Copyright © 2015 Tech Science Press CMES, vol.108, no.4, pp.239-262, 2015

A DMLPG Refinement Technique for 2D and 3D Potential Problems

Annamaria Mazzia[1], Giorgio Pini[1] and Flavio Sartoretto[2]

Abstract: Meshless Local Petrov Galerkin (MLPG) methods are pure meshless techniques for solving Partial Differential Equations (PDE). MLPG techniques are nowadays used for solving a huge number of complex, real–life problems. While MLPG aims to approximate the solution of a given differential problem, its "dual" Direct MLPG (DMLPG) technique relies upon approximating linear functionals. Assume adaptive methods are to be implemented. When using a mesh–based method, inserting and/or deleting a node implies complex adjustment of connections. Meshless methods are more apt to implement adaptivity, since they does not require such adjustments. Nevertheless, ad–hoc insertion and/or deletion algorithms must be devised, in order to attain a good accuracy. In this paper we introduce a fresh refinement technique for DMLPG methods. Nodes are inserted in a discretization cloud where the local variation in the solution is supposed to be "large". The variation is estimated using the (local) Total Variation (TV). DMLPG allows to directly estimate the partial derivatives, in order to compute the TV. MLPG must rely upon approximating the derivatives of the shape functions, hence MLPG refinement results to be more involved than its DMLPG counterpart. We show that our DMLPG refinement procedure allows one to efficiently solve a given diffusion problem whose solution undergoes large variations on a small portion of the domain. The accuracy afforded by a fine uniform cloud can be attained by using far less non–uniformly arranged nodes.

Keywords: Meshless Methods; MLPG; Direct MLPG; Refinement; Diffusion.

1 Introduction

Meshless methods nowadays supply a good alternative or integrative technique to Finite Element (FE) methods. As an example, meshless methods apply as an eli-

[1] Dipartimento ICEA, Università di Padova, Via Trieste 63, 35121 Padova, Italy {annamaria.mazzia,giorgio.pini}@unipd.it

[2] DAIS, Università Ca' Foscari Venezia, Via Torino 155, 10173 Venezia, Italy flavio.sartoretto@unive.it

gible choice for fracture analysis, see Yang, Budarapu, Mahapatra, Bordas, Zi, and Rabczuk (2015).

Meshless methods are particularly attractive when meshing is a time–consuming and cumbersome task [Gerace, Erhart, Kassab, and Divo (2013)], as it happens in adaptive methods.

Kriging was proposed as a method to identify a suitable discretization cloud [Chen and Liew (2011)]. This is a quite involved procedure. A key element for adaptivity, is a refinement procedure, which allows one to refine a discretization only where the solution u of the given differential problem undergoes large variations. In this paper, we introduce a refinement procedure, based on approximating the variation in u by estimating the (local) Total Variation (TV) of the numerical solution $\tilde{u}$.

Meshless Petrov Galerkin (MLPG) methods [Atluri (2004)] are truly meshless, and widely used. Our previous paper [Mazzia, Pini, and Sartoretto (2014)] deals with devising a refinement procedure for MLPG, that is based on the Moving Least Squares (MLS) method proposed in Lancaster and Salkauskas (1981).

Direct MLPG (DMLPG) [Mirzaei and Schaback (2013)], exploits the Generalized Moving Least Squares method (GMLS) [Mirzaei, Schaback, and Dehghan (2012)]. While MLS aims at approximating a multivariate function, GMLS approximates linear functionals. The solution of a linear Partial Differential Equation (PDE), formulated in strong or weak form, can be computed by approximating via GMLS the associated functionals. DMLPG exploits GMLS to approximate weak forms of PDE problems. DMLPG can be seen as a “dual” method to MLPG.

To identify the domain regions where any refinement is required, we adopted the (local) Total Variation (TV) as an indicator of the local variation in u. Estimating the TV entails computing the partial derivatives. DMLPG enables us to approximate the partial derivatives directly, while other meshless methods, MLPG included, must exploit expensive approximations of the partial derivatives of u.

To analyze the numerical properties of a refinement method, one must consider sound test problems and ad–hoc, simple domains. In the sequel, we numerically analyze the accuracy and efficiency achieved with our DMLPG–based refinement procedure when solving Poisson problems on either $[0,1]^2$ or $[0,1]^3$.

This paper is organized as follows. Section 2 recalls basic facts about the approximating schemes exploited by MLPG and DMLPG. Section 3 sketches the fundamental concepts underlying MLPG and DMLPG techniques. Section 4 identifies the trial and test spaces, which are a key ingredient of our numerical methods. Section 5 details our refinement strategy. Section 6 describes our distinguished test problems. Section 7 describes and discusses our numerical results. Section 8 summarizes our conclusions.

2 Approximation schemes

The Moving Least Squares (MLS) method [Lancaster and Salkauskas (1981)] has been proposed for approximating a function, $u(\underline{x})$, inside a region $\Omega \subset \mathbb{R}^d$ given a number, N, of its values, $u(\underline{x}_i)$, $\underline{x}_i \in \Omega$. 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 grounds of a suitable "weight" function [Mazzia, Pini, and Sartoretto (2008); Mazzia and Sartoretto (2010)]. The ensuing trial functions are called the "shape" functions of MLS.

The Generalized Moving Least Squares (GMLS) method [Mirzaei, Schaback, and Dehghan (2012)], is a technique which generalizes the MLS formulation [Levin (1998)], aiming to approximate continuous linear functionals in the dual of $\mathscr{C}^k(\Omega)$, for any $k \geq 0$. GMLS aims to approximate a linear functional, $\lambda(u)$, based on a given set of N linear functionals $\lambda_i(u)$. One obtains an approximation

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

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 $\lambda_i(u) = u(\underline{x}_i)$ can be a given set of u values on given nodes $\underline{x}_i \in \Omega$; $\widehat{\lambda(u)}$ in this case can be interpreted as a generalization of Finite Difference methods, based on all $u(\underline{x}_i)$ values, instead of a given stencil.

For a given continuous linear functional $\lambda(u)$, the polynomial–based GMLS computes an approximation $\widehat{\lambda(u)} = \lambda(p^*)$, where p^* minimizes a weighted least–squares error functional [Mazzia, Pini, and Sartoretto (2012)].

For more details on the algorithm, see e.g. Levin (1998); Mirzaei and Schaback (2013).

3 MLPG and DMLPG techniques

Let us consider the linear Poisson equation on the domain Ω

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

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

$$u = \bar{u} \quad \text{on } \Gamma_u, \quad \frac{\partial u}{\partial \underline{n}} \equiv q = \bar{q} \quad \text{on } \Gamma_q \tag{3}$$

where $\bar{u}$ and $\bar{q}$ are the prescribed potential and normal flux, respectively, on the Dirichlet boundary, Γ_u, and on the Neumann boundary, Γ_q, being $\partial\Omega = \Gamma = \Gamma_u \cup \Gamma_q$, $\Gamma_u \cap \Gamma_q = \emptyset$. The outward normal direction to Γ is denoted by $\underline{n}$.

In order to compute an approximation $\tilde{u} = \sum_i \tilde{u}_i \xi_i$ of the solution, a finite set of trial functions ξ_i is chosen.

Let us assume that the residual of eq. (2) is multiplied by a suitable test function τ. The divergence theorem is applied, thus obtaining the weak formulation for (2)

$$\int_\Omega \nabla u \cdot \nabla \tau d\Omega - \int_\Gamma (\nabla u \cdot \underline{n})\, \tau d\Gamma = \int_\Omega f\, \tau\, d\Omega, \quad \forall \tau \in \mathscr{S}, \tag{4}$$

for any τ in a suitable functional space $\mathscr{S}$. Many MLPG methods have been proposed [Atluri (2004); Fries and Matthies (2004)], each of which 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.

A set of trial functions, ξ_i, is usually adopted, each of which is "centered" on node $\underline{x}_i$. The support of ξ_i, S_{ξ_i} is a ball centered at $\underline{x}_i$, whose radius is r_i. Actually, for each i we set $\xi_i = 0$ outside Ω, which is equivalent to considering $S_{\xi_i} \cap \Omega$, instead of S_{ξ_i}. The support of each test function τ_i, $S_{\tau_i} = \Omega_i$, is a ball (or a cuboid) centered at $\underline{x}_i$, whose radius (or half side length) is ρ_i. We assume $\tau_i = 0$ outside Ω, thus considering $S_{\tau_i} \cap \Omega$. We assume that $\tau_i = 0$ is on $\partial\Omega_i$, a typical setting in many MLPG schemes [Mirzaei and Schaback (2013); Mazzia and Sartoretto (2010)]. In principle the test functions need not necessarily be centered on the $\underline{x}_i$. For the sake of simplicity, we use as support centers the $\underline{x}_i$ nodes, for both the trial and the test functions.

A set of Local Weak Forms (LWF) is obtained by writing eq. (4) for each test function

$$\int_{\Omega_i} \nabla u \cdot \nabla \tau_i d\Omega - \int_{\Gamma_i^{(u)}} (\nabla u \cdot \underline{n})\, \tau_i d\Gamma = \int_{\Omega_i} f\, \tau_i\, d\Omega + \int_{\Gamma_i^{(q)}} (\nabla u \cdot \underline{n})\, \tau_i d\Gamma, \tag{5}$$

where $\Gamma_i^{(u)} = \partial\Omega_i \cap \partial\Gamma_u$ is the intersection of our local integration domain boundary with the *Dirichlet* boundary. Similarly, $\Gamma_i^{(q)} = \partial\Omega_i \cap \partial\Gamma_q$ is the intersection of our local integration domain boundary with the *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, contribute nothing, since $\tau_i = 0$ is on $\partial\Omega_i$. The boundary conditions are managed using suitable techniques [Atluri (2004); Mazzia, Pini, and Sartoretto (2008)].

The DMLPG technique is implemented by applying GMLS to the weak problem (5). The solution of our Poisson problem is approximated by using a polynomial space.

All the details of our implementation of the DMLPG technique for diffusion problems are given in Mazzia, Pini, and Sartoretto (2012).

4 Finite dimensional spaces

On the grounds of our previous 2D and 3D results [Mazzia, Pini, and Sartoretto (2008); Mazzia and Sartoretto (2010); Mazzia, Pini, and Sartoretto (2014)], we exploit appropriate *weight* functions $w(t)$ [Atluri and Zhu (2002); Belytschko, Krongauz, Organ, Fleming, and Krysl (1996); Lancaster and Salkauskas (1981); Mazzia, Pini, and Sartoretto (2008)], in order to identify suitable trial and test spaces.

4.1 Generating functions

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

Our i-th trial function ξ_i is a Radial Based Function (RBF) associated with node $\underline{x}_i$. After identifying a support radius r_i (see the sequel), we set

$$\xi_i = w\left(\frac{\|\underline{x} - \underline{x}_i\|_2}{r_i}\right),$$

where $w(t)$ is a Gaussian generator according to Lu, Belytschko, and Gu (1994), i.e.

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

Note that σ is a parameter controlling the function shape. Throughout this paper we assume that $\sigma = 1$.

Our test functions τ_i are Tensor Product Functions (TPF). We give the definitions for 3D problems; 2D problems are treated by disregarding the z values. With each node, $\underline{x}_i$, we associate the TPF test function

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

by choosing the appropriate $\rho_i^{(*)}$ factors. For the sake of simplicity, we assume that $\rho_i^{(x)} = \rho_i^{(y)} = \rho_i^{(z)} = \rho_i$, so the support of $\tau_i(\underline{x})$ is a cube centered at $\underline{x}_i$. The value ρ is called the "radius" of the cube.

The function $f(t)$ is in turn a polynomial generator according to Liu (2009)

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

4.2 Identifying the radiuses

A crucial step in obtaining accurate (D)MLPG schemes involves identifying the support trial basis radiuses r_i, and the test basis ones ρ_i, $i = 1, \dots, N$ [Mazzia and Sartoretto (2010)].

For each node we arrange in ascending order its distances from the closest $n_c \ll N$ neighbors. The n_c integer value was identified by suitable numerical experiments (see below).

Let us assume a given irregular discretization I.

In the case of 2D domains, for each node $\underline{x}_i$ in I we compute the $n_c = 7$ nodes closest to $\underline{x}_i$. We arrange their distances in ascending order $d_i^{(1)} \leq d_i^{(2)} \leq \dots \leq d_i^{(n_c)}$.

(a) If $d_k = d$, $k = 1, \dots, 4$, we can assume that the node distribution around the i-th node is "practically uniform". We set $r_i = \beta d$, $\rho_i = \alpha d$, α, β are parameters to be tuned. See the sequel for details.

(b) If not, we set

$$r_i = \beta d_i^{(n_c)}, \quad \rho_i = \alpha d_i^{(n_c)}. \tag{9}$$

The boundary nodes are treated in the same way as the internal nodes.

The parameters α and β were identified by means of numerical experiments: They fall in not too large ranges [Mazzia and Sartoretto (2010)]. Typical intervals are

$$0.5 \leq \alpha \leq 1.5, \quad 1.0 \leq \beta \leq 5.0, \quad \alpha < \beta. \tag{10}$$

Throughout this paper we assume that $\alpha = 1$. This setting proved suitable for all the test problems shown in the sequel.

As for 3D problems, it should be noted that there are six faces of a cube centered on any given point $\underline{x}_i$. We perform step (a) on six points, i.e. "local uniformity" corresponds to $d_i^{(k)} = d$ for $k = 1, \dots, 6$.

Alternatively, we perform step (b) by assuming that $n_c = 9$.

As we saw before, the parameter setting $\alpha = 1$ was adopted.

5 Discretizations

5.1 Uniform grids

Let us assume for the sake of simplicity that our domain is either the $[0,1]^2$ square, for 2D problems, or the $[0,1]^3$ cube for 3D problems.

When dealing with 2D problems, let us start with the uniform grid on $[0,1]^2$ identified by intersecting the sides of the square and those three x–parallel and three y–parallel lines that divide the x- and y-side evenly into $n_x = n_y = 4$ parts. Let us call this uniform discretization U_1, the "level" $\ell = 1$ uniform discretization. The interval spacing is $h_1 = 1/4$, $N = 5 \times 5 = 25$ nodes are identified. Each finer uniform discretization level is obtained by halving each sub–interval. We thus obtain $N = 25, 81, 289, 1089, 4225$ nodes, corresponding to levels $\ell = 1, 2, 3, 4, 5$.

The 3D uniform discretization grids on $[0,1]^3$ are obtained in a similar manner, by setting $n_x = n_y = n_z = 4$. On the initial level $\ell = 1$, we thus have $h_1 = 1/4$, $N = 5 \times 5 \times 5 = 125$ nodes are enrolled. Each finer level is obtained by halving each sub–interval. We obtain $N = 125, 729, 4913, 35937, 274625$ nodes, corresponding to levels $N = 1, 2, 3, 4, 5$.

5.2 Refinement strategy

The efficacy for DMLPG computations of the irregular discretizations built using our refinement strategy has to be tested.

For any irregular discretization, there are two measures worth considering. They are the fill distance $h_{I,\Omega}$, and the separation distance, q_I, i.e. Fasshauer (2007)

$$h_{I,\Omega} = \sup_{\underline{x} \in \Omega} \min_{1 \le i \le N} \|\underline{x} - \underline{x}_i\|, \quad q_I = \frac{1}{2} \min_{k \neq i} \|\underline{x}_k - \underline{x}_i\|, \quad 1 \le k, i \le N. \tag{11}$$

5.2.1 2D problems

The first discretization level $\ell = 1$ is set to $I_1 = U_1$, the $N = 25$ node uniform grid.

To refine a given discretization I_k at level $\ell = k$, we consider on each node $\underline{x}_i$ the "local" Total Variation (TV) of the solution u, defined as

$$\|u\|_{TV,i} = \|\nabla u\|_{1,i} = \int\int_{\Omega_i} (|u_x| + |u_y|) \, d\Omega,$$

where u_x and u_y are the partial derivatives of the solution. The advantages of this "variation measure" are clearly explained in Strang, (2007), pag. 415: The variational methods which use the TV norm "allow for discontinuities but disfavor oscillations".

To approximate the TV on node $\underline{x}_i$ by taking our numerical solution $\tilde{u}$ into account, we set

$$\|\tilde{u}\|_{TV,i} \simeq (|\tilde{u}_x(\underline{x}_i)| + |\tilde{u}_y(\underline{x}_i)|) \, |\Omega_i|,$$

$|\Omega_i|$ being the area of the i-th DMLPG integration subdomain. One can prove that, due to our previous assumptions on the domains of the trial and test functions, Ω_i

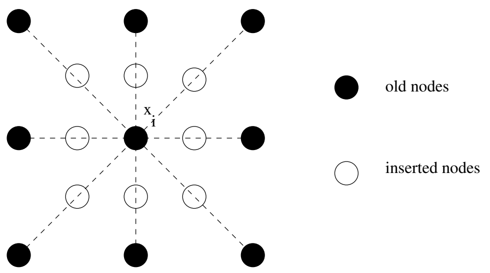


Figure 1: Node insertion 2D strategy, when a "locally–uniform" discretization is detected around node $\underline{x}_i$.

is the support of the i-th test function, i.e. the square centered at $\underline{x}_i$, the radius of which is ρ_i.

The partial derivatives $\tilde{u}_x(\underline{x}_i)$, $\tilde{u}_y(\underline{x}_i)$, are in turn approximated by exploiting the GMLS approach. Note that, using the MLPG approach, the partial derivatives are estimated by differentiating the MLS solution [Mazzia, Pini, and Sartoretto (2014)]. Now, using DMLPG enables the derivatives to be approximated directly. From this point of view, DMLPG is superior to MLPG.

Let

$$\mu = \max_{i=1,\ldots,N} \|\tilde{u}\|_{TV,i}.$$

If we assume a given threshold parameter γ, we refine our discretization around each given node $\underline{x}_i$ if and only if

$$\|\tilde{u}\|_{TV,i} > \gamma\mu.$$

Our refinement procedure around node $\underline{x}_i$ involves adding one node in the middle of each line joining $\underline{x}_i$ with the n_c' nodes closest to it. The value n_c' must be guessed on the grounds of geometrical considerations. We set $n_c' = 8$ for 2D problems, so that all "diagonal" nodes are added on an uniform grid (see Figure 1).

Note that when a node being inserted overlaps an old one, its insertion is skipped.

Once all the nodes in a given discretization I_k have been processed, we say that a new "discretization level" I_{k+1} is ready.

Our refinement procedure is repeated until a maximum number $N_{\max}$ of nodes has been reached, or a maximum number of "discretization levels" $\ell_{\max}$ has been computed. The maximum number of nodes depends on the storage space available.

Note that, under the previous assumptions, uniform refinements are obtained when $\gamma = 0$ and an initial uniform distribution has been adopted, On the other hand, uniform refinements can be easily computed by evenly dividing intervals, whereas performing our refinement procedure demands some more assessments. That is why, although the nodes are the same, the CPU time spent on computing an uniform refinement U_ℓ by setting $\gamma = 0$ is considerably longer than the time needed to compute the nodes directly in U_ℓ. This applies to the 3D domain, $[0,1]^3$ too.

5.2.2 3D problems

The first discretization level $\ell = 1$ is set to $I_1 = U_1$, the $N = 125$ node uniform grid on $[0,1]^3$.

When dealing with 3D problems, the straightforward generalization of our 2D "local" TV applies, where subdomain volumes are involved instead of areas.

Our refinement strategy is updated accordingly by setting $n'_c = 24$. When a (local) uniform discretization is detected, all the "diagonal" nodes in a cube. As in 2D domains, the setting $\gamma = 0$ produces uniform clouds.

5.3 Implementation issues

We implemented our algorithm in FORTRAN 77 codes, compiled via XLF v9.1. They were run on a machine operating under Ubuntu with an Intel Core i7-2600K, 3.40 GHz (quad core) processor, and 2x4GB, 1333 MHz, RAM.

No complicated data structures were needed to deal with our discretizations because they are mere clouds of points.

Our Fortran codes use a CSR sparse matrix storage representation to deal with the large number of nodes needed for volume discretizations in non–structural, (e.g. flow, heat) problems.

To arrange in ascending order the distances of each node from its neighbors, we used the Fortran subroutine SORT2 available in the NAPACK library.

6 Test problems

To check our adaptive strategy, we assign the forcing function f and compute the boundary conditions in eq. (2), so that its "test" solution is a function u undergoing large variations on a small portion of the domain.

6.1 2D problems

First, we consider the classical Gaussian function, centered on a given point $P_0 = (x_0, y_0)$, i.e.

$$u(x,y) = \exp(-c\,\left((x-x_0)^2 + (y-y_0)^2\right)). \tag{12}$$

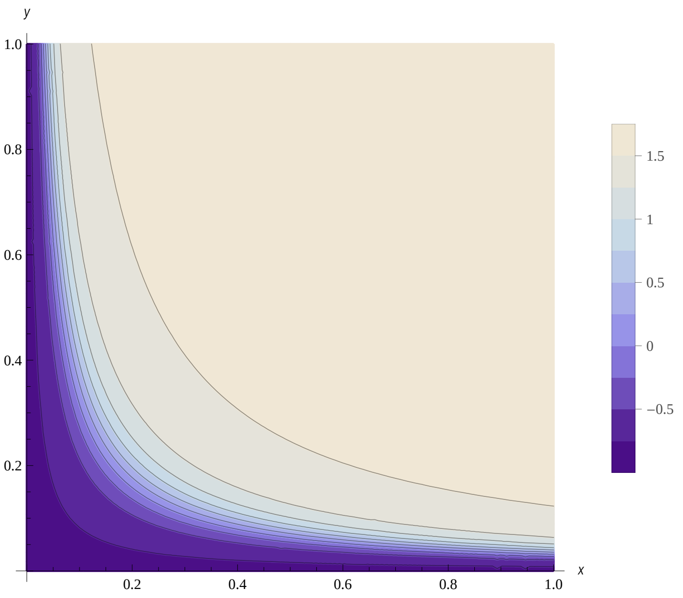


Figure 2: Contour regions for the solution of the test problem $P_2(T)$.

The parameter c is a large positive value that generates a high "hump" around P_0. In the sequel, we set $c = 200$ unless stated otherwise.

Let us assume that we numerically solve the Poisson problem (2) in $\Omega = [0,1]^2$, having set the Dirichlet boundary conditions such that its solution is the function (12). The setting $P_0 = (1/2, 1/2)$, the centroid of our domain, corresponds to the 2D problem called $P_2(GC)$ in the sequel, where "GC" stands for "Gaussian–centroid".

Any adaptive procedure is likely to be effective when finer discretizations adopt a large number of discretization nodes near the point P_0 where a large variation in u occurs. On the other hand, "far" away from P_0 the u values are small, and u does not display large variations, so the nodes can be distributed quite coarsely with no appreciable loss of accuracy.

As a further test problem we consider, as in Kee, Liu, Zhang, and Lu (2008)

$$u(x,y) = \tan^{-1}(1000x^2y^2 - 1). \tag{13}$$

This function displays a "hill" rising from the bottom left of $[0,1]^2$. Figure 2 shows the contour levels of the surface.

Let us assume that we numerically solve the Poisson problem (2) in $\Omega = [0,1]^2$, having set the Dirichlet boundary conditions such that its solution is the function (13).

The ensuing differential problem is labeled test problem $P_2(T)$, where "T" stands for the "arcTan–based" solution.

6.2 3D problems

Simply extending our 2D test problems, we consider Gaussian functions centered at a given point $P_0 = (x_0, y_0, z_0)$

$$u(x,y,z) = \exp(-c\left((x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2\right). \tag{14}$$

When dealing with 3D problems, we set $c = 200$ unless stated otherwise.

Let us assume that we aim to solve the Poisson problem on $[0,1]^3$ with Dirichlet boundary conditions such that the function (14) is its solution. We exploited many settings, corresponding to Gaussian "centers" on suitable domain points, either inside or on the boundary of our domain.

- The setting $P_0 = (1/2, 1/2, 1/2)$, is called test problem $P_3(GC)$. The label "GC" stands for "Gaussian–Centroid".
- When we set $P_0 = (1/2, 1/2, 0)$, we are speaking of test problem $P_3(GXY)$. The label "GXY" reminds us that the Gaussian is centered on $X = Y = 1/2$.
- The setting $P_0 = (1/2, 0, 0)$, is called test problem $P_3(GX)$. The label "GX" stands for "Gaussian centered on $X = 1/2$" (and $Y = Z = 0$).
- The setting $P_0 = (1/2, 1/2, 0)$, $c = 500$, is called test problem $P_3(GXY5)$. The "GXY5" is a mnemonic for the GXY case, where "$c = 500$ was set".

We also consider the function

$$u(x,y,z) = \tan^{-1}(1000x^2y^2z^2 - 1).$$

By applying the Dirichlet boundary conditions to the 3D Poisson equation, the ensuing test problem is called $P_3(T)$. The label "T" stands for "arcTan".

7 Numerical results

N being the number of discretization nodes, we define the numerical error

$$e = e_{u,N} = \frac{\max_{i=1}^{N} |u_i - \tilde{u}_i|}{\max_{i=1}^{N} |u_i|} \simeq \frac{||u - \tilde{u}||_\infty}{||u||_\infty},$$

Table 1: Number of nodes, fill distance $h_{U_\ell,[0,1]^2}$ and separation distance q_{U_ℓ} computed for uniform and refined 2D clouds, when solving problem $P_2(GC)$. The refined clouds were obtained by setting $\gamma = 10^{-3}$.

ℓ	Uniform			$\gamma = 10^{-3}$		
	N	$h_{U_\ell,[0,1]^2}$	q_{U_ℓ}	N	$h_{R_\ell,[0,1]^2}$	q_{R_ℓ}
1	25	1.77E-001	1.25E-001	25	1.77E-001	1.25E-001
2	81	8.84E-002	6.25E-002	81	8.84E-002	6.25E-002
3	289	4.42E-002	3.13E-002	289	4.42E-002	3.13E-002
4	1089	2.21E-002	1.56E-002	497	4.42E-002	1.56E-002
5	4225	1.10E-002	7.81E-003	1790	4.42E-002	7.81E-003

where u_i is the exact value on node $\underline{x}_i$, while $\tilde{u}_i$ is the corresponding approximated value.

When solving 2D problems, we exploited cubic polynomials in the GMLS approximation. When solving 3D problems, we exploited the quadratic polynomial approximation space.

We are mainly interested in comparing the accuracy achieved by means of our refinement procedure as opposed to uniform discretizations.

7.1 2D Numerical Results

We performed several 2D numerical experiments, but to summarize here our 2D results we only report here on two test problems. Our procedure actually proves most effective on 3D problems, which demand a large number of nodes in order to be solved accurately.

Let us assume that $\beta = 3.5$ for 2D problems.

Based on our previous experiments with MLPG refinements [Mazzia, Pini, and Sartoretto (2014)], and extensive numerical experiments, we set the refinement parameter γ in the range of $0 \leq \gamma \leq 10^{-2}$.

7.1.1 Discretizations

For reasons of storage and efficiency, our finest uniform cloud is the one with $N = 4,225$ nodes obtained at level $\ell = 5$. This is the finest uniform cloud that we used in all our 2D numerical experiments.

Table 1 shows the number of nodes, and the fill and separation distance for some discretization levels ℓ generated when solving the test problem $P_2(GC)$. We consider the uniform clouds U_ℓ and the corresponding refined ones R_ℓ obtained by setting $\gamma = 10^{-3}$. When using other γ values, and dealing with the other test prob-

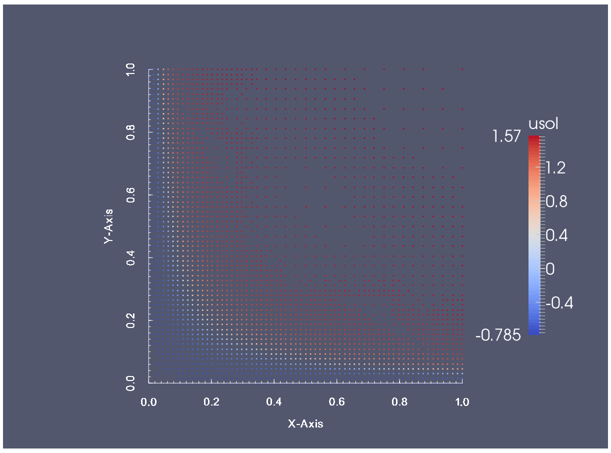
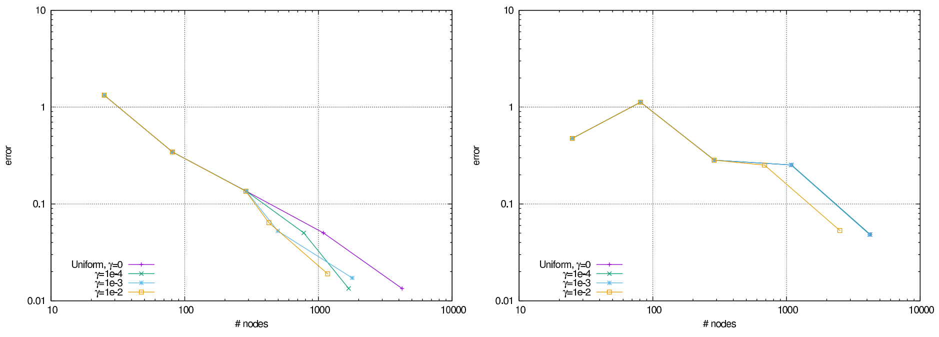


Figure 3: Cloud nodes adopted by setting $\gamma = 10^{-3}$ at the refinement level $\ell = 5$, when Problem $P_2(T)$ is solved.



(a) (b)

Figure 4: Errors recorded when our 2D test problems are solved. The errors obtained using uniform clouds coincide with the $\gamma = 0$ refinement levels. Frame (a) refers to Problem $P_2(GC)$. Frame (b) applies to Problem $P_2(T)$, where the curve for $\gamma = 0$ overlaps those corresponding to $\gamma =$ 1e-4, 1e-3.

lems, the distances recorded come between the "uniform" and the $\gamma = 10^{-3}$ types of behavior.

As we can see from the columns labeled "Uniform" in Table 1, at each new level the fill and separation distances are halved when an uniform refinement is performed, as expected. The refinement with $\gamma = 10^{-3}$ produces the same separation distances q_{R_ℓ}, while the fill distance $h_{R_\ell,[0,1]^2}$ remains unchanged at levels $\ell = 3,4,5$. Recalling the definitions (11) one can argue that our refinement procedure generates "fine sub–regions" the distances of which are larger than their internal separation, while

Table 2: The cloud level needed for attaining the HUA is given for some values of the refinement parameter γ. The corresponding number of nodes N, and the error e obtained are also shown.

Problem	γ	ℓ	N	e
	0	5	4225	1.34E-2
$P_2(GC)$	1.00E-4	5	1681	1.34E-2
	1.00E-3	5	1790	1.72E-2
	1.00E-2	4	1172	1.90E-2
	0	5	4225	4.83E-2
$P_2(T)$	1.00E-4	5	4225	4.83E-2
	1.00E-3	5	4172	4.83E-2
	1.00E-2	5	2504	5.33E-2

the uniform refinement does not.

We could show that the refined (non–uniform) clouds cluster around the Gaussian center P_0, but we prefer to show the more interesting behavior seen when dealing with Problem $P_2(T)$. Figure 3 shows the nodes identified by our refinement procedure. The refinement level $\ell = 5$ is considered. By comparing Figure 3 with Figure 2, one can see that the nodes coalesce well around the "hill" corresponding to the region of greatest variation in the solution.

Recalling our considerations on the cloud distance measures, we argue that our refinement technique effectively clusters the nodes around the regions of high variation in the solution.

7.1.2 Convergence history

Figure 4 plots the behavior of the errors vs the number of nodes in the discretizations. The refinement parameter $\gamma = 0$, 1e-4, 1e-3, 1e-2 values were set. Looking at Figure 4, one can see that the error is almost always lower when the number of refined nodes is larger. Frame (b) in Figure 4 shows that the error increases slightly when we go form U_1 to U_2, and this applies to non–uniform refinements too.

Figure 4 shows that, by setting $\gamma = 0$, we can replicate the uniform clouds. Larger and larger values γ values produce non–uniform clouds that enable the same accuracy to be achieved with a smaller number of nodes.

Table 2 reports the number of refined nodes needed to to obtain the same accuracy order as with our finest, uniform discretization level, which contains $N = 4225$ nodes. In the sequel this is called the "Highest Uniform Accuracy" (HUA).

It is easy to see by comparing Figure 4 with Table 2 that our refinement procedure enables the HUA to be reached using a considerably smaller number of nodes than

Table 3: Fill distances and separation distances computed for uniform and refined 3D clouds when solving problem $P_3(GX)$, $\gamma = 5 \times 10^{-4}$.

ℓ	Uniform			$\gamma = 10^{-3}$		
	N	$h_{U_\ell,[0,1]^3}$	q_{U_ℓ}	N	$h_{R_\ell,[0,1]^3}$	q_{R_ℓ}
1	125	2.17E-01	1.25E-01	125	2.17E-01	1.25E-01
2	729	1.08E-01	6.25E-02	611	1.29E-01	6.25E-02
3	4913	5.41E-02	3.13E-02	1131	1.32E-01	3.13E-02
4	35937	2.71E-02	1.56E-02	1938	1.33E-01	1.56E-02

in our finest uniform cloud.

As an example, row 4 in Table 2 reports that when dealing with problem $P_2(GC)$, by setting $\gamma = 1.0\text{e-}2$, only $N = 1172$ nodes in the refined cloud enable the HUA to be reached, which corresponds to $N = 4225$ nodes in the uniform discretization.

We note that, as one can infer from Table 2, when dealing with the non–Gaussian solution (Problem $P_2(T)$), more nodes are required to reach the HUA. This comes as no surprise: Gaussian solutions undergo large variations in the vicinity of the Gaussian center, and small variations elsewhere, while the "arcTan–based" test solution undergoes large variations over a much broader portion of the domain.

7.2 3D Numerical Results

By means of extensive numerical experiments we tuned $3.0 \leq \beta \leq 4.0$, using the value that afforded the best accuracy for each test problem.

Our analysis and numerical experiments showed that $0 \leq \gamma \leq 5 \times 10^{-3}$.

7.2.1 Discretizations

Table 3 shows the fill and separation distance for our 3D clouds generated when solving test problem $P_3(GX)$. The level $\ell = 1$ corresponds to an uniform $N = 125$ node cloud, as mentioned earlier. We consider the uniform clouds and our refined clouds obtained by setting $\gamma = 10^{-3}$. The finest uniform cloud contains $N = 35937$ nodes. Table 3 shows that the refined clouds have the same separation distances as the uniform grids. The fill distances in the uniform grids halve at each level, as one can guess. On the other hand, the fill distances in the non–uniform, refined clouds remain practically unchanged for levels $\ell = 2, 3, 4$. These results confirm that, as in 2D problems, our 3D refinement procedure generates fine sub–clouds containing more nodes than in other regions where a coarser discretization is produced.

Figure 5 sketches the cloud obtained with our refinement procedure at level $\ell = 5$, when Problem $P_3(GX)$ is dealt with. The refinement parameter value is $\gamma = 5 \times$

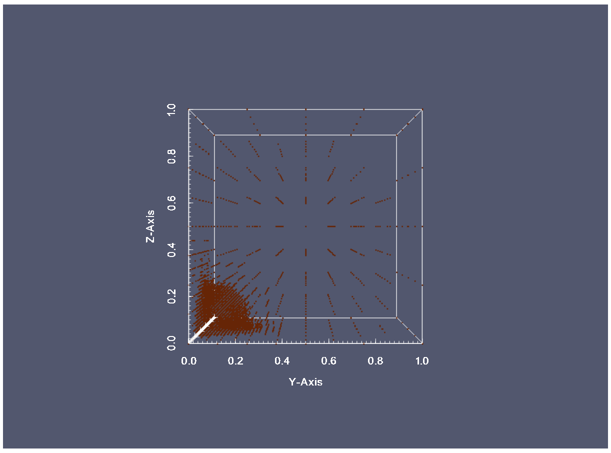
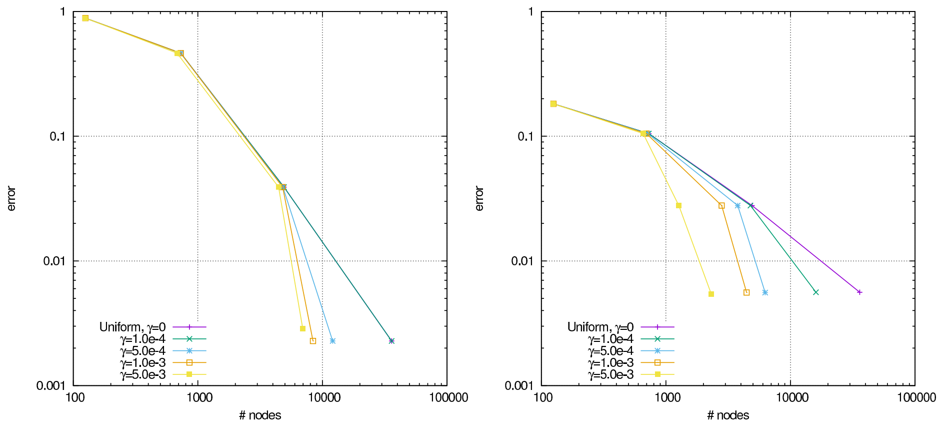


Figure 5: Non–uniformly refined clouds at levels $\ell = 5, 6$, when dealing with Problem $P_3(GX)$. The Gaussian center is $P_0 = (1/2,0,0)$ The refinement parameter value was set to $\gamma = 5.0 \times 10^{-4}$.



(a) (b)

Figure 6: Errors recorded when our 3D test problems are solved, using some γ values. We recall that, by setting $\gamma = 0$, one generates uniform clouds. Frame (a) refers to Problem $P_3(GC)$, and Frame (b) to Problem $P_3(GXY)$.

10^{-4}. As expected, many nodes are clustered around the Gaussian center $P_0 = (1/2,0,0)$.

Note that the cloud obtained at level $\ell = 6$ produces exactly the same Figure. The refinement procedure adds points close to the "blob" shown, so in Figure 5 they overlap with the nodes at level $\ell = 5$.

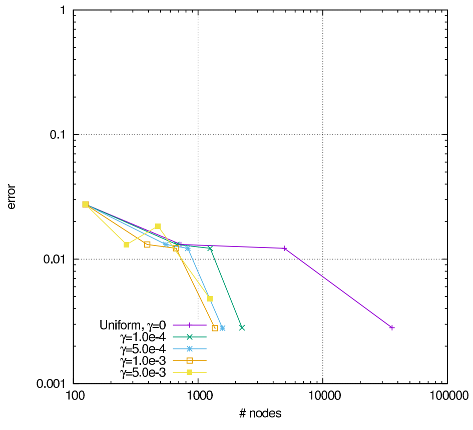
Table 4: For each problem and refinement parameter value, we report the level ℓ and the number of nodes enrolled for reaching the HUA (see the error values e). The CPU seconds T_γ spent on solving the Problem and the "speedup" with respect to the time spent when using the finest uniform discretization T_U are also shown.

Problem	γ	ℓ	N	e	T_γ	S_γ
	Uniform	4	35937	2.28E-3	471.32	1.00
	1.00E-4	4	12091	2.28E-3	95.28	4.95
$P_3(GC)$	5.00E-4	4	8399	2.28E-3	61.89	7.62
	1.00E-3	4	6957	2.86E-3	48.29	9.76
	5.00E-3	4	5193	6.66E-3	29.61	15.92
	Uniform	4	35937	1.56E-2	469.15	1.00
	1.00E-4	4	25605	1.56E-2	305.57	1.54
$P_3(GXY5)$	5.00E-4	4	11401	1.56E-2	82.45	5.69
	1.00E-3	4	7399	1.55E-2	45.62	10.28
	5.00E-3	4	3014	1.55E-2	14.02	33.46
	Uniform	4	35937	5.61E-3	470.66	1.00
	1.00E-4	4	20151	5.61E-3	202.67	2.32
$P_3(GXY)$	5.00E-4	4	8037	5.58E-3	51.74	9.10
	1.00E-3	4	5645	5.55E-3	22.10	21.30
	5.00E-3	4	2808	5.38E-3	8.18	57.54
	Uniform	4	35937	2.81E-3	470.00	1.00
	1.00E-4	4	3009	2.81E-3	11.31	41.56
$P_3(GX)$	5.00E-4	4	1938	2.80E-3	6.58	71.43
	1.00E-3	4	1646	2.81E-3	5.38	87.36
	5.00E-3	4	1315	2.91E-3	3.68	127.72
	Uniform	4	35937	4.35E-3	481.23	1.00
	1.00E-4	4	35866	4.35E-3	604.66	0.80
$P_3(T)$	5.00E-4	4	34699	4.44E-3	568.58	0.85
	1.00E-3	4	33587	4.54E-3	533.74	0.90
	5.00E-3	4	30165	4.92E-3	431.92	1.11

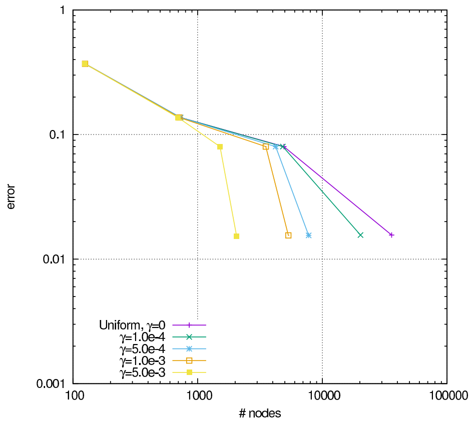
These considerations, supplemented with our previous analysis on Table 3, confirm that our 3D refinement procedure enables an effective clustering of the nodes around the regions where the greatest variations in the solution occur.

7.2.2 Convergence history

Figures 6, 7, and 8 report the errors raised when various refinement parameter values are set. Note that the error almost always decreases when the refining nodes



(c)



(d)

Figure 7: Same as in the previous Figure. Frame (c) refers to Problem $P_3(GX)$, and Frame (d) to Problem $P_3(GXY5)$.

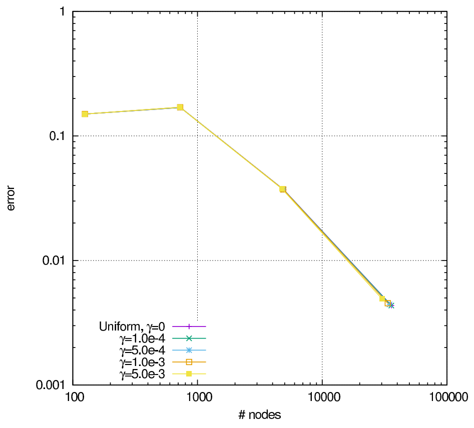


Figure 8: Errors recorded when Problem $P_3(T)$ is attacked. Note that the curves overlap almost completely.

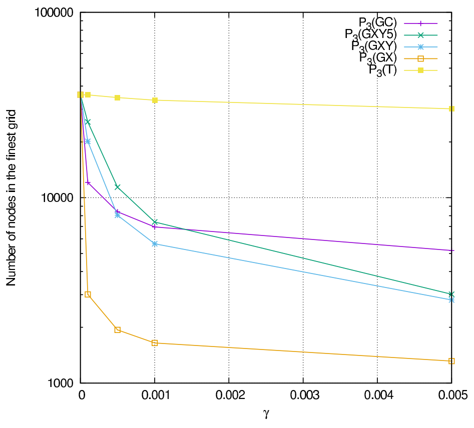


Figure 9: Number of nodes in the HUA 3D discretization level vs the refinement parameter value γ.

are added to the discretization cloud. By inspecting these Figures, one can confirm that, like for 2D problems, by setting $\gamma = 0$ one can replicate the uniform clouds, whereas setting $\gamma > 0$ produces non–uniform clouds.

We aim to attain the highest accuracy afforded by the finest uniform mesh (HUA), counting $N = 35,937$ nodes. The HUA can clearly be achieved with a far smaller number of nodes, however, by exploiting our refinement procedure.

Table 4 summarizes our results. As shown in the Table, the larger the γ, the smaller

Table 5: FEM errors when the level $\ell = 5$ uniform mesh of Type (a) or Type (b) is used. To reach the HUA, $\ell = 5$ levels were generated. The finest uniform meshes adopt the same $N = 274,625$ nodes (though their connections differ).

Problem	Type (a)	Type (b)
$P_3(GC)$	9.96E-003	1.01E-002
$P_3(GXY5)$	1.74E-002	2.04E-002
$P_3(GXY)$	7.00E-003	8.54E-003
$P_3(GX)$	4.50E-003	7.97E-003
$P_3(T)$	4.03E-003	3.97E-003

the number of nodes required to attain the HUA. By setting $\gamma =$ 5e-3 the HUA is attained on Gaussian–based problems $P_3(GC)$, $P_3(GXY5)$, $P_3(GXY)$, $P_3(GX)$, using less than 5200 nodes. The best case concerns Problem $P_3(GX)$, where $N = 1315 \ll 35937$ nodes in the refined non–uniform cloud suffice to arrive at the HUA.

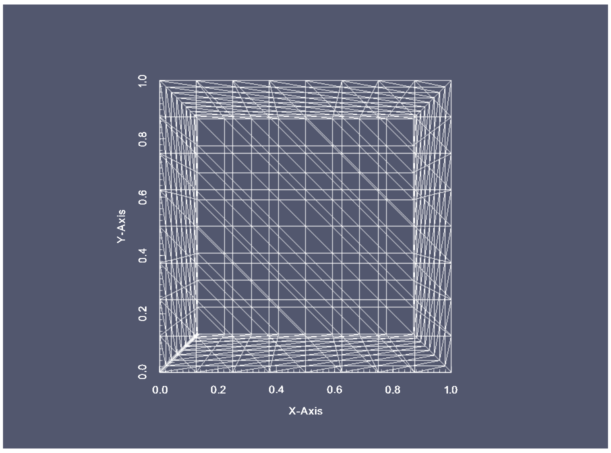
Now let us consider the curve pertaining to Problem $P_3(T)$, whose solution is "arcTan–based". When $\gamma =$ 5e-3, Table 4 shows that $N = 30,165$ nodes are required to reach the HUA. Such a large number is due to the large region where the solution of problem $P_3(T)$ undergoes high variations.

Figure 9 plots the number of discretization nodes needed to attain the HUA in relation to the refinement parameter γ. We recall that $\gamma = 0$ corresponds to uniform discretizations. By inspecting this Figure, it easy to see that our refinement is very effective for Gaussian solutions, but less for the arcTan–based.

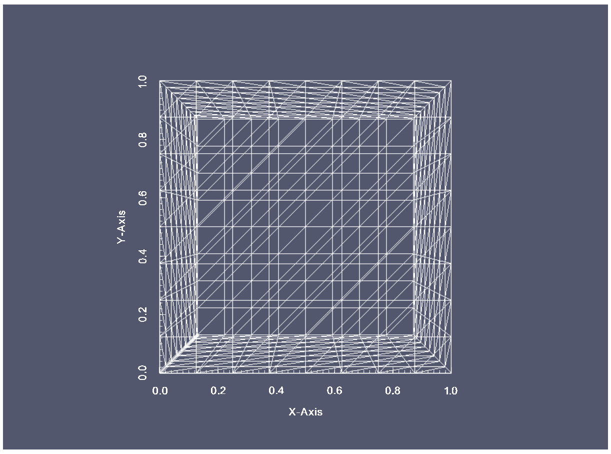
Let us compare our results with those obtained using a standard numerical method. Figure 11 shows the errors raised using linear tetrahedral Finite Elements vs the number of nodes. The plotted values were obtained using two types of 3D uniform mesh, called "Type (a)" and "Type (b)" in the sequel. Level $\ell = 2$ of our Type (a) and Type (b) uniform meshes are plotted in Figure 10. Our two meshes differ in the orientation of some sides of the triangles forming the base of some of the tetrahedral elements discretizing the inside of the domain cube. Note the different behavior of the errors in Frames (a) and (b) of Figure 11. One can see that using the same uniform distributed nodes, but choosing the elements differently, gives rise to a very different error behavior. This is a known FEM effect. The difference is exacerbated when the Gaussian centroid is set on a boundary side of the domain cube (see the curve for Problem $P_3(GX)$, $X = 1/2, Y = Z = 0$ in Figure 11).

These changes are not found when different clouds obtained by our refinement procedure are used. This suggests that our meshless procedure is less prone to node distribution in the discretization clouds than FEM is to mesh connections.

Table 5 shows the FEM errors. To reach the HUA by FEM, the level $\ell = 5$ mesh,



(a)



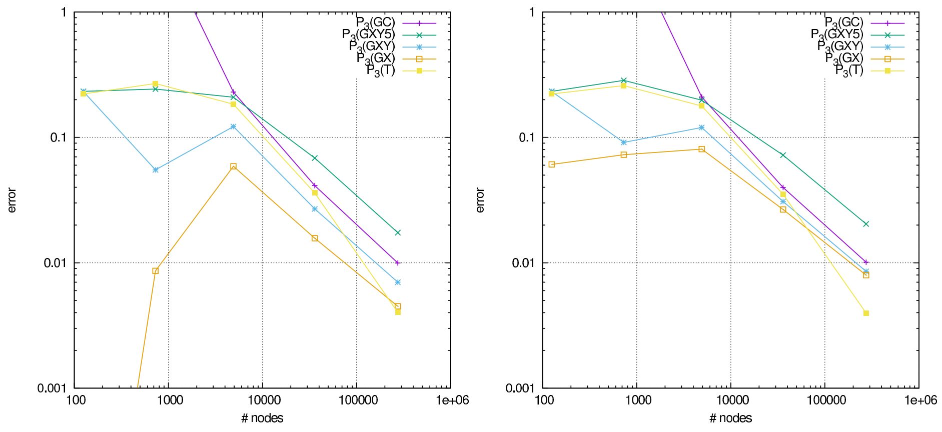
(b)

Figure 10: Sketch of the boundary triangles which form the bases of the tetrahedral elements inside one of our 3D FEM meshes. Frame (a) shows a mesh of Type (a), Frame (b) refers to Type (b).

containing $N = 274,625$ nodes, must be exploited. This number of mesh nodes is far larger than in our refined clouds; we needed to use less than $N = 35,937$ nodes, which is the number of nodes in the $\ell = 4$ uniform grid (see Table 4).

7.2.3 *Efficiency*

Let T_U be the CPU seconds spent on computing the solution using our finest uniform cloud. We recall that performing our refinement procedure with the setting



(a) (b)

Figure 11: Errors obtained when our 3D test problems are solved via linear FEM using uniform, tetrahedral meshes. Frame (a) shows the results obtained by Type (a) meshes. Frame (b) refers to Type (b) meshes.

$\gamma = 0$ produces the same nodes as in the uniform clouds. But the CPU time taken up by the former procedure is longer than when using "pre–computed" uniform clouds. When reporting CPU times, we must therefore distinguish between situations when "pre–computed" uniform discretizations are used as opposed to computing refinements with $\gamma = 0$. In order to perform a fair analysis, the CPU times labeled "Uniform" in Table 4 refer to when the less time–consuming, "pre–computed" uniform clouds were used.

We recall that HUA stands for the highest uniform accuracy (achieved with our finest uniform cloud). Let T_γ be the number of seconds spent on attaining the HUA by means of our refinement procedure with a given refinement parameter value $\gamma > 0$.

The last column in Table 4 shows the speed–up, $S_\gamma = T_U/T_\gamma$. Note that $S_U = T_U/T_U = 1$. When $S_\gamma > 1$, our refined cloud at level ℓ enables the overall computational cost to be reduced with respect to the uniform discretization U_ℓ.

By inspecting the S_γ column in Table 4, one can see that high speedups are recorded for Gaussian–based problems $P_3(GC)$, $P_3(GXY5)$, $P_3(GXY)$, $P_3(GX)$.

The highest speedups are obtained when dealing with the $P_3(GX)$ problem. When $\gamma = 10^{-3}$, an extremely effective speedup value $S_\gamma > 127.72$ is attained.

By inspecting Table 4 one can see that the worst speedups are obtained with the arcTan–based Problem $P_3(T)$. Only the row corresponding to $\gamma = 5 \times 10^{-3}$ reports a speedup larger than one.

To explain this not exciting result, we recall that large variations in the solution of problem $P_3(T)$ occur over a far broader region than in the other test problems. We recall that in the Gaussian 2D test problems, high variations occur inside a small circle around the Gaussian center, whereas the solution of the "arcTan" 2D test problem undergoes high variations on a wider "front" (see Figure 3). Correspondingly, the solutions of the Gaussian 3D test problems undergo non-negligible variations inside a small sphere around the Gaussian center, whereas the solution of problem $P_3(T)$ undergoes large variations over a large volume inside $[0,1]^3$, hence many more nodes are needed in order to obtain accurate approximations.

8 Conclusions

An effective, fresh refinement procedure for solving 2D and 3D diffusion problems using DMLPG was introduced and numerically analyzed.

The following points are worth considering.

- Concerning 2D problems:
  - our refinement procedure efficiently coalesces nodes around the regions where the solution undergoes large variations. Elsewhere, the discretization is left unchanged.
  - A moderate to large improvement in efficiency over uniform clouds is apparent. A far smaller number of nodes may be required to reach a given accuracy by comparison with uniform discretizations.
- As for 3D problems:
  - our refinement procedure clusters nodes around the regions of large variation in the solution, as in 2D problems.
  - By comparison with uniform discretizations, our refinement procedure allows to significantly reduce the number of nodes, with a consequent saving in CPU running time.
  - Let us deal with a test problem in which the Gaussian 3D solution is non–negligible only in the vicinity of the Gaussian "center". On the other hand, a very high speedup can be achieved by using our refinement procedure instead of uniform clouds.
  - Let us assume that the solution of a (test) problem undergoes a large variation on a large portion of the domain. As one can guess, in order to attain a given accuracy, a larger number of nodes is required than in the Gaussian centered case. A smaller gain in efficiency is recorded.

– Linear, tetrahedral FEM requires far more mesh nodes in order to achieve the same accuracy as our DMLPG procedure. As FEM practitioners know, for a given set of discretization nodes, the accuracy of (linear, tetrahedral) FEM strongly depends on the connections in the discretizing mesh. Conversely, the accuracy of our DMLPG–based refinement procedure does not rely on node connections, and it is more robust in response to changes in the discretizations.

Acknowledgement: This work was partially funded by INDAM–GNCS. The third Author received partial support from ADIR funding of Universitá Ca' Foscari Venezia. Some computations were performed on SCSCF, a multiprocessor cluster system owned by Universitá Ca' Foscari Venezia.

References

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

Atluri, S. N.; Zhu, T. (2002): The meshless local Petrov-Galerkin (MLPG) method: A simple & less–costly alternative to the finite element methods. *Computer Modeling in Engineering and Sciences*, vol. 3, no. 1, pp. 11–51.

Belytschko, T.; Krongauz, Y.; Organ, D.; Fleming, M.; Krysl, P. (1996): Meshless methods: an overview and recent developments. *Comp. Methods App. Mech. Eng.*, vol. 139, pp. 3–47.

Chen, L.; Liew, K. (2011): A local Petrov–Galerkin approach with moving Kriging interpolation for solving transient heat conduction problems. *Computational Mechanics*, vol. 47, no. 4, pp. 455–467.

Fasshauer, G. E. (2007): *Meshfree approximation methods with MATLAB*, volume 6 of *Interdisciplinary Mathematical Sciences*. World Scientific Publishing Co., Singapore. With 1 CD-ROM (Windows, Macintosh and UNIX).

Fries, T.-P.; Matthies, H.-G. (2004): Classification and overview of meshfree methods. Technical Report 2003-3, Technical University Braunschweig, Brunswick, Germany, 2004.

Gerace, S.; Erhart, K.; Kassab, A.; Divo, E. (2013): A model-integrated localized collocation meshless method (MIMS). *Computer Assisted Methods in Engineering and Science*, vol. 20, pp. 207–225.

Kee, B. B. T.; Liu, G. R.; Zhang, G. Y.; Lu, C. (2008): A residual based error estimator using radial basis functions. *Finite Elem. Anal. Des.*, vol. 44, no. 9-10, pp. 631–645.

Lancaster, P.; Salkauskas, K. (1981): Surfaces generated by moving least squares methods. *Math. Comp.*, vol. 37, no. 155, pp. 141–158.

Levin, D. (1998): The approximation power of moving least–squares. *Math. Comp.*, vol. 67, no. 234, pp. 1517–1531.

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

Lu, Y. Y.; Belytschko, T.; Gu, L. (1994): A new implementation of the element free Galerkin method. *Comp. Methods App. Mech. Eng.*, vol. 113, pp. 397–414.

Mazzia, A.; Pini, G.; Sartoretto, F. (2008): Accurate MLPG solution for 3D potential problems. *Computer Modeling in Engineering & Sciences*, vol. 36, no. 1, pp. 43–63.

Mazzia, A.; Pini, G.; Sartoretto, F. (2012): Numerical investigation on direct MLPG for 2d and 3d potential problems. *Computer Modeling in Engineering & Sciences*, vol. 88, pp. 183–210.

Mazzia, A.; Pini, G.; Sartoretto, F. (2014): Meshless techniques for anisotropic diffusion. *APPLIED MATHEMATICS AND COMPUTATION*, vol. 236C, pp. 54–66.

Mazzia, A.; Sartoretto, F. (2010): Meshless solution of potential problems by combining radial basis functions and tensor product ones. *Computer Modeling in Engineering & Sciences*, vol. 68, no. 1, pp. 95–112.

Mirzaei, D.; Schaback, R. (2013): Direct meshless local Petrov–Galerkin (DMLPG) method: A generalized MLS approximation. *Applied Numerical Mathematics*, vol. 68, no. 0, pp. 73 – 82.

Mirzaei, D.; Schaback, R.; Dehghan, M. (2012): On generalized moving least squares and diffuse derivatives. *IMA Journal of Numerical Analysis*, vol. 32, no. 3, pp. 983–1000.

Strang, G. (2007): *Computational Science and Engineering*. Wellesley–Cambridge Press, Wellesley, MA.

Yang, S.-W.; Budarapu, P.; Mahapatra, D.; Bordas, S.; Zi, G.; Rabczuk, T. (2015): A meshless adaptive multiscale method for fracture. *Computational Materials Science*, vol. 96, no. PB, pp. 382–395. cited By 1.