Applied Mathematical Sciences, Vol. 8, 2014, no. 164, 8233 - 8250
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/ams.2014.49739

The DMLPG Meshless Technique for Poisson Problems

Flavio Sartoretto

DAIS, Università Ca' Foscari Venezia
Via Torino 155, 10173 Venezia, Italy

Annamaria Mazzia and Giorgio Pini

Dipartimento ICEA Università di Padova
Via Trieste 63, 35121 Padova, Italy

Copyright © 2014 Flavio Sartoretto, Annamaria Mazzia and Giorgio Pini. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract

Meshless Local Petrov Galerkin (MLPG) techniques are pure meshless methods nowadays used to solve a large class of Partial Differential Equations. Recently, the Direct MLPG (DMLPG) techniques have been proposed. They are based upon the Generalized Least Square Method. DMLPG technique alleviates some difficulties found in MLPG, such as tricky numerical integration. In this article we report our expansion of DMLPG techniques to 3D Poisson problems. We also compare MLPG vs DMLPG performances.

Keywords: meshless methods, Poisson problem, generalized moving least squares, radial basis functions, tensor product functions

1 Introduction

This paper deals with *true* meshless methods for the solution of Partial Diffeential Equations (PDE). They do not use any mesh, neither for discretization of the problem domain, nor for numerical integrations. True meshless methods are designed to reduce the high computational cost spent in order to design discretization mesh for complex geometries. They also facilitate the implementation of adaptive techniques.

Meshless Local Petrov Galerkin (MLPG) methods (see eg. [2]) are among the most frequently exploited. In the literature there are many implementations based on radial basis functions (RBF) [6, 25], see eg [1], and references herein. A smaller number of contributes which use also tensor product functions (TPF), are available [22, 7, 28, 14, 26, 18].

The advantages of the latter methods are particularly dominant when 3D problems are considered, see [17] and references herein.

Recently, new MLPG methods have been proposed, called Direct MLPG (DMLPG) [19], that take advantage of the Generalized Moving Least Squares method (GMLS) [20]. While the Moving Least Squares (MLS) technique, proposed in [11] aims to approximate a multivariate function, GMLS approximates linear functionals. The solution of a linear PDE either in strong or weak form, can be calculated by GMLS approximation of the functional involved. While MLPG approximates the PDE solution by projection on a finite dimensional space, DMLPG approximates the functionals involved in the formulation of the problem. In other words, MLPG generalizes Finite Element (FE) Methods, while DMLPG generalizes Finite Difference (FD) techniques.

We have extended DMLPG from 2D to 3D problems; a not straightforward task. It requires handling sparse data structures, and identify appropriate techniques in order to solve the final linear system.

Non–standard numerical methods for the solution of differential equations are often based on heuristics and ideas verified by numerical experiments [3]. Theoretical convergence analysis of DMLPG and MLPG methods are not so mature as for FE, then we analyze their advantages and disadvantages by numerical experiments.

2 Meshless techniques

Let us consider the linear Poisson equation on the domain Ω

$$-\nabla^2 u(\mathbf{x}) = f(\mathbf{x}), \tag{1}$$

where f is a given source function; $\mathbf{x}$ is any point in Ω. Dirichlet and Neumann boundary conditions are imposed on the boundary $\partial\Omega$

$$u = \bar{u} \quad \text{on } \Gamma_u, \quad \frac{\partial u}{\partial \mathbf{n}} \equiv q = \bar{q} \quad \text{on } \Gamma_q \tag{2}$$

Γ_u is the Dirichlet piece of boundary, Γ_q the Neumann piece of boundary. We assume $\partial\Omega = \Gamma = \Gamma_u \cup \Gamma_q$, $\Gamma_u \cap \Gamma_q = \emptyset$. The vector $\mathbf{n}$ denotes the outward normal direction to Γ. Moreover, $\bar{u}$ is the prescribed potential, and $\bar{q}$ the given normal flux.

Applying the divergence theorem after multiplication of the residual by a suitable test function, we obtain the weak formulation for (1)

$$\int_\Omega \nabla u \cdot \nabla \tau d\Omega - \int_\Gamma (\nabla u \cdot \mathbf{n})\, \tau d\Gamma = \int_\Omega f\, \tau\, d\Omega, \quad \forall \tau \in \mathcal{S}, \tag{3}$$

There are many Meshless Petrov–Galerkin (MLPG) methods [1, 9], each one can be identified by an appropriate choice of trial and test functions.

In order to approximate the solution of our weak formulation, a set of discretization nodes must be given. Let N be the total number of nodes.

In order to compute an approximation $\tilde{u} = \sum_i \tilde{u}_i \xi_i$, a set of trial functions, ξ_i, is enrolled, each one being "centered" on node $\mathbf{x}_i$. The support of ξ_i, S_{ξ_i} is a ball centered at $\mathbf{x}_i$, whose radius is r_i.

A finite set of test functions τ_i, $i = 1, \ldots, N$ is elected. The support of τ_i, $S_{\tau_i} = \Omega_i$, is a cuboid "centered" at $\mathbf{x}_i$, whose half side–length is ρ_i. We assume $\tau_i = 0$ outside Ω, and $\tau_i = 0$ on $\partial\Omega_i$, a typical setting in many MLPG schemes [19, 18]. In principle the test functions can be centered on nodes which do not coincide with the $\mathbf{x}_i$, but for simplicity we exploit the same set of nodes, both for the trial and the test functions.

A set of Local Weak Forms (LWF) is obtained by writing eq. (3) for each test function

$$\int_{\Omega_i} \nabla u \cdot \nabla \tau_i d\Omega - \int_{\Gamma_i^{(u)}} (\nabla u \cdot \mathbf{n})\, \tau_i d\Gamma = \int_{\Omega_i} f\, \tau_i\, d\Omega + \int_{\Gamma_i^{(q)}} (\nabla u \cdot \mathbf{n})\, \tau_i d\Gamma, \tag{4}$$

where $\Gamma_i^{(u)} = \partial\Omega_i \cap \partial\Gamma_u$ is the intersection of our local integration domain boundary with *Dirichlet* boundary. Moreover, $\Gamma_i^{(q)} = \partial\Omega_i \cap \partial\Gamma_q$ is the intersection of our local integration domain boundary with *Neumann* boundary. Integrals on $\Gamma_i = \partial\Omega_i \backslash (\Gamma_i^{(u)} \cup \Gamma_i^{(q)})$, the portion of $\partial\Omega_i$ lying inside Γ, give null contribution, being $\tau_i = 0$ on $\partial\Omega_i$. Boundary conditions are treated by exploiting suitable techniques [1, 17].

2.1 Approximation schemes

The Moving Least Squares (MLS) method [11] was proposed for approximating a function, $u(\mathbf{x})$, inside a region $\Omega \subset \mathbb{R}^d$ after a number, N, of its values, $u(\mathbf{x}_i)$, $\mathbf{x}_i \in \Omega$, are given. In the context of MLPG methods, in order to approximate a given d-dimensional problem, MLS is exploited for generating a set of d-dimensional trial functions on the ground of a suitable "weight" function [17, 18]. The ensuing trial functions are called the "shape" functions of MLS.

The Generalized Moving Least Squares (GMLS) method [20], is a technique developed after MLS formulation in [12], for approximating continuous linear functionals in the dual of $\mathcal{C}^k(\Omega)$, for any $k \geq 0$. GMLS aims to approximate a linear functional, $\lambda(u)$, based upon a given set of N linear functionals $\lambda_i(u)$. One obtains an approximation

$$\widehat{\lambda(u)} = \sum_{i=1}^{N} \phi_i(\lambda)\, \lambda_i(u). \tag{5}$$

Each coefficient $\phi_i(\lambda)$, called a GMLS shape function, must be linear in λ. As an example, $\lambda(u)$ can be a partial derivative of u, while $\lambda_i(u) = u(\mathbf{x}_i)$ can be a given set of u values on given nodes $\mathbf{x}_i \in \Omega$; $\widehat{\lambda(u)}$ in this case can be interpreted as a generalization of Finite Difference methods, based upon all $u(\mathbf{x}_i)$ values, in place of a given stencil.

Both methods are based upon approximating on the space of polynomials of degree up to a given value δ. Usually either $\delta = 2$ (quadratic polynomials) or $\delta = 3$ (cubic polynomials) are exploited.

For more details, see e.g. [12, 19].

2.2 Discretizing equations

In the sequel, following [18], we consider MLPG methods where: (a) the trial functions are the MLS shape functions generated by suitable Radial Basis

Functions (RBF); (b) the test functions are either RBF, or Tensor Product Functions (TPF).

Concerning DMLPG, it is obtained by applying GMLS to the weak problem (4). We identify some linear functionals, to be approximated via GMLS, by using the point functionals $\lambda_i(u) = u(\mathbf{x}_i)$.

For convenience we divide the discretization nodes into three subsets, i.e. the nodes $\mathbf{x}_j$, falling inside Ω, the set of nodes $\mathbf{x}_k \in \Gamma_u$, the set of nodes $\mathbf{x}_l \in \Gamma_q$. For each internal node $\mathbf{x}_j$, the left-hand side of the weak form (4)

$$\mu_j(u) \equiv \int_{\Omega_j} \nabla u \cdot \nabla \tau_j d\Omega - \int_{\Gamma_j^{(u)}} (\nabla u \cdot \mathbf{n})\, \tau_j d\Gamma,$$

is the functional to be approximated, while the known term is

$$\beta_j \equiv \int_{\Omega_j} f\, \tau_j\, d\Omega + \int_{\Gamma_j^{(q)}} (\nabla u \cdot \mathbf{n})\, \tau_j d\Gamma,$$

Following GMLS steps, the ensuing approximation is

$$\widehat{\mu_j(u)} = \sum_i \phi_i(\mu_j) u(\mathbf{x}_i).$$

The j-th DMLPG approximated equation is

$$\widehat{\mu_j(u)} = \beta_j.$$

The weight functions are the same compact supported RBF used in MLS.

Note that $\mu_j(\mathbf{p})$ involves evaluating the weak form where the integrating functions are products whose factors are test functions (or their partial derivatives), and polynomials, whereas in MLPG the factors corresponding to the latter ones are more complex MLS shape functions, and their partial derivatives.

Dirichlet boundary conditions can be directly imposed as $\lambda_k(u) = u(\mathbf{x}_k) = \bar{u}(\mathbf{x}_k)$, $\mathbf{x}_k \in \Gamma_k$. Such approach is a simplification over using MLPG techniques, where the unknowns are the so called "fictitious" values, which do not coincide with $u(\mathbf{x}_i)$ [1], leading to many techniques developed in order to fulfill Dirichlet BC (see e.g. [17, 31]). Moreover, if one needs to approximate $u(\mathbf{x}_i)$, a "reconstruction" step must be
performed [17].

Neumann BC are imposed by approximating the normal derivative operator $\eta_l(u) = \partial u(\mathbf{x}_l)/\partial \mathbf{n}$ and setting the approximate relations $\widehat{\eta_l(u)} = \bar{q}(\mathbf{x}_l)$ for each $\mathbf{x}_l \in \Gamma_q$.

We emphasize that the MLPG methods described above require computing for each discretization node the MLS shape functions and their partial derivatives on its associated numerical quadrature points. In place of MLS shape functions, DMLPG involves polynomials. This is a key difference: MLS shape functions are computationally more demanding. Moreover, DMLPG requires evaluating once for each node a local contribution matrix, while each evaluation on a node of a MLS shape function in MLPG needs evaluating a local contribution matrix. Such differences are the main responsible of the higher CPU time one well guesses it is spent by MLPG over DMLPG, for a given approximation tolerance.

3 Finite dimensional spaces

In order to identify a trial space, both for MLPG and DMLPG we exploit a set of suitable *weights*, hence obtaining the so called *shape* functions [2, 4, 11, 17].

In order to fully specify our meshless procedures, we need to identify suitable *test* functions. On the ground of our previous results on 2D problems [18], as the test function space generators, we exploited TPF generated by suitable 1D functions (see the sequel).

3.1 Generating functions

Assume $w(t)$ is a given, differentiable, compact supported, generator function; when $t > 1$, $w(t) = 0$ holds.

A RBF associated to node $\mathbf{x}_i$ is identified by suitably setting a support radius r_i and considering

$$w\left(\frac{\|\mathbf{x}-\mathbf{x}_i\|_2}{r_i}\right).$$

Our weights and test spaces were obtained by using the Gaussian generator after [15]

$$w(t) = \begin{cases} \dfrac{\exp(-(\sigma t)^2) - \exp(-\sigma^2)}{1-\exp(-\sigma^2)} & 0 \le t \le 1 \\ 0 & t \ge 1, \end{cases} \tag{6}$$

where σ is a parameter controlling the function shape.

Concerning TPF, to each node, $\mathbf{x}_i$, we associate the TPF

$$\tau_i(x,y,z) = f(|x-x_i|/\eta_i^{(x)}) \cdot f(|y-y_i|/\eta_i^{(y)}) \cdot f(|z-z_i|/\eta_i^{(z)}), \tag{7}$$

by suitably choosing the $\eta_i^{(*)}$ factors. For simplicity, we assume $\eta_i^{(x)} = \eta_i^{(y)} = \eta_i^{(z)} = \bar{\eta}_i$, hence the support of $\tau_i(\mathbf{x})$ is a cube centered at $\mathbf{x}_i$. We call $\bar{\eta}_i$ the "radius" of the cube. We exploited polynomial generators after [14]

$$f(t) = \begin{cases} 1 - t^2, & 0 \leq t \leq 1, \\ 0, & t \geq 1. \end{cases} \tag{8}$$

4 Implementation issues

We extended DMLPG technique, proposed in [19], to 3D problems. This is a non trivial task, which needs identifying suitable data structures in order to deal with large sets of nodes. Moreover, solving the ensuing large linear systems demands for efficient iterative solvers (see the sequel).

Now we aim to compare DMLPG with MLPG techniques. Either quadratic or cubic polynomial bases are enrolled for MLS and GMLS. On the ground of our previous results [17, 18], we consider two effective MLPG techniques. One technique, labelled "RR" in the sequel, uses Gaussian RBF weights to provide MLS shape functions as the trial functions. The test functions are again Gaussian RBF weights, hence local integration domains are spheres. Our alternative MLPG technique exploits the same MLS shape functions as the trial functions, but the TPF arising from generators of type (8) as the test functions. This latter technique will be called "RT", a shorthand for RBF-TPF, as it uses trial RBF, and test TPF.

We numerically compare RR and RT when solving 3D problems, with our DMLPG technique, for shortness also labelled "D" in the sequel. Gaussian generators are exploited in order to provide GMLS weights. The test functions are the same TPF as in RT. Cuboidal local integration domains are thus considered.

Numerical cubatures on spheres were performed using Stroud's 512 node, degree 15 cubature rule (subroutine `SPH15`, after [27]). In our previous works, we found that such rule provide accurate integral values at affordable computational costs [17]. Numerical integrations on cuboids were accomplished using Gauss-Legendre product formula with 5^3 nodes.

4.1 Domain discretization

In order to analyze the main features of our meshless procedures, we focus on *uniform* grids, U_j, on $[0,1]^3$, i.e. sets of nodes which are corners of uniform grids. Their edge lengths are $h_j = 1/2^{j+1}$, $j = 1,2,3,4$. The number of discretization nodes is 125, 729, 4913, 35937.

Many parameters must be tuned in order to identify effective DMLPG methods. Recall that, despite DMLPG was introduced in order to deal with irregular "clouds" of points, tuning is not like to be safely performed on completely random sets of discretization points.

Assume $\tilde{u}_i$ is our approximate solution on $\mathbf{x}_i$, while $u_i = u(\mathbf{x}_i)$ is the exact solution.

Identifying effective meshless methods, requires tuning the trial support radius, r_i, on each node $\mathbf{x}_i$, and the test support radius, ρ_i. In order to fix ideas, for computational convenience, and according to our numerical experience, see [17, 18], we assume that $r_i > \rho_i$, $i = 1, \ldots, N$, hold true.

When the uniform discretization U_j with edge–length h_j is exploited, we set

$$\rho_i = \rho = \alpha\, h_j, \quad r_i = r = \beta\, h_j, \quad i = 1, \ldots, N.$$

the α and β parameters must be tuned.

Assume $\mathbf{x}_i$ is a node internal to $\Omega = [0,1]^2$, its distance from the boundary of Ω being b_i. In order to avoid that the integration domain in (4) go outside Ω, we set $\rho_i := b_i$, when $b_i < \rho_i$.

4.2 Solving linear systems

When considering 2D problems, MLPG linear systems are usually solved via direct methods, like [13, 21, 23]. These methods are efficient and accurate for 2D problems, but too storage demanding when 3D problems are attacked by using fine discretizations. Preconditioned iterative methods are the best choice for large 3D problems. We efficiently solved our non symmetric, sparse, 3D linear systems via preconditioned Bi-CGSTAB [29]. Preconditioning was performed via incomplete Crout factorization [10, 24]. Parallel implementations of this solver for non symmetric systems were exploited inside meshless techniques [5, 8]. The iterations were stopped when the i-th relative residual is smaller than 10^{-15}. Such tolerance ensures achieved double precision (64-bit) numerical accuracy attained. Each linear system was solved to maximum

accuracy, in order to avoid extra numerical errors in the solution procedure.

4.3 Hardware and software

We implemented our algorithms into FORTRAN 77 codes, compiled via XLF v9.1. They were run on an IBM Power 5, 2 dual–core 1.9 GHz CPUs. The machine has a 16 GB RAM, two–level cache, 64 KB first level, 2 MB second level. Its operating system is AIX version 5.3.

More details on hardware/software configuration are not useful, since in this paper we compare codes written in the same programming language, running on the given machine described above.

5 Numerical experiments

5.1 Test solutions

In the sequel we report our numerical results concerning the solution of problem (1) on the $[0,1]^3$ domain. Both the forcing function, f, and the BC, were computed after each solution in a given set of test ones. The latter solutions are drawn after our previous work on 3D potential problems [17]. One polynomial sample was added, in order to enlarge our set.

After [32], we consider the harmonic polynomial

$$u(x,y,z) = x^3 + y^3 + z^3 - 3yx^2 - 3xz^2 - 3zy^2. \tag{9}$$

In the sequel, this test solution is labeled "P3".

Moreover, we added the slightly different, incomplete polynomial (labeled "P3i")

$$u(x,y,z) = x^3 + y^3 + z^3 + x\,y\,z. \tag{10}$$

After [30], two trigonometric functions are considered.

- The function "T1" which evenly changes along the three coordinate directions

$$u(x,y,z) = \sin x + \sin y + \sin z + \sin(3x) + \sin(3y) + \sin(3z). \tag{11}$$

- The function "T2" which changes more rapidly in the z- and y-directions.

$$u(x,y,z) = \sin x + \sin y + \sin z + \sin(5y) + \sin(10z). \tag{12}$$

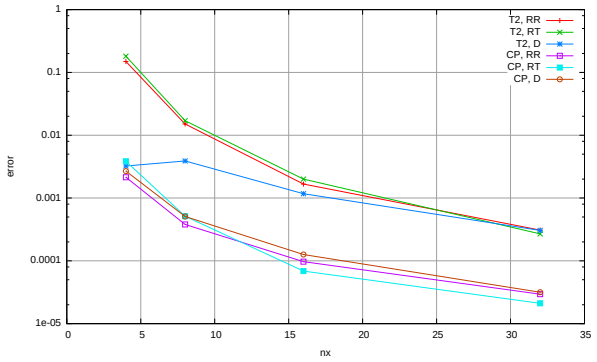

Figure 1: RR, RT, and D, error behaviors when solving problems T2 and CP.

Moreover, we consider the *composed cosine–polynomial* solution "CP" after [16]

$$u = \cos\left(3\pi\left(\frac{x^3+y^3+z^3}{3} - \frac{x^2+y^2+z^2}{2}\right)\right). \tag{13}$$

5.2 Results

In the sequel, in order to focus on local errors on an N–node discretization, we exploit the maximum norm error

$$e^{(max)} = \max_{i=1}^{N} |u_i - \tilde{u}_i|. \tag{14}$$

Table 1 reports errors and convergence rates for RR, RT, and D methods, when pure Dirichlet BC are set. Values $\beta = 4$, $\sigma = 4$ were set for RR and RT, while $\beta = 1.9$, $\sigma = 3$ for D technique. The second column reports the number of nodes on the x-axis, which equals those on the y- and z- ones. Note that when approximating cubic polynomial solutions (i.e. P3, P3i) DMLPG displays much smaller errors (in the range 10^{-14} to 10^{-10}) than the other methods, errors which slightly increase when the discretization is refined. This counterintuitive behavior is to be ascribed to high precision in the approximate solution, which slightly decreases when a larger number of nodes, and hence of floating point operations, is introduced. This behavior produces meaningless convergence ratios, due to floating point errors. Note that a quadratic polynomial basis is exploited, hence DMLPG is not guaranteed to exactly recover third order polynomials. By inspecting Table 1, one can see that the errors raised by DMLPG when approximating our non–polynomial solutions are well comparable with those recorded by RR and RT techniques.

		$e^{(max)}$			convergence rates		
Test	nx	RR	RT	D	RR	RT	D
	4	7.37E-04	1.13E-03	1.21E-13	-	-	-
P3	8	1.04E-04	1.78E-04	2.25E-12	2.8	2.7	-
	16	1.39E-05	2.26E-05	1.67E-11	2.9	3.0	-
	32	2.26E-06	3.78E-06	5.29E-10	2.6	2.6	-
	4	8.73E-04	1.05E-03	9.30E-14	-	-	-
P3i	8	1.18E-04	1.63E-04	1.69E-12	2.9	2.7	-
	16	1.50E-05	2.15E-05	2.01E-11	3.0	2.9	-
	32	2.10E-06	3.40E-06	6.24E-10	2.8	2.7	-
	4	3.98E-03	5.07E-03	1.58E-03	-	-	-
T1	8	5.36E-04	7.34E-04	4.18E-04	2.9	2.8	1.9
	16	6.87E-05	3.50E-05	1.06E-04	3.0	4.4	2.0
	32	1.69E-05	1.20E-05	2.66E-05	2.0	1.5	2.0
	4	1.49E-01	1.81E-01	3.24E-03	-	-	-
T2	8	1.51E-02	1.70E-02	3.89E-03	3.3	3.4	-
	16	1.67E-03	2.00E-03	1.17E-03	3.2	3.1	1.7
	32	3.09E-04	2.69E-04	3.05E-04	2.4	2.9	1.9
	4	2.14E-03	3.85E-03	2.68E-03	-	-	-
CP	8	3.80E-04	5.14E-04	5.08E-04	2.5	2.9	2.4
	16	9.70E-05	6.88E-05	1.26E-04	2.0	2.9	2.0
	32	2.95E-05	2.11E-05	3.16E-05	1.7	1.7	2.0

Table 1: Dirichlet BC. Errors and convergence rates recorded when approximating the proposed test solutions. Dashes identify non computable and meaningless (negative) rates.

		CPU seconds			CPU ratios		Iterations		
Test	nx	RR	RT	D	RR/D	RT/D	RR	RT	D
P3	4	9.9	2.8	3.4	2.9	0.8	3	3	4
	8	129.3	33.5	3.6	36.2	9.4	5	5	6
	16	1291.2	329.6	11.0	118.1	30.2	8	7	10
	32	11583.5	2881.4	461.7	25.1	6.2	13	12	18
P3i	4	9.7	2.9	3.2	3.0	0.9	3	3	2
	8	134.6	33.6	3.6	37.7	9.4	5	5	5
	16	1310.7	323.6	11.1	118.4	29.2	8	7	9
	32	11657.4	2786.5	462.3	25.2	6.0	14	12	19
T1	4	9.8	2.9	2.3	4.3	1.3	3	3	2
	8	129.4	33.9	3.3	39.8	10.4	5	5	5
	16	1292.1	320.5	8.1	160.3	39.8	8	7	10
	32	11755.1	2859.2	469.6	25.0	6.1	14	12	19
T2	4	9.9	2.9	3.4	2.9	0.9	3	3	2
	8	129.1	34.5	3.6	36.4	9.7	5	5	6
	16	1289.4	323.2	10.9	118.7	29.8	8	7	10
	32	11333.5	2898.7	460.6	24.6	6.3	14	12	18
CP	4	9.8	2.9	2.5	3.9	1.2	3	3	2
	8	129.6	33.8	2.8	46.0	12.0	5	5	5
	16	1302.6	328.5	10.9	119.7	30.2	8	7	9
	32	11578.7	2890.7	412.0	28.1	7.0	14	12	16

Table 2: Dirichlet BC. CPU seconds, their ratios, and iterations spent for solving the final linear systems.

		$e^{(max)}$				convergence rates			
		$\delta = 2$			$\delta = 3$	$\delta = 2$			$\delta = 3$
Test	nx	RR	RT	D	D	RR	RT	D	D
	4	1.47E-01	2.17E-01	8.76E-01	9.13E-01	-	-	-	-
T2	8	1.81E-02	7.83E-02	4.69E-01	5.80E-01	3.0	1.5	0.9	0.7
	16	1.52E-02	1.42E-02	1.11E-01	4.33E-02	0.3	2.5	2.1	3.7
	32	1.37E-02	2.90E-03	2.94E-02	2.06E-03	0.1	2.3	1.9	4.4
	4	2.12E-03	7.23E-03	1.86E-01	9.30E-02	-	-	-	-
CP	8	3.08E-04	1.06E-03	2.40E-02	2.60E-01	2.8	2.8	3.0	-
	16	9.75E-05	2.14E-04	3.78E-03	1.36E-03	1.7	2.3	2.7	7.6
	32	2.95E-05	4.37E-05	6.80E-04	8.10E-05	1.7	2.3	2.5	4.1

Table 3: Mixed BC. Analogous to Table 1.

On the other hand, DMLPG convergence rates oscillates less respect to RR and RT, when the number of nodes changes. While RR and RT rates are far from constant, DMLPG ones quite well approach the theoretically optimal $\gamma = 2$ value. Figure 1 sketches the error behaviors when computing T2 and CP solutions. One can immediately grasp the similar RR, RT, and D, error behaviors, when solving either T2 or CP problem.

Table 2 reports CPU seconds spent for solving our test problems, together with the number of iterations spent to solve the final linear system. Note that inspecting column 6, which reports the ratio between CPU seconds spent by RR divided by D running seconds, DMLPG time can be up to approximately 160 times smaller. Column 7 reports RT/D CPU ratio, which is at most 39.8, since RT is faster than RR. On the other hand, let us see the number of iterations spent. Both RR, RT and D spent quite the same number of iterations in each test case.

Note that when the number of nodes on each axis is doubled, RR and RT CPU times are roughly multiplied by 9. On the other hand, DMLPG time increases slower, except an abrupt increase when going from 16 to 32 nodes per axis. Such increase is likely to be ascribed to higher DMLPG storage requirements, which triggers storage access problems when the number of discretization nodes increases much.

Concerning 3D problems with Mixed BC, assume we set Neumann boundary conditions on the portion $0.5 \leq x, y \leq 0.75$ of the $z = 0$ face on the $[0, 1]^3$

domain. Dirichlet conditions are set elsewhere. For each given test solution, we compute the corresponding BC for Poisson problem on the unit cube.

Table 3 shows errors and convergence ratios for tests T2 and CP. The value $\alpha = 1$, as above, was set in both tests. Parameter tuning for RR, RT, and D techniques resulted to be a bit harder for Mixed BC than for pure Dirichlet BC. For RR and RT, the scaling parameter in Gauss generator was $\sigma = 3$, while $\beta = 4$ was set.

When $\delta = 2$ CPU times are quite the same as in Table 1, hence they are not reported in the sequel. It is not worth running RR and RT with $\delta = 3$, since CPU times are too large. On the contrary, switching from $\delta = 2$ to $\delta = 3$, DMLPG CPU times do not increase appreciably. DMLPG results are reported for both quadratic ($\delta = 2$) and cubic ($\delta = 3$) bases. The value $\sigma = 5$ was set, while $\beta = 2.75$ was set when $\delta = 2$. Recall that a higher degree basis requires a bit larger β, i.e. larger supports for trial functions, in order to attain a non–singular Q matrix in GMLS. When $\delta = 3$ we set $\beta = 3.5$.

By inspecting Table 1 one can see that DMLPG with $\delta = 3$ is preferable over $\delta = 2$: good accuracy and higher convergence ratios, even better than the assumed optimal $\gamma = 3$ value, are attained. Recalling that DMLPG CPU times do not appreciably increase when $\delta = 3$, we infer that this technique is expected to be more effective than RR and RT.

6 Conclusions

The accuracy and efficiency of a DMLPG technique when solving 3D problems were numerically analyzed. Our DMLPG technique was compared with two MLPG procedures.

Our main points are the following.

- The CPU time spent by DMLPG can be up to one hundred time lower than RR running time, when achieving comparable or even higher DMLPG accuracy respect to RR and RT.

- Due to storage requirements, DMLPG running time can increase more rapidly when the number of discretization nodes becomes large.

- Respect to exploiting a quadratic basis, when considering a mixed problem, a cubic basis improves the accuracy and convergence rates of DMLPG, without appreciably increasing the CPU time spent. On the other hand

RR and RT with a cubic basis are too time consuming to be worth exploiting.

Future work is planned in order to solve unsteady diffusion problems.

Acknowledgments

The first Author received support from Università Ca' Foscari Venezia.

References

[1] S. N. Atluri. *The Meshless method (MLPG) for domain & BIE discretizations*. Tech Science, 2004, Forsyth GA, 2004.

[2] S. N. Atluri and T. Zhu. The meshless local Petrov-Galerkin (MLPG) method: A simple & less–costly alternative to the finite element methods. *Computer Modeling in Engineering and Sciences*, 3(1):11–51, 2002.

[3] I. Babuska, U. Banerjee, and J. E. Osborn. Survey of meshless and generalized finite element methods: A unified approach. Technical report, Office of Naval Research, One Liberty Center, 875 North Randolph Street Suite 1425, Arlington, VA, 22203-1995, 2005.

[4] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl. Meshless methods: an overview and recent developments. *Comp. Methods App. Mech. Eng.*, 139:3–47, 1996.

[5] L. Bergamaschi, A. Martinez, and G. Pini. An efficient parallel mlpg method for poroelastic models. *Computer Modeling in Engineering and Science*, 49(3):191–216, 2009.

[6] M. D. Buhmann. *Radial Basis Functions: Theory and Implementation*, volume 12 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2003.

[7] M. Duflot and H. Nguyen-Dang. A truly meshless method based on a moving least squares quadrature. *Commun. Numer. Methods Eng.*, 18:441–449, 2002.

[8] M. Ferronato, C. Janna, and G. Pini. Shifted FSAI preconditioner for the efficient parallel solution of non-linear groundwater flow models. *International Journal for Numerical Methods in Engineering*, 89:1707–1719, 2012.

[9] T.-P. Fries and H.-G. Matthies. Classification and overview of meshfree methods. Technical Report 2003-3, Technical University Braunschweig, Brunswick, Germany, 2004.

[10] D. S. Kershaw. The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations. *J. Comp. Phys.*, 26:43–65, 1978.

[11] P. Lancaster and K. Salkauskas. Surfaces generated by moving least squares methods. *Math. Comp.*, 37(155):141–158, 1981.

[12] D. Levin. The approximation power of moving least–squares. *Math. Comp.*, 67(234):1517–1531, 1998.

[13] S. Li, J. Demmel, J. Gilbert, and L. Grigori. SuperLU, 2012. Last accessed: September 7, 2012.

[14] G. R. Liu. *Meshfree Methods: Moving Beyond the Finite Element Method*. CRC Press, second edition, 2009.

[15] Y. Y. Lu, T. Belytschko, and L. Gu. A new implementation of the element free Galerkin method. *Comp. Methods App. Mech. Eng.*, 113:397–414, 1994.

[16] A. Mazzia, G. Pini, M. Putti, and F. Sartoretto. Comparison of 3D flow fields arising in mixed and standard unstructured finite elements. In P. S. et al., editor, *Computational Science – ICCS 2003*, volume 2657 of *Lecture Notes in Computer Sciences*, pages 560–567, Berlin, 2003. Springer-Verlag.

[17] A. Mazzia, G. Pini, and F. Sartoretto. Accurate MLPG solution for 3D potential problems. *Computer Modeling in Engineering & Sciences*, 36(1):43–63, 2008.

[18] A. Mazzia and F. Sartoretto. Meshless solution of potential problems by combining radial basis functions and tensor product ones. *Computer Modeling in Engineering & Sciences*, 68(1):95–112, 2010.

[19] D. Mirzaei and R. Schaback. Direct meshless local Petrov–Galerkin (DMLPG) method: A generalized MLS approximation. *Applied Numerical Mathematics*, 68(0):73 – 82, 2013.

[20] D. Mirzaei, R. Schaback, and M. Dehghan. On generalized moving least squares and diffuse derivatives. *IMA Journal of Numerical Analysis*, 32(3):983–1000, 2012.

[21] MUMPS: a MUltifrontal Massively Parallel sparse direct Solver, 2012. Last accessed: September 7, 2012.

[22] G. Ni, S. L. Ho, S. Yang, and P. Ni. Meshless local Petrov-Galerkin method and its application to electromagnetic field computations. *International Journal of Applied Electromagnetics and Mechanics*, 19(1):111–117, 2004.

[23] PARDISO solver project, 2012. Last accessed: September 7, 2012.

[24] G. Pini and G. Zilli. Preconditioned iterative algorithms for large sparse unsymmetric problems. *Numerical Methods for Partial Differential Equations*, 5:107–120, 1989.

[25] M. J. D. Powell. The theory of radial basis function approximation in 1990. In W. Light, editor, *Advances in Numerical Analysis. Wavelets, Subdivision Algorithms, and Radial Basis Functions*, pages 105–210. Clarendon Press, Oxford, 1992.

[26] M. Sterk and R. Trobec. Meshless solution of a diffusion equation with parameter optimization and error analysis. *Engineering Analysis with Boundary Elements*, 32:567–577, 2008.

[27] A. H. Stroud. *Approximate Calculation of Multiple Integrals*. Prentice-Hall, Englewood Cliffs, NJ, 1971.

[28] H. Sun, Y. Wang, and Y. Miao. Meshless numerical method based on tensor product. *Front. Archit. Civ. Eng. China*, 2(2):166–171, 2008.

[29] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13:631–644, 1992.

[30] J. Wang, W. Zhong, and J. Zhang. A general meshsize fourth-order compact difference discretization scheme for 3D Poisson equation. *Applied Mathematics and Computation*, 183(2):804–812, 2006.

[31] C.-K. C. Wu and M. E. Plesha. Essential boundary condition enforcement in meshless methods: boundary flux collocation method. *Int. J. Numer. Meth. Engng.*, 53:499–514, 2002.

[32] J. Zhang, M. Tanaka, and M. Endo. Meshless analysis of potential problems in three dimensions with the hybrid boundary node method. *International Journal for Numerical Methods in Engineering.*, 59:1147–1166, 2004.

Received: September 19, 2014; Published: November 24, 2014