{2m-d}\log r$ is to be replaced by $h^d r^{2m-d}(\log r - \log h)$, which is a multiplication with h^d and an addition of a polynomial that cancels out whenever we calculate the quadratic form. We could call this the *scaled* Beppo–Levi kernel, but as long as we compare quadratic forms, it just has a corrected h^d factor to guarantee the expected convergence rate. This is why we can ignore this detour altogether, knowing now that this change of the norm would result in the expected approximation order.

For nonpolynomial discretizations without exactness on polynomials, we can proceed similarly and use their optimality. If we work on hX, we get optimal coefficients $\mathbf{a}^*(h)$ that we can compare to $h^{-k}\hat{\mathbf{a}}$ for any fixed $\mathcal{P}^d_m$–exact discretization with coefficients $\hat{\mathbf{a}}$ for work on X, following

$$\begin{array}{rcl}
\|\epsilon_{\lambda,hX,\mathbf{a}^*(h)}\|^2_{\mathcal{H}^*} & \leq & \|\epsilon_{\lambda,hX,h^{-k}\hat{\mathbf{a}}}\|^2_{\mathcal{H}^*}\\
 & = & \epsilon^u_{\lambda,hX,h^{-k}\hat{\mathbf{a}}}\epsilon^v_{\lambda,hX,h^{-k}\hat{\mathbf{a}}}K(u,v).
\end{array}$$

If the error bound (17) is applied here, this quantity will behave like h^{2m-2k} times a constant that depends on K, provided that the kernel K is smooth enough. This proves that kernel–based optimal discretizations at least attain the maximal convergence order that is possible on a set X for polynomially exact formulae. They do this without being exact on polynomials, but the price to be paid is that they will usually be based on the whole set X, without sparsity, and they need evaluation of λ on the functions $K(\cdot,x_j)$ which can be expensive in case of weak functionals.

8 Sparsity First

For solving the system (9), sparsity of the coefficient matrix will be a highly important issue. Thus we shall now focus on the problem how to get the smallest error for a given sparsity, i.e. for the number N of point locations being small and the points themselves properly selected. We first ignore the point selection problem and assume that we take the first N points from a larger set of candidates, with N slowly increasing.

For polynomially exact formulas, it is reasonable to go for the maximal order of polynomial exactness. If the points are in general position with respect to a fixed order m in $\mathbb{R}^d$, there is only one polynomially exact discretization and we have no leeway for further optimization of coefficients. If the number of points is increased somewhat, but not enough to go for the next higher order of polynomial exactness, there is a formula with the minimal number of nonzero coefficients, namely the one we get when ignoring the additional points. We call this the *greedy* $\|.\|_0$ solution. But we can also minimize $\|a\|_1$ or $\|a\|_{1,m}$ under the constraint of exactness of order m. If N is increased further, these solutions will coincide as soon as the next higher order of exactness is reached. In parallel, one can also calculate kernel–based discretizations at these points, and we know that they will also lead to maximal orders of approximation, but without polynomial exactness, provided that we use smooth kernels. This is why we should look at experiments with increasing N, letting all possible candidates compete, while the polynomially exact formulae always go for the maximal order of exactness. Comparisons should be made by checking the norm of the error functional on a fixed –order Sobolev space. We shall provide such examples in the next section.

But we still have to discuss how to select points effectively. If working locally around a point z, one can take neighbors of z with increasing distance. Though this may be unstable if two points are very close to each other, it turns out to be quite a good strategy for all formulas in the competition so far.

There is a method [16] that chooses useful interpolation points adaptively, and it can be applied here because (14) shows that the discretization problem is just interpolation of the function $\lambda^x K(x,\cdot)$. We shall compare it to the nearest–neighbor choice, but it performs worse since it tends to prefer points further away from z.

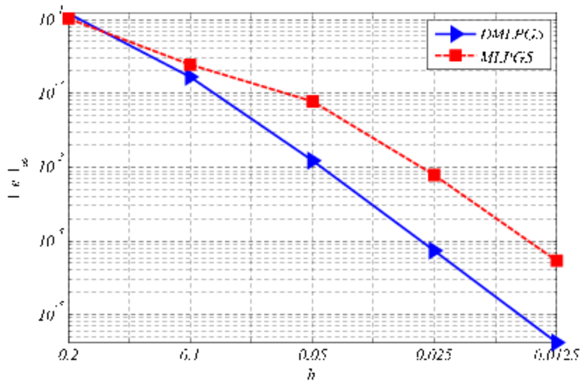

Figure 1: Comparison of MLPG5 and DMLPG5 in terms of maximum errors for $m = 4$.

9 Numerical Examples

For examples that show the gain in computational efficiency by replacing standard Moving Least Squares discretizations by direct discretizations via polynomials, we refer to [11, 10].

From [11] we take an example for variation 5 of the Meshless Local Petrov–Galerkin (MLPG5) method (7) in comparison to the DMLPG5 method, i.e. the standard versus the direct discretizations of normal derivatives along edges of squares in $\mathbb{R}^2$. The overall setting is a standard inhomogeneous Poisson problem on $[0,1]^2$ with Dirichlet boundary conditions and Franke's function [7] as known smooth solution. Discretization was done via Moving Least Squares in the MLPG5 method, while a direct discretization with the same weights was used in DMLPG5. For $0 \le \|x - x_j\|_2 \le \delta$, the MLS used the truncated Gaussian weight function

$$w_\delta(x, x_j) = \frac{\exp\left(-(\|x-x_j\|_2/c)^2\right) - \exp\left(-(\delta/c)^2\right)}{1 - \exp\left(-(\delta/c)^2\right)}$$

where $c = c_0 h$ is a constant controlling the shape of the weight function and $\delta = \delta_0 h$ is the size of the support domains. The parameters $m = 4$, $c_0 = 0.8$ and $\delta_0 = 2m$ were selected. With direct discretizations in DMLPG5, a 2-point Gaussian quadrature is enough to get exact numerical integration. But for MLPG5 and the right hand sides we used a 10-point Gaussian quadrature for every edge of the squares to ensure sufficiently small integration errors. The results are depicted in Table 1 and Fig. 1. DMLPG is more accurate and approximately gives the full order $m = 4$ in this case. Note that we have $k = 1$ here, but we integrate over a line of length h. Besides, as is to be expected, the computational cost of DMLPG is remarkably less than for MLPG.

	$\|e\|_\infty$		$\|e\|_\infty$		CPU sec.	CPU sec.
h	MLPG5	rate	DMLPG5	rate	MLPG5	DMLPG5
0.2	$0.10 \times 10^{\pm 0}$	–	$0.12 \times 10^{\pm 0}$	–	0.5	0.2
0.1	0.25×10^{-1}	2.04	0.17×10^{-1}	2.87	2.7	0.2
0.05	0.78×10^{-2}	1.66	0.12×10^{-2}	3.75	19.2	0.9
0.025	0.79×10^{-3}	3.30	0.75×10^{-4}	4.04	142.2	4.7
0.0125	0.55×10^{-4}	3.86	0.43×10^{-5}	4.12	2604.9	43.9

Table 1: Results for MLPG5 methods

For the remaining examples, we focus on direct discretizations of the Laplacian $\lambda(u) := \Delta u(0)$ in $\mathbb{R}^2$. Since the five–point discretization is exact on $\mathcal{P}^2_4$ with dimension $Q = 10$, we choose $m = 4$. This leads to the Beppo–Levi space generated by the thin–plate spline $r^6 \log r$ and to Sobolev space $W^4_2(\mathbb{R}^2)$ with kernel $r^3 K_3(r)$. We want some additional degrees of freedom in case of scattered data, so we take $N = 27$ fixed points in $[-1,1]^2$ that include the five–point discretization at stepsize 1 and are in Figure 2. Clearly, the

Figure 2: 27 points for discretization of Laplacian at zero

sparsest discretization which is exact on $\mathcal{P}^2_4$ uses 5 points with coefficient norm $\|\mathbf{a}\|_1 = 8$, but other discretizations will have between $Q = 10$ and $N = 27$ points.

The rows of Tables 2 and 3 contain various discretization methods. They start with the discretization OMP calculated by Orthogonal Matching Pursuit along the lines of section 4.1, followed by what the backslash operator in MATLAB produces. The next columns are minimization of $\|\mathbf{a}\|_1$ and $\|\mathbf{a}\|_{1,m}$, respectively, followed by the generalized Moving Least Squares (GMLS) solution of [12] with

	$\|\mathbf{a}\|_0$	$\|\mathbf{a}\|_1$	$\|\mathbf{a}\|_{1,4}$	$Q_{BL}(\mathbf{a})$	$Q_S(\mathbf{a})$
OMP	10	9.539	4.516	10.271	1.235
MATLAB	5	8.000	4.000	10.180	1.229
min $\|\mathbf{a}\|_1$	5	8.000	4.000	10.180	1.229
min $\|\mathbf{a}\|_{1,4}$	10	32.658	1.681	6.162	0.801
GMLS	27	18.183	2.533	7.799	0.988
min $Q_{BL}(\mathbf{a})$	27	264.145	82.249	3.083	0.435
min $Q_{BLF}(\mathbf{a})$	27	146.112	40.997	3.582	0.506
min $Q_S(\mathbf{a})$	27	271.955	84.131	3.131	0.431

Table 2: Results for $m = 4$ on 27 points

	$\|\mathbf{a}\|_0$	$\|\mathbf{a}\|_1$	$\|\mathbf{a}\|_{1,4}$	$Q_{BL}(\mathbf{a})$	$Q_S(\mathbf{a})$
OMP	21	101.452	25.837	12.036	0.145
MATLAB	21	62.579	11.886	10.493	0.131
min $\|\mathbf{a}\|_1$	21	59.061	9.932	9.515	0.120
min $\|\mathbf{a}\|_{1,4}$	21	72.876	8.987	7.089	0.091
GMLS	27	70.798	11.312	9.090	0.114
min $Q_{BL}(\mathbf{a})$	27	223.053	68.925	4.202	0.056
min $Q_{BLF}(\mathbf{a})$	27	146.112	32.763	6.109	0.067
min $Q_S(\mathbf{a})$	27	246.868	78.200	5.720	0.047

Table 3: Results for $m = 6$ on 27 points

global Gaussian weights $\exp(-3\|x-x_j\|_2^2)$. The next rows are based on minimizing quadratic forms: Q_{BL} on the Beppo–Levi space for $\mathcal{P}_m^d$–exact discretizations, while Q_{BLF} for bandlimited functions and Q_S for Sobolev space are minimized without any exactness.

The columns contain comparison criteria. The first column counts the essentially nonzero coefficients, and the others are self–explaining. The final column has no polynomial exactness, and thus its evaluation in Q_{BL} is not supported by any theory, while the methods of all other columns are competing for the optimal error in the final column.

Table 2 shows the results for $m = 4$ and 27 points. Rows 2 and 3 pick the five–point discretization. Minimizing the quadratic forms yields optimal formulae in the associated spaces, but their coefficients get rather large and the formulae use all available degrees of freedom. If two entries in one of the final two columns differ by a factor of four, the convergence like $h^{m-k} = h^2$ implies that the same accuracy will be reached when one of the methods uses $h/2$ instead of h. In 2D this means that one method needs four times as many unknowns as the other to get similar accuracy. If no sophisticated solvers are at hand, this means a serious increase in computational complexity.

Surprisingly, all formulae are reasonably good in Sobolev space, showing that one can use formulae based on polynomial exactness and sparsity without much loss.

The maximal possible polynomial exactness order for the same points is $m = 6$, and the corresponding results are in Table 3. This corresponds to the kernels $r^{10}\log r$ and $r^5K_5(r)$ for the Beppo–Levi and Sobolev spaces. The polynomial formulae lose their sparsity advantage, and the weight norms of all discretizations do not differ dramatically. Note that the h^4 convergence now implies that a method can equalize a factor of 16 in the above numbers by going from h to $h/2$. Again, the formulae with polynomial exactness behave quite well in Sobolev and Beppo–Levi spaces. Since for weak functionals they will be much cheaper to evaluate, it may not make much sense to go for Hilbert or Beppo-Levi space optimality in that case. But the latter will work fine in strong discretizations, and in particular if users fix the number of neighboring nodes to be considered. This will fix the bandwidth of the system, and each functional will get an optimal discretization within the admitted bandwidth, disregarding exactness on polynomials.

We now compare discretizations of the Laplacian at zero via various kernels, but without polynomial exactness. We take $N = 14$ fixed points in $[-1,1]^2$ including the five–point discretization points at stepsize $h = 1$, and we take different discretizations there, constructed with $m = 4$ in mind whenever it makes sense, and taking hX for $h \to 0$. Then we measure errors on $W_2^4(\mathbb{R}^2)$ and $W_2^6(\mathbb{R}^2)$ to see how serious it is to choose "wrong" kernels. All discretizations will be of order h^2 in $W_2^6(\mathbb{R}^2)$ and of order h in $W_2^4(\mathbb{R}^2)$, except the optimal formula of $W_2^4(\mathbb{R}^2)$ evaluated in $W_2^6(\mathbb{R}^2)$. Figures 3 and 4 show the behavior of these errors for $h \to 0$, evaluated as $\|\varepsilon_{\lambda,hX,\mathbf{a}^*(h)}\|$. The results are another example confirming that excess smoothness does no damage error–wise, but it increases instability of calculation. The formulas based on smooth kernels always seem to reach the optimal order in a specific space of functions, but they never show some superconvergence there. They just adapt nicely to the smoothness of the functions in the underlying space, if the kernel is smooth enough.

We finally present examples with increasing N. If we use nearest neighbors, work for finally 27 points as in Figure 2 and evaluate the error in W_2^6, we get Figure 5 for a single run and Figure 6 for the means of 24 sample runs with 27 random points in $[-1,1]^2$ each. Before taking means, we divided the errors by the minimal error obtainable by using all points, i.e. the level of the bottom line in Figure 5. One can clearly see that the minimization of $\|a\|_{1,m}$ performs best among the polynomial methods. We added the optimal kernel–based formulae in W_2^6 for both the nearest neighbor choice and the greedy choice of [16] to see that the nearest neighbor choice is fine for $N \geq 5$. Note that the sparsity increases roughly linearly with N, because even the greedy $\|.\|_0$ solution needs 3, 6, 10, 15, and 21 points when stepping through orders 2,3,4, 5, and 6.

Acknowledgement: Special thanks go to Oleg Davydov and Davoud Mirzaei for some helpful comments on an earlier version of this article, and to a referee for a very thorough and helpful report.

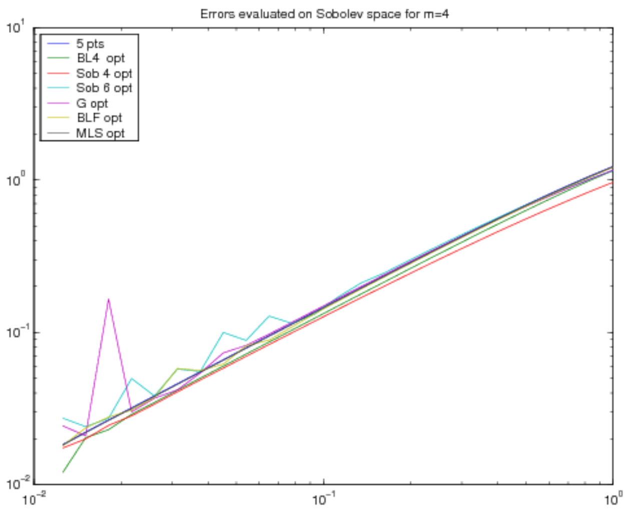

Figure 3: Errors in $W_2^4(\mathbb{R}^2)$ as functions of h

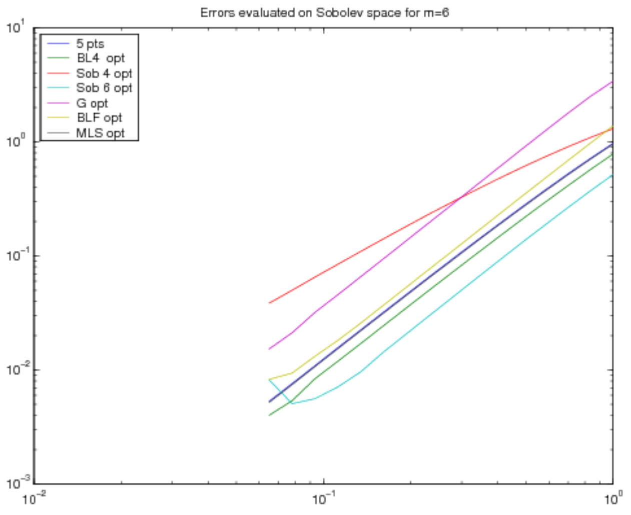

Figure 4: Errors in $W_2^6(\mathbb{R}^2)$ as functions of h

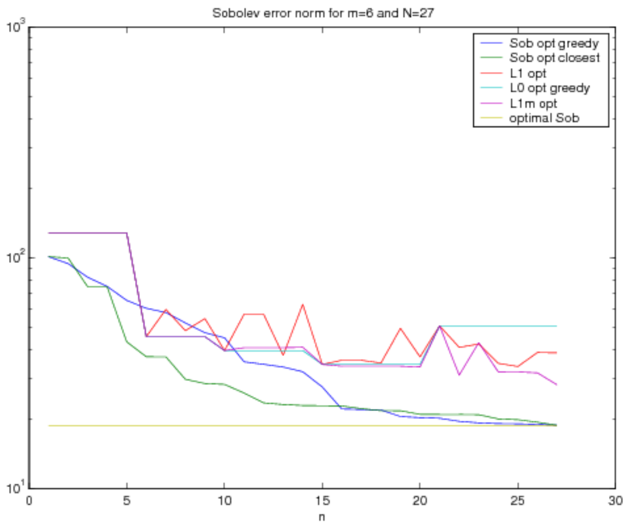

Figure 5: Errors in $W_2^6(\mathbb{R}^2)$ as functions of N

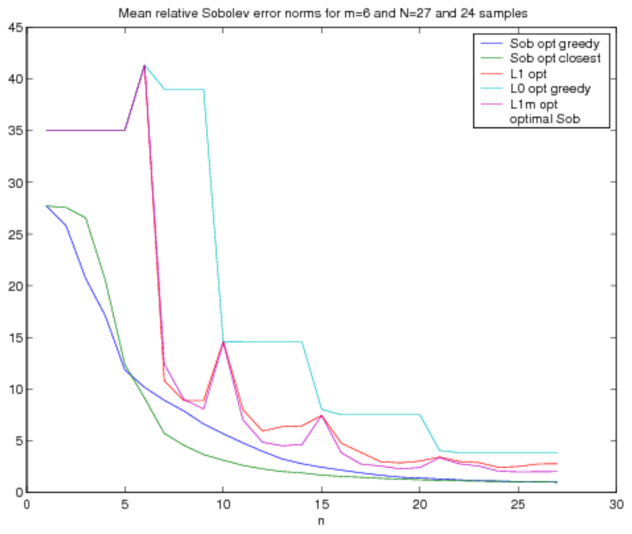

Figure 6: Errors in $W_2^6(\mathbb{R}^2)$ as functions of N

References

[1] M. G. Armentano. Error estimates in Sobolev spaces for moving least square approximations. *SIAM J. Numer. Anal.*, 39(1):38–51, 2001.

[2] N. Aronszajn. Theory of reproducing kernels. *Trans. Amer. Math. Soc.*, 68:337–404, 1950.

[3] S. N. Atluri. *The meshless method (MLPG) for domain and BIE discretizations*. Tech Science Press, Encino, CA, 2005.

[4] T. Belytschko, Y. Krongauz, D.J. Organ, M. Fleming, and P. Krysl. Meshless methods: an overview and recent developments. *Computer Methods in Applied Mechanics and Engineering, special issue*, 139:3–47, 1996.

[5] O. Davydov. Error bound for radial basis interpolation in terms of a growth function. In A. Cohen, J. L. Merrien, and L. L. Schumaker, editors, *Curve and Surface Fitting: Avignon 2006*, pages 121–130. Nashboro Press, Brentwood, 2007.

[6] O. Davydov and R. Schaback. Error bounds for kernel-based numerical differentiation. Draft, 2013.

[7] R. Franke. Scattered data interpolation: tests of some methods. *Mathematics of Computation*, 48:181–200, 1982.

[8] Y.C. Hon and T. Wei. Numerical differentiation by radial basis functions approximation. *Advances in Computational Mathematics*, 27:247–272, 2007.

[9] H. Meschkowski. *Hilbertsche Räume mit Kernfunktion*. Springer, Berlin, 1962.

[10] D. Mirzaei and R. Schaback. Solving heat conduction problems by the direct meshless local Petrov-Galerkin (DMLPG) method. Preprint Göttingen, 2012.

[11] D. Mirzaei and R. Schaback. Direct Meshless Local Petrov-Galerkin (DMLPG) method: A generalized MLS approximation. *Applied Numerical Mathematics*, 68:73–82, 2013. http://dx.doi.org/10.1016/j.apnum.2013.01.002.

[12] D. Mirzaei, R. Schaback, and M. Dehghan. On generalized moving least squares and diffuse derivatives. IMA Journal of Numerical Analysis 2011, doi: 10.1093/imanum/drr030, 2011.

[13] B. Nyroles, G. Touzot, and P. Villon. Generalizing the finite element method: Diffuse approximation and diffuse elements. *Comput. Mech.*, 10:307–318, 1992.

[14] C. Prax, H. Sadat, and P. Salagnac. Diffuse approximation method for solving natural convection in porous media. *Transport in Porous Media*, 22:215–223, 1996.

[15] R. Schaback. An adaptive numerical solution of MFS systems. In C.S. Chen, A. Karageorghis, and Y.S. Smyrlis, editors, *The Method of Fundamental Solutions - A Meshless Method*, pages 1–27. Dynamic Publishers, 2008.

[16] R. Schaback and H. Wendland. Adaptive greedy techniques for approximate solution of large RBF systems. *Numer. Algorithms*, 24(3):239–254, 2000.

[17] H. Wendland. *Scattered Data Approximation*. Cambridge University Press, 2005.

[18] Z. Wu. Hermite–Birkhoff interpolation of scattered data by radial basis functions. *Approximation Theory and its Applications*, 8/2:1–10, 1992.