lajss.org© Latin American Journal of Solids and Structures www.lajss.org

A Locally-Continuous Meshless Local Petrov-Galerkin Method Applied to a Two-point Boundary Value Problem

Suzana Matos França de Oliveira[a*], **Laise Lima de Carvalho Sousa**[b], **Creto Augusto Vidal**[c], **Joaquim Bento Cavalcante-Neto**[c]

[a]Universidade Estadual do Piauí, BR-343, Paraíso, Floriano, Piauí, Brasil. E-mail: suzana.matos@frn.uespi.br
[b]Universidade Federal do Ceará - Crateús Campus, BR-226, KM 3, Venâncios, Crateús, Ceará, Brasil. E-mail: laise@crateus.ufc.br
[c]Departamento de Computação, Universidade Federal do Ceará, Block 910 - Pici Campus, Fortaleza, Ceará, Brasil. E-mail: cvidal@dc.ufc.br, joaquimb@dc.ufc.br

*Corresponding author

https://doi.org/10.1590/1679-78256021

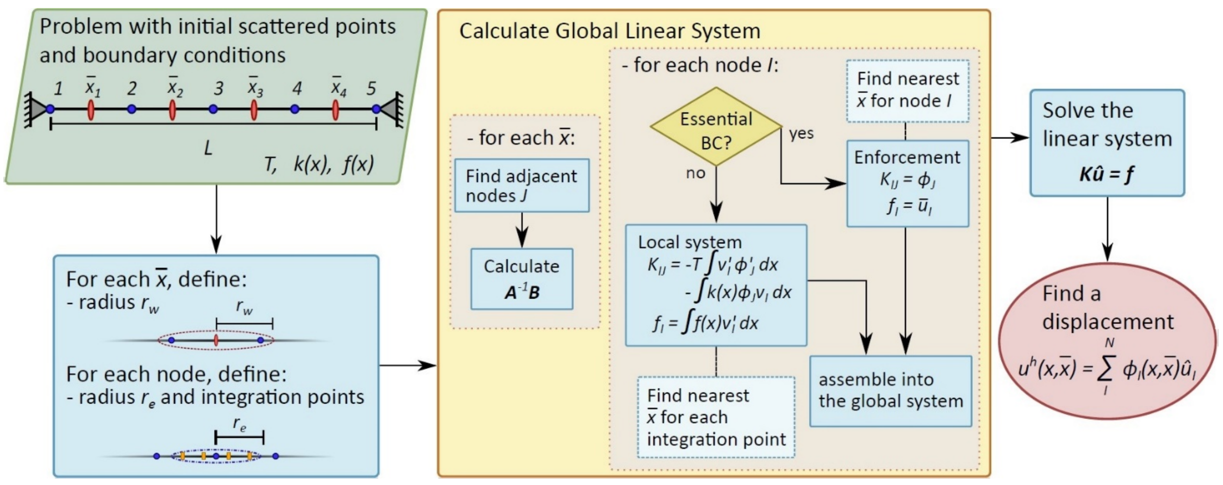
Abstract

In recent years, the Meshless Local Petrov-Galerkin (MLPG) Method has attracted the attention of many researchers in solving several types of boundary value problems. This method is based on a local weak form, evaluated in local subdomains and does not require any mesh, either in the construction of the test and shape functions or in the integration process. However, the shape functions used in MLPG have complicated forms, which makes their computation and their derivative's computation costly. In this work, using the Moving Least Square (MLS) Method, we dissociate the point where the approximating polynomial's coefficients are optimized, from the points where its derivatives are computed. We argue that this approach not only is consistent with the underlying approximation hypothesis, but also makes computation of derivatives simpler. We apply our approach to a two-point boundary value problem and perform several tests to support our claim. The results show that the proposed model is efficient, achieves good precision, and is attractive to be applied to other higher-dimension problems.

Keywords

meshless method, meshless local Petrov-Galerkin (MLPG) method, boundary value problem

Graphical Abstract


Problem with initial scattered points and boundary conditions
1 $\bar{x}_1$ 2 $\bar{x}_2$ 3 $\bar{x}_3$ 4 $\bar{x}_4$ 5
L
T, k(x), f(x)
For each $\bar{x}$, define:
- radius r_w
r_w
For each node, define:
- radius r_e and integration points
r_e
Calculate Global Linear System
- for each $\bar{x}$:
Find adjacent nodes J
Calculate $A^{-1}B$
- for each node I:
Essential BC?
yes
no
Find nearest $\bar{x}$ for node I
Enforcement $K_{IJ} = \phi_J$ $f_I = \bar{u}_I$
Local system $K_{IJ} = -T\int v_I'\phi_J'\,dx - \int k(x)\phi_J v_I\,dx$ $f_I = \int f(x)v_I'\,dx$
Find nearest $\bar{x}$ for each integration point
assemble into the global system
Solve the linear system $K\hat{u} = f$
Find a displacement $u^h(x,\bar{x}) = \sum_I^N \phi_I(x,\bar{x})\hat{u}_I$

Received: March 13, 2020. In Revised Form: October 26, 2020. Accepted: October 28, 2020. Available online: November 06, 2020.
https://doi.org/10.1590/1679-78256021

1 INTRODUCTION

Engineers and scientists often need to solve and simulate physical problems for which analytical solutions do not exist. Therefore, numerical methods are used to approximate those solutions. Among the existing numerical methods, the Finite Element Method (FEM) is one of the most widely used.

However, despite its great applicability, the FEM might have some drawbacks, especially due to its dependence on a good-quality mesh for delivering accurate approximations. Constructing such meshes may require either an intense human intervention or complex automated meshing techniques, which are highly expensive and complicated to perform in 3D domains. Also, sometimes complex remeshing techniques need to be used in the analyses of large deformation and fracture propagation problems (Liu, 2009; Sladek et al., 2013). In order to free the analyses from the problems associated with mesh generation, meshless methods have been developed, in which the domain of the problem is represented through a set of scattered points (nodes), without any explicit connection between them. Thus, in meshless methods, it is simple to include or to remove points, whenever necessary, during iterative computations.

Several meshless methods have already been proposed: smooth particle hydrodynamics (SPH) (Gingold and Monaghan, 1977); element free Galerkin (EFG) (Belytschko et al., 1994); reproducing kernel particle method (RKPM) (Liu et al., 1995); partition of unity finite element method (PUFEM) (Babuska and Melenk, 1997); natural element method (NEM) (Sukumar et al., 1998). However, as Atluri and Zhu (1998) pointed out, they were not truly meshless methods, since they make use of a background mesh for integrating the global weak form.

Atluri and Zhu (1998) proposed the Meshless Local Petrov-Galerkin (MLPG) Method. That method is based on a local weak form, which is evaluated in local subdomains of simple forms, such as line segments, circles, squares and spheres. The method does not use a mesh at all, neither in the construction of test and shape functions nor in the integration process. That is why the authors called it a truly meshless method.

The MLPG has already been used to solve various types of boundary value problems (Amini et al., 2018; Han and Atluri, 2004; Hu and Sun, 2011; Kamranian et al., 2017; Liu et al., 2011; Sheikhi et al., 2019; Zhang et al., 2006). However, in developing those formulations, the authors broke the underlying consistency with the Moving Least-square assumptions, which, in our view, led to shape functions that have unduly complex forms, making their computation and their derivatives' computation quite costly (Liu, 2009; Mirzaei and Schaback, 2013). We believe that such complexity is unnecessary if the formulation maintains consistency with the basic assumptions. Thus, in this work, we show a Least-square-consistent displacement-based Meshless Local Petrov-Galerkin formulation and apply it to the analysis of a two-point boundary value problem.

The remainder of this paper is organized as follows. In Section 2, we introduce the Least-square-consistent displacement-based MLPG, describing the problem to be addressed, the computation of the shape functions, the computation of the shape functions' derivatives, the choice of the test functions, the numerical integration and the enforcement of the essential boundary conditions. In Section 3, we present some tests. Finally, in Section 4, we present our conclusions.

2 LOCALLY-CONTINUOUS MLPG (LC-MLPG) FORMULATION

In this section, we explain the LC-MLPG formulation (see the overview shown in the Graphical Abstract), focusing on how we propose to compute a shape function's derivative. To achieve this goal, we consider a two-point boundary value problem.

2.1 Boundary Value Problem

Consider the ordinary differential equation

$$T\frac{d^2u(x)}{dx^2} - k(x)u(x) + f(x) = 0\,, \tag{1}$$

that governs the problem of a cable, under constant tension T (small deflection theory), subjected to a transverse force per unit length, $f(x)$. The cable is embedded in a medium that provides a stiffness $k(x)$ to its transverse displacement $u(x)$. The domain Ω of the problem is one-dimensional, with $0 < x < L$, and its boundary Γ consists of the two end points $x = 0$ and $x = L$ (Fig. 1).

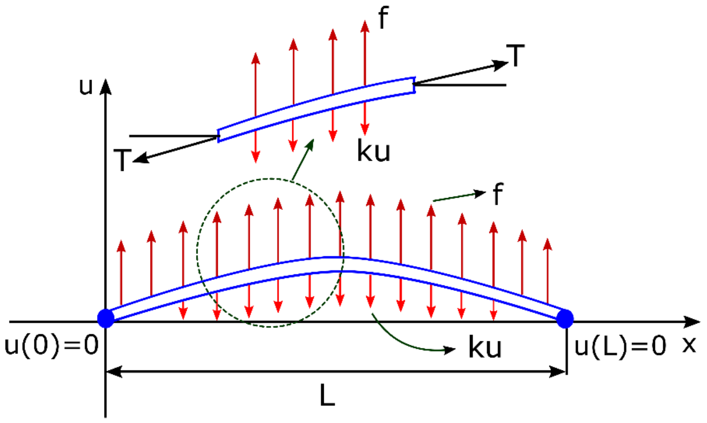


Figure 1: Cable with Dirichlet boundary conditions in the interval $[0, L]$, deflecting upward in the positive y direction. $u(x)$ is the transverse displacement of the cable (deflection), $f(x)$ is the external transverse load, $k(x)$ is the stiffness offered by the medium in which the cable is embedded; and T is a constant tension.

The following essential boundary conditions are assigned at the endpoints:

$$\begin{cases} u(0) = 0 \\ u(L) = 0 \end{cases}. \tag{2}$$

2.2 Local Weak Form

In the MLPG method, the domain and the boundary of the problem are covered with an arbitrary number of scattered nodes. Each node I has a local subdomain, Ω_q^I, with its local boundary, Γ_q^I (Fig. 2). In two and three-dimensional cases, subdomains can have arbitrary shapes. However, in one-dimensional cases, the subdomains are line segments whose union must cover the entire domain of the problem (Atluri and Zhu, 1998).

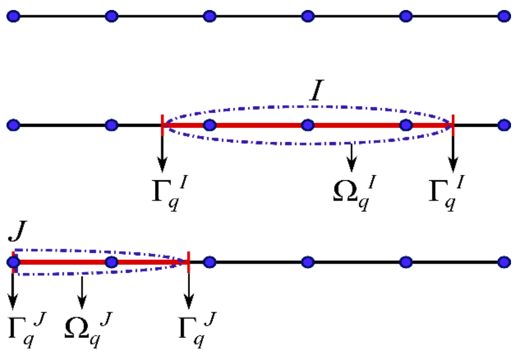


Figure 2: The domain of the cable is covered by nodes scattered arbitrarily. All nodes will have a local subdomain Ω_q with its local boundary Γ_q. Ω_q^I and Ω_q^J represent, in that order, the local subdomains of nodes I and J, and their boundaries are Γ_q^I and Γ_q^J.

The generalized local weak form of the differential equation (Eq. (1)) over the local subdomain Ω_q^I of a node I can be written as

$$\int_{\Omega_q^I} \left[T \frac{d^2u(x)}{dx^2} - k(x)u(x) + f(x) \right] v^I(x)\, dx = 0\,, \tag{3}$$

where $v^I(x)$ is the test function associated with node I.

Equation (3) can be expanded as

$$T \int_{\Omega_q^I} \frac{d^2u(x)}{dx^2} v^I(x)\, dx - \int_{\Omega_q^I} k(x)u(x)\, v^I(x)\, dx + \int_{\Omega_q^I} f(x)\, v^I(x)\, dx = 0\,, \tag{4}$$

which, after integrating the leftmost integral by parts, yields

$$T\left[v^I(x_r)\frac{du(x_r)}{dx} - v^I(x_l)\frac{du(x_l)}{dx} - \int_{\Omega_q^I}\frac{dv^I(x)}{dx}\frac{du(x)}{dx}\,dx\right] - \int_{\Omega_q^I} k(x)u(x)\,v^I(x)\,dx + \int_{\Omega_q^I} f(x)\,v^I(x)\,dx = 0, \tag{5}$$

where x_l and x_r are, respectively, the coordinates at the beginning (left) and at the end (right) of subdomain Ω_q^I.

2.3 Trial Functions

The trial functions are local approximations of the true solution in a given subregion of arbitrary shape of the problem's domain. In this work, we use the moving least squares (MLS) to compute the trial functions' approximations, for its accuracy and simplicity to be extend to problems in higher dimensions. Because of those properties, MLS is commonly used in the literature (Atluri and Shen, 2002; Atluri and Zhu, 1998; Han et al., 2005; Atluri et al., 1999a; Atluri and Zhu, 2000).

Let $u^h(x,\bar{x})$ be the approximation of the true solution $u(x)$ at a point x in the subregion defined in the vicinity of point $\bar{x}$. MLS defines $u^h(x,\bar{x})$ as the following polynomial approximation

$$u^h(x,\bar{x}) = \boldsymbol{p}^T(x)\boldsymbol{a}(\bar{x})\,, \tag{6}$$

where $\boldsymbol{p}^T(x) = [p_1(x) \quad p_2(x) \quad \cdots \quad p_m(x)]$ is a complete monomial basis and $\boldsymbol{a}(\bar{x}) = [a_1(\bar{x}) \quad a_2(\bar{x}) \quad \cdots \quad a_m(\bar{x})]^T$ is the coefficient vector, which should be determined in such a way that the polynomial approximation is optimal, in the Least Squares sense, in the vicinity of $\bar{x}$. In a one-dimensional problem, m is equal to $t + 1$, where t is the degree of the approximating polynomial (Atluri and Shen, 2002). For example, if $t = 2$, then $m = 3$ and $\boldsymbol{p}^T(x) = [1 \quad x \quad x^2]$.

Suppose that, for the construction of the polynomial approximation in the vicinity of $\bar{x}$, we consider a set of N nodes in that vicinity (Fig. 3). Thus, for a given node I in that set, the difference between its exact displacement $u(x_I)$ and its approximate displacement $u^h(x_I,\bar{x})$ is

$$e_I = u(x_I) - u^h(x_I,\bar{x})\,. \tag{7}$$

However, since the exact displacement $u(x_I)$ is not known, we replace it with an unknown pseudo-displacement $\hat{u}_I$ to be determined, and Equation (7) is rewritten as

$$e_I = \hat{u}_I - u^h(x_I,\bar{x})\,. \tag{8}$$

Substituting Equation (6) into Equation (8) yields

$$e_I = \hat{u}_I - \boldsymbol{p}^T(x_I)\,\boldsymbol{a}(\bar{x})\,. \tag{9}$$

Considering the vector of errors for the N nodes in the vicinity of $\bar{x}$

$$\boldsymbol{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix}, \tag{10}$$

we want to find the coefficients of the approximating polynomial such that the L_2 norm of the error vector is minimized. However, instead of using the vector of errors of Equation (10), we construct a vector of weighted errors, to attribute more importance to the nodes that are closer to $\bar{x}$ (Fig. 3). Thus, for node I, the weight is defined as

$$w_I = w(|x_I - \bar{x}|)\,, \tag{11}$$

where $w(r)$ is a bell-shaped weight function on the radial distance, $r = |x - \bar{x}|$, from point $\bar{x}$. Therefore, the vector of weighted errors is written as

$$\bar{\boldsymbol{e}} = \begin{bmatrix} w_1 e_1 \\ w_2 e_2 \\ \vdots \\ w_N e_N \end{bmatrix}. \tag{12}$$

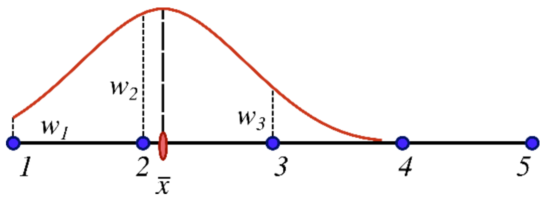


Figure 3: Influenced nodes in the vicinity of $\bar{x}$: nodes 1, 2 and 3. Node 2 has the strongest influence on $\bar{x}$ and node 1 has the least influence on $\bar{x}$.

In order to find the coefficients $\boldsymbol{a}(\bar{x})$ of the approximating polynomial, we minimize the squared L_2 norm of the vector of weighted errors (Notice that, in (Atluri and Zhu, 1998), the weighted squared errors are minimized while we minimize the squared weighted errors). Thus, defining the functional

$$\mathrm{J}\big(\boldsymbol{a}(\bar{x})\big) = \|\bar{\boldsymbol{e}}\|^2 = \textstyle\sum_{I=1}^{N}\big[w_I\big(\hat{u}_I - \boldsymbol{p}^T(x_I)\,\boldsymbol{a}(\bar{x})\big)\big]^2, \tag{13}$$

we pose the following unconstrained minimization problem:

$$\underset{\boldsymbol{a}(\bar{x})}{\text{minimize}}\,\mathrm{J}\big(\boldsymbol{a}(\bar{x})\big), \tag{14}$$

to find the optimum coefficients for $u^h(x,\bar{x})$, using the first order necessary condition

$$\nabla_{\boldsymbol{a}}\mathrm{J} = \boldsymbol{0}. \tag{15}$$

Substituting Equations (6) to (12) into (15) yields the following

$$\nabla_{\boldsymbol{a}}\mathrm{J} = \nabla_{\boldsymbol{a}}(\bar{\boldsymbol{e}}^T\bar{\boldsymbol{e}}) = 2\bar{\boldsymbol{e}}^T\nabla_{\boldsymbol{a}}(\bar{\boldsymbol{e}}) = 2\boldsymbol{e}^T\boldsymbol{W}\nabla_{\boldsymbol{a}}(\boldsymbol{e}) = 2\boldsymbol{e}^T\boldsymbol{W}(-\boldsymbol{P}) = -2\boldsymbol{P}^T\boldsymbol{W}^T(\boldsymbol{e}) = -2\boldsymbol{P}^T\boldsymbol{W}^T(\hat{\boldsymbol{u}} - \boldsymbol{P}\boldsymbol{a}) = \boldsymbol{0}. \tag{16}$$

Thus, at $\bar{x}$, and knowing that $\boldsymbol{W}$ is a diagonal matrix,

$$\boldsymbol{P}^{\mathrm{T}}\boldsymbol{W}(\bar{x})\big(\hat{\boldsymbol{u}} - \boldsymbol{P}\boldsymbol{a}(\bar{x})\big) = \boldsymbol{0} \Rightarrow \boldsymbol{P}^{\mathrm{T}}\boldsymbol{W}(\bar{x})\boldsymbol{P}\boldsymbol{a}(\bar{x}) = \boldsymbol{P}^{\mathrm{T}}\boldsymbol{W}(\bar{x})\hat{\boldsymbol{u}} \Rightarrow \boldsymbol{A}(\bar{x})\boldsymbol{a}(\bar{x}) = \boldsymbol{B}(\bar{x})\hat{\boldsymbol{u}}, \tag{17}$$

where $\boldsymbol{A}(\bar{x}) = \boldsymbol{P}^{\mathrm{T}}\boldsymbol{W}(\bar{x})\boldsymbol{P}$ and $\boldsymbol{B}(\bar{x}) = \boldsymbol{P}^{\mathrm{T}}\boldsymbol{W}(\bar{x})$, with

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{p}^T(e_1) \\ \boldsymbol{p}^T(e_2) \\ \vdots \\ \boldsymbol{p}^T(e_N) \end{bmatrix}, \tag{18}$$

$$\boldsymbol{W}(\bar{x}) = \begin{bmatrix} w_1^2 & 0 & \cdots & 0 \\ 0 & w_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_N^2 \end{bmatrix} \tag{19}$$

and

$$\hat{\boldsymbol{u}} = [\hat{u}_1 \quad \hat{u}_2 \quad \cdots \quad \hat{u}_N]^T. \tag{20}$$

After solving the system of algebraic linear equations, the optimized coefficients are

$$\boldsymbol{a}(\bar{x}) = \boldsymbol{A}^{-1}(\bar{x})\boldsymbol{B}(\bar{x})\hat{\boldsymbol{u}} \,. \tag{21}$$

2.3.1 Shape Functions

Substituting Equation (21) into Equation (6) yields

$$u^h(x,\bar{x}) = \boldsymbol{p}^T(x)\boldsymbol{A}^{-1}(\bar{x})\boldsymbol{B}(\bar{x})\hat{\boldsymbol{u}} = \boldsymbol{\phi}^T(x,\bar{x})\hat{\boldsymbol{u}} \,, \tag{22}$$

where $\boldsymbol{\phi}^T(x,\bar{x})$ represents the shape function vector,

$$\boldsymbol{\phi}^T(x,\bar{x}) = \boldsymbol{p}^T(x)\boldsymbol{A}^{-1}(\bar{x})\boldsymbol{B}(\bar{x}) = [\phi_1(x,\bar{x}) \quad \phi_2(x,\bar{x}) \quad \cdots \quad \phi_N(x,\bar{x})] \,, \tag{23}$$

so that $\phi_I(x,\bar{x})$, with $I = 1,\cdots,N$, denotes the shape function for node I.

Considering Equation (23), Equation (22) can be rewritten as

$$u^h(x,\bar{x}) = \sum_{I=1}^{N} \phi_I(x,\bar{x})\hat{u}_I \,, \tag{24}$$

in which $\phi_I(x,\bar{x})$ is defined as

$$\phi_I(x,\bar{x}) = \sum_{j=1}^{m} p_j(x)[\boldsymbol{A}^{-1}(\bar{x})\boldsymbol{B}(\bar{x})]_{jI} \,, \tag{25}$$

where $p_j(x)$ is the j-th term of the polynomial basis $\boldsymbol{p}(x)$ and $[\boldsymbol{A}^{-1}(\bar{x})\boldsymbol{B}(\bar{x})]_{jI}$ is the element (j,I) of matrix $[\boldsymbol{A}^{-1}(\bar{x})\boldsymbol{B}(\bar{x})]$. Note that, in order to compute the shape functions, the $\boldsymbol{A}$ matrix needs to be invertible. To ensure that condition, the selected number of nodes, N, in the vicinity of $\bar{x}$ has to be greater than or equal to the number of monomials in the polynomial basis, i.e., $N \geq m$ (see (Atluri and Shen, 2002; Liu, 2009)).

2.3.2 Derivatives of the Shape Functions

The derivative of the shape functions with respect to x are calculated as

$$\phi_I'(x,\bar{x}) = \sum_{j=1}^{m} p_j'(x)[\boldsymbol{A}^{-1}(\bar{x})\boldsymbol{B}(\bar{x})]_{jI} \,, \tag{26}$$

where

$$p_j'(x) = \frac{dp_j(x)}{dx} \,.$$

Notice that, to compute the derivative of a shape function at $\bar{x}$, we simply compute $\phi_I'(\bar{x},\bar{x})$, using Equation (26). This is totally consistent with the formulation developed for the approximating polynomial and its coefficients (Eqs. (6) to (24)). However, methods, such as MLPG1 (Atluri et al., 1999b), use the polynomial in (6) just as an auxiliary polynomial to construct the trial function at $\bar{x}$, i.e.,

$$u^{\text{trial}}(\bar{x}) = u^h(x,\bar{x}) = \boldsymbol{p}^T(x)\boldsymbol{A}^{-1}(\bar{x})\boldsymbol{B}(\bar{x})\hat{\boldsymbol{u}} = \boldsymbol{\phi}^T(\bar{x},\bar{x})\hat{\boldsymbol{u}} \,. \tag{27}$$

Those methods assume that the trial function $u^{\text{trial}}(\bar{x})$ computed in such a way (Eq. (27)) is continuous and differentiable everywhere ($\forall\, \bar{x}$ -- since $\bar{x}$ is the new variable of domain) with respect to $\bar{x}$. Since the optimization of the coefficients of the auxiliary polynomial is based on a discrete set of nodes in the vicinity of $\bar{x}$, it is not guaranteed that the same set of discrete nodes is used to optimize the coefficients of the auxiliary polynomial at $\bar{x} + \mathrm{d}\bar{x}$. So, in that sense, there is an inconsistency between the discrete nature of the optimization process and the continuity assumption required to ensure the differentiability of the matrices $\boldsymbol{A}^{-1}$ and $\boldsymbol{B}$ in (27).

2.3.3 Weight Function

In this work, the weight function is a bell-shaped function, whose support region is centered at $\bar{x}$. Thus, any node that falls outside that support region will not contribute to the trial function associated with the subregion around $\bar{x}$. We use the fourth order spline with compact support as weight function (Atluri and Shen, 2002; Atluri et al., 1999b; Sladek et al., 2010), i.e.,

$$w(r) = \begin{cases} 1 - 6\left(\frac{r}{r_s}\right)^2 + 8\left(\frac{r}{r_s}\right)^3 - 3\left(\frac{r}{r_s}\right)^4 , & 0 \le r \le r_s \, ; \\ 0 \, , & r > r_s \, ; \end{cases} \tag{28}$$

where $r = \|x - \bar{x}\|$ is the distance from a point in the location x to point $\bar{x}$ and r_s is the size (radius) of the support region of the weight function centered at $\bar{x}$. For this particular case, r_s is referred to as $r_w(\bar{x})$. The size $r_w(\bar{x})$ is determined, as explained in Section 2.3.1, to contain the N required points to ensure that matrix $\boldsymbol{A}$ is invertible ($N \ge m$). However, $r_w(\bar{x})$ should also be small enough to maintain the local nature of the MLS approach (Atluri and Shen, 2002).

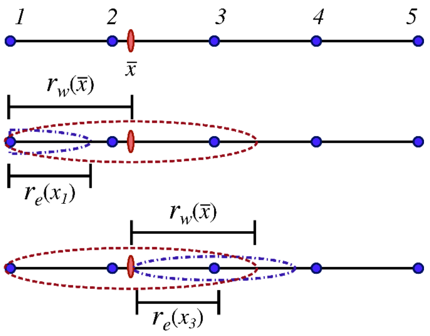


Figure 4: Support zones for the test functions, $r_e(x_I)$, and for the trial functions, $r_w(\bar{x})$: $r_e(x_I)$ of nodes 1 and 3 and $r_w(\bar{x})$ centered at $\bar{x}$.

2.4 Test Functions

In MLPG the trial and test functions may be chosen from different function spaces. The test functions usually have compact support, which causes the integral to be calculated in a limited region, where the function is non-zero. Different test functions result in different MLPG methods (Atluri and Shen, 2002). In this work, the test functions are also bell-shaped functions centered at the nodes (the same as the weight function of Eq. (28) with $r = \|x - x_I\|$ and $r_s = r_e(x_I)$), resulting in the so-called MLPG1. However, $r_w(\bar{x})$ need not be equal to $r_e(x_I)$ (Fig. 4). The support of a test function is usually confined to a region around its associated node, such that all the neighboring nodes fall outside that region.

2.5 Discretization

Substituting Equation (24) into Equation (5) the local weak form for a node I can be rewritten as

$$\sum_{J=1}^{N} \left\{ T \left[v^I(x_r)\phi_J'(x_r,\bar{x}\,)\,\hat{u}_J \; - \; v^I(x_l)\phi_J'(x_l,\bar{x})\,\hat{u}_J \; - \int_{\Omega_q^I} v'^I(x)\phi_J'(x,\bar{x})\hat{u}_J dx \right] - \int_{\Omega_q^I} k(x)\phi_J(x,\bar{x})\hat{u}_J v^I(x) dx \right\}$$

$$+ \int_{\Omega_q^I} f(x) v^I(x) dx = 0 \, . \tag{29}$$

This equation can be simplified into the following linear algebraic equation in $\hat{u}_J$

$$\textstyle\sum_{J=1}^{N} K_{IJ}\hat{u}_J = f_I \, , \tag{30}$$

where

$$K_{IJ} = T \left[v^I(x_r)\phi_J'(x_r,\bar{x}\,)\,\hat{u}_J \; - \; v^I(x_l)\phi_J'(x_l,\bar{x})\,\hat{u}_J \; - \textstyle\int_{\Omega_q^I} v'^I(x)\phi_J'(x,\bar{x})\hat{u}_J dx \right] - \textstyle\int_{\Omega_q^I} k(x)\phi_J(x,\bar{x})\hat{u}_J v^I(x) dx \tag{31}$$

and

$$f_I = -\int_{\Omega_q^I} f(x) v^I(x) dx \,. \tag{32}$$

The test function $v^I(x)$ is chosen to vanish in the support boundary, then Equation (31) can be simplified. Therefore, for internal nodes (Fig. 4, nodes 2 to 4), we have

$$K_{IJ} = -T \int_{\Omega_q^I} v'^I(x) \phi_J'(x,\bar{x}) \hat{u}_J dx - \int_{\Omega_q^I} k(x) \phi_J(x,\bar{x}) \hat{u}_J v^I(x) dx \,; \tag{33}$$

for the left node $x_l = 0$ (Fig. 4, node 1), then

$$K_{IJ} = T\left[-\phi_J'(0,\bar{x})\, \hat{u}_J - \int_{\Omega_q^I} v'^I(x) \phi_J'(x,\bar{x}) \hat{u}_J dx \right] - \int_{\Omega_q^I} k(x) \phi_J(x,\bar{x}) \hat{u}_J v^I(x) dx \,; \tag{34}$$

and, for the right node $x_r = L$ (Fig. 4, node 5), then:

$$K_{IJ} = T\left[\phi_J'(L,\bar{x})\, \hat{u}_J - \int_{\Omega_q^I} v'^I(x) \phi_J'(x,\bar{x}) \hat{u}_J dx \right] - \int_{\Omega_q^I} k(x) \phi_J(x,\bar{x}) \hat{u}_J v^I(x) dx \,. \tag{35}$$

Applying Equation (30) to all n nodes in the problem's domain results in set of n equations that, when grouped, constitute the final system of global equations

$$\boldsymbol{K}_{(n\times n)} \hat{\boldsymbol{u}}_{(n\times 1)} = \boldsymbol{f}_{(n\times 1)} \,.$$

After solving this system, the displacement at any point in the domain can be obtained through Equation (24), making use of the displacements computed for the nodes in the support domain of that point.

2.6 Numerical Integration

The integrals are calculate using Gauss-Legendre quadrature. Thus, for the internal nodes (Eqs. (33) and (32)), making the necessary parameterization,

$$K_{IJ} = -r_e \sum_{q=1}^{LP} \omega_q \left[T\, v'^I\left(x(\varepsilon_q)\right) \phi_J'\left(x(\varepsilon_q),\bar{x}\right) + k\left(x(\varepsilon_q)\right) \phi_J\left(x(\varepsilon_q),\bar{x}\right) v^I\left(x(\varepsilon_q)\right) \right] \tag{36}$$

and

$$f_I = -r_e \sum_{q=1}^{LP} \omega_q f\left(x(\varepsilon_q)\right) v^I\left(x(\varepsilon_q)\right) \,, \tag{37}$$

where ε_q, $q = 1, \dots, LP$, are the Legendre quadrature points, ω_q are their associated weights, and $x(\varepsilon_q) = x_I + r_e \varepsilon_q$; for the left node (Eqs. (34) and (32)),

$$K_{IJ} = -\frac{r_e}{2} \sum_{q=1}^{LP} \omega_q \left[T\, v'^I\left(x(\varepsilon_q)\right) \phi_J'\left(x(\varepsilon_q),\bar{x}\right) + k\left(x(\varepsilon_q)\right) \phi_J\left(x(\varepsilon_q),\bar{x}\right) v^I\left(x(\varepsilon_q)\right) \right] - T\phi_J'(0,\bar{x}) \tag{38}$$

and

$$f_I = -\frac{r_e}{2} \sum_{q=1}^{LP} \omega_q f\left(x(\varepsilon_q)\right) v^I\left(x(\varepsilon_q)\right) , \tag{39}$$

where $x(\varepsilon_q) = \frac{r_e}{2}(1+\varepsilon_q)$; and for the right node (Eqs. (35) and (32)),

$$K_{IJ} = -\frac{r_e}{2} \sum_{q=1}^{LP} \omega_q \left[T\, v'^I\left(x(\varepsilon_q)\right) \phi_J'\left(x(\varepsilon_q),\bar{x}\right) + k\left(x(\varepsilon_q)\right) \phi_J\left(x(\varepsilon_q),\bar{x}\right) v^I\left(x(\varepsilon_q)\right) \right] + T\phi_J'(L,\bar{x}) \,, \tag{40}$$

where f_I is computed using Equation (39), and $x(\varepsilon_q) = \left(L - \frac{r_e}{2}\right) + \frac{r_e}{2}\varepsilon_q$.

Notice that, in the computation of K_{IJ}, the value of $\bar{x}$ to be used is the one adopted for the trial function in whose definition node J takes part. It is possible, for $I = J$, that the node takes part in multiple trial functions. In those cases, their contributions to K_{IJ} should be added.

2.7 Enforcement of the Essential Boundary Conditions

The trial functions based on MLS, as described in Section 2.3, are not interpolating functions. Therefore, the shape functions associated with each node do not possess the Kronecker Delta property (Fig. 5). Thus, some special method is required to impose the essential boundary conditions. Although the most popular methods for that are the penalty method and the method of Lagrange multipliers, we use the MLS collocation method proposed in (Liu, 2009; Mirzaei, 2015), where:

$$u^h(0,\bar{x}) = \sum_{J=1}^{N} \phi_J(0,\bar{x})\hat{u}_J = 0 \tag{41}$$

and

$$u^h(L,\bar{x}) = \sum_{J=1}^{N} \phi_J(L,\bar{x})\hat{u}_J = 0\,. \tag{42}$$

Equations (41) and (42) replace Eqs. (38), (39) and (40) for the left and right nodes.

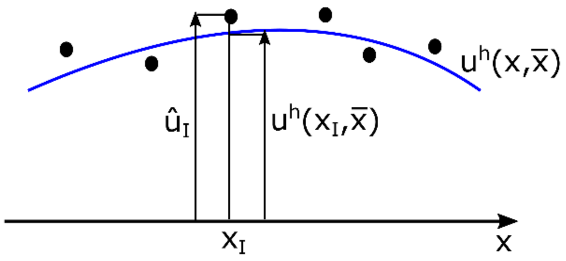


Figure 5: Distinction between $\hat{u}_I$ and $u^h(x_I,\bar{x})$ in the MLS approximation

2.8 Algorithm

The main implementation of the Locally-Consistent Meshless Local Petrov-Galerkin (LC-MLPG) method followed the Algorithm 1.

Algorithm 1: LC-MLPG

```
Choose a finite number of nodes in the domain $\Omega$ and on the boundary $\Gamma$ and a finite number of $\bar{x}$ in the domain $\Omega$.
Define the monomial basis and the radii $r_e(x_I)$, $r_w(\bar{x})$ respectively for the nodes and for $\bar{x}$.

Loop over all $\bar{x}$:
    Determine all nodes $x_J$ in the vicinity, i.e., those nodes with $w(|x_J - \bar{x}|) > 0$ and calculate the matrix $\boldsymbol{A}^{-1}\boldsymbol{B}$ (Eq. (21)).

Loop over all nodes $I$ without prescribed displacement:
    Loop over all integration points $q$:
        Calculate local force vector $f_I$ (Eq. (37)).
        Find the nearest $\bar{x}$ from integration point $q$.
        Loop over all $x_J$ in the vicinity of $\bar{x}$:
            Calculate local stiffness matrix $K_{IJ}$ (Eq. (36)).
        Assemble local contributions into the global linear system $\boldsymbol{K}$ and $\boldsymbol{f}$.

Loop over all nodes $I$ with prescribed displacement:
    Find the nearest $\bar{x}$ from node $I$.
    Loop over all x_J in the vicinity of $\bar{x}$:
        Enforce prescribed displacement (Eqs. (41) and (42)).

Solve the global linear system $\boldsymbol{K}\hat{\boldsymbol{u}} = \boldsymbol{f}$.

Calculate the displacement at any point $x$ using Equation (24)
```

3 TESTS AND RESULTS

In this section, several tests demonstrate the advantages of our Least-Square-Consistent Meshless Local Petrov-Galerkin formulation over the traditional MLPG1 formulation. Both methods were implemented in C++, and the Eigen Library (Guennebaud et al., 2019) was used to handle matrix operations and to solve the linear system of algebraic equations. The simulations were performed in an Intel® Core™ i7-4500U CPU @ 1.80GHz × 4, 8.0GB RAM with the system Ubuntu 18.04.2 LTS.

In the following subsections, we compare our method with the original MLPG1 (Atluri and Shen, 2002; Atluri and Zhu, 1998), using a simple problem of a cable with fixed ends illustrated in Figure 1. In Section 3.1, we compare the relative errors for various parameter settings which are obtained by changes of: Polynomial order (t), number of nodes, number of integration points, support radius of the test function and support radius of the trial functions. The error is measured against the analytical solution given in Equation (43). In Section 3.2, we fix all the parameters and vary the polynomial order of the trial function. Again, the results are compared against the analytical solution. In Section 3.3, we repeat the tests of Section 3.2 using a non-constant $k(x)$ function. In that case, the ground truth for computing the error is the solution obtained by the finite difference method with a very fine discretization. In Section 3.4, we compare the computational performances of the approaches used in the previous subsections.

3.1 Parametric Error Analysis

For the analyzed problem shown in Figure 1, when $k(x)$ and $f(x)$ are constant, i.e., $k(x) = k$ and $f(x) = f$, the analytical solution $u(x)$ can be written as (Buchanan, 1994)

$$u(x) = \frac{f[\cosh(\gamma L)-1]\sinh(\gamma x)}{k\sinh(\gamma L)} - \frac{f\cosh(\gamma x)}{k} + \frac{f}{k}, \tag{43}$$

where $\gamma^2 = k/T$. The radii r_e and r_w of the support regions for the test and trial functions, respectively, are written as

$\mathrm{r_e}(\alpha) = \alpha d_{\min}$

and

$\mathrm{r_w}(\alpha) = \beta d_{\min}$,

where $d_{\min}$ is the minimum distance from a given node to its closest adjacent node.

In this section, we present the results of a parametric error analysis in which we report the effects of varying the number nodes, the polynomial order of the trial function and the number of integration points for several values of α and β. The relative errors are computed as

$$\mathrm{error} = \frac{\|\boldsymbol{u} - \boldsymbol{u}^{\mathrm{h}}\|}{\|\boldsymbol{u}\|} \times 100\,, \tag{44}$$

where $\boldsymbol{u}$ and $\boldsymbol{u}^{\mathrm{h}}$ are vectors whose components are the displacements at discrete points of the domain computed, respectively, with the analytical and the numerical solutions.

In this section, the tests use the following parameters: $L = 3$m, $T = 20$N, $k = 2$N/m^2 and $f = 10$N/m. The tests are run with the following values of α and β: $\alpha = 0.1$ to 1.0, with steps of 0.1 and $\beta = 0.1$ to 4.0, with steps of 0.1. However, to avoid unnecessary clutter in the plots, we show only four values of α ($0.1, 0.4, 0.7$ and 1.0). The upper limit for β was established as 4.0 to ensure enough range for the cubic polynomial order case, while still keeping the locality of the trial functions. We ran the same tests with 7, 13, 25 and 49 nodes. Those number of nodes were chosen so that the same set of tests could be performed for both the quadratic and the cubic order trial functions.

3.1.1 Trial functions of second degree ($\boldsymbol{t} = \mathbf{2}$)

In our method, $\bar{x}$ is located as illustrated in Figures 6a and 6b for the 7-node case, ensuring that matrix $\boldsymbol{A}$ has an inverse ($N \geq 3$). First, we consider the minimum number of trial functions, i.e., with $\bar{x}$ located at every other node (Fig. 6a). Then, we tested trial functions with $\bar{x}$ located at each internal node (Fig. 6b) to show the flexibility of the method. The distribution of trial functions for the 13, 25 and 49-node cases follows the same pattern shown in Figures 6a and 6b.

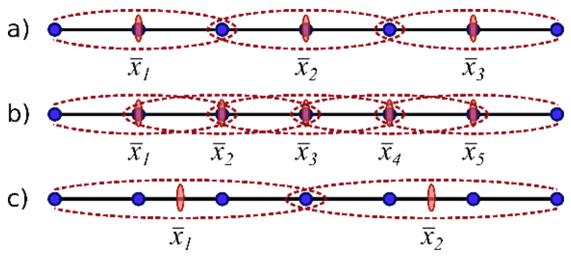


Figure 6: Location of $\bar{x}$ for the 7-node case with a quadratic polynomial order (a and b) and with a cubic polynomial order (c).

In the original MLPG1, $\bar{x}$ is fused with x, requiring a new computation of the shape functions for every integration point (see (Atluri and Zhu, 1998; Atluri et al., 1999b).

The results plotted in Figures 7 and 8 allow us to make some general observations. First, notice that, for our method, the error diminishes as the number of nodes and the number of integration points increase. However, MLPG1 shows a similar behavior only for some specific values of α. Second, MLPG1's error is highly sensitive to variations of β and α, especially for a small number of nodes (for example, 7 and 13 nodes in the tests), making it difficult to suggest adequate values for those parameters. On the other hand, our method shows very smooth error curves, which are almost insensitive to changes in β for a given value of α. Third, MLPG1 requires a larger starting value of β to be able to invert matrix $\boldsymbol{A}$. For smaller number of nodes, this implies an undue restriction on the locality of the solution.

We also want to point out some more specific observations from Figures 7 and 8: 1) for all the tests (with 7, 13, 25 and 49 nodes), regardless of the value of α, MLPG1 requires β to be at least 2.1 for $\boldsymbol{A}$ to have an inverse, while our method requires a β of 1.1; 2) The dependence relation of the error on the $\alpha \times \beta$-combination, on the number of nodes and on the number of integration points is very well-behaved in our method, and shows that the error diminishes as the number of nodes and integration points increase for any values of α and β. On the other hand, that dependence relation in MLPG1 is not well-behaved when the number of points is small (e.g., 7 and 13 points). Notice that, in those cases, for the whole range of β a given value of α is not always the best choice, while our method shows clearly the best α to choose.

3.1.2 Trial functions of third degree ($\boldsymbol{t = 3}$)

Since the results for different distributions of trial functions (locations of $\bar{x}$) in the quadratic case were equivalent, for the trial functions of third degree, $\bar{x}$ is located as illustrated in Figure 6c for the 7-node case, ensuring that matrix $\boldsymbol{A}$ has an inverse ($N \geq 4$). The distribution of the trial functions for the 13, 25 and 49-node cases follows the same pattern shown in Figure 6c.

Once again, notice that, in MLPG1 $\bar{x}$ is fused with x, requiring a new computation of the shape functions for every integration point.

The results plotted in Figure 9 allow us to make some general observations. First, notice that, for our method, the error diminishes as the number of nodes and integration points increase. However, MLPG1 shows a similar behavior only for some values of α. Second, MLPG1's error is highly sensitive to variations of β and α, making it difficult to suggest adequate values for those parameters. On the other hand, our method shows very smooth error curves, which are almost insensitive to changes in β for a given value of α. Third, MLPG1 requires a larger starting value of β to be able to invert matrix $\boldsymbol{A}$. For smaller number of nodes, this implies an undue restriction on the locality of the solution.

Again, we want to point out some more specific observations from Figure 9: 1) for the test with 7, 13 and 25 nodes, regardless of the value of α, MLPG1 requires β to be at least 3.1 for $\boldsymbol{A}$ to have an inverse, while our method requires $\beta = 1.6$. With 49 nodes, MLPG1 requires β to be at least 3.2 for $\boldsymbol{A}$ to have an inverse, while our method requires $\beta = 1.9$; 2) The dependence relation of the error on the combination of α, β, number of nodes and number of integration points is very well-behaved in our method, and shows that the error diminishes as the number of nodes and the number of integration points increase for any values of α and β. On the other hand, that dependence relation in MLPG1 is very erratic and, for some values of α, it deteriorates as the number of nodes and the number of integration points increase, which is a rather inconsistent behavior.

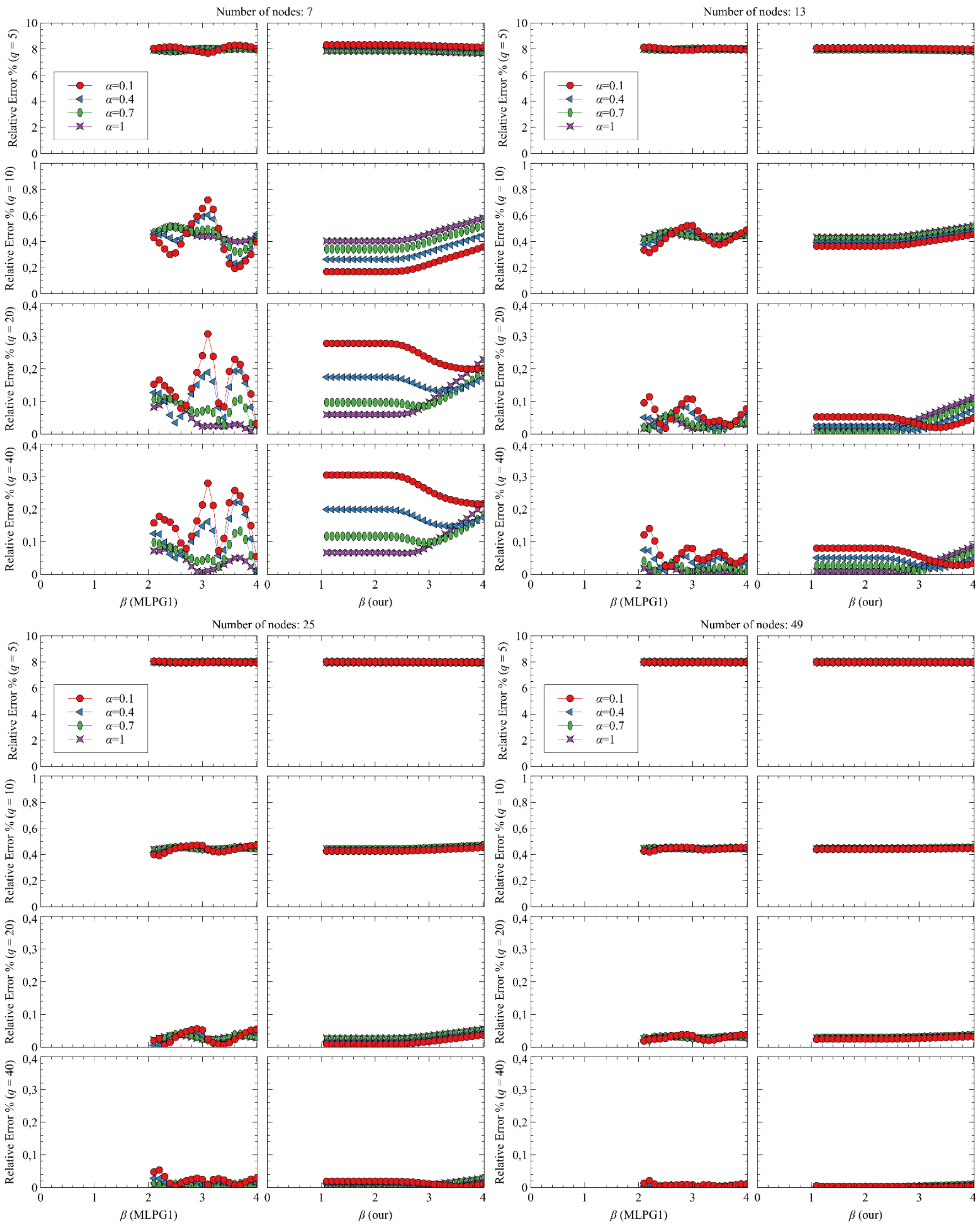


Figure 7: Exhaustive test with quadratic polynomial base using 7, 13, 25 and 49 nodes, where $\bar{x}$ is located in accordance with the pattern shown in Figure 6a. The graphs on the left columns are from the original MLPG1 method and on the right columns are from our approach. The tests are performed with 5, 10, 20 and 40 integration points as shown in each row. The y-axis represents the percentage relative error (Eq. (44)), the x-axis shows the values of β, and each curve corresponds to a specific value of α.

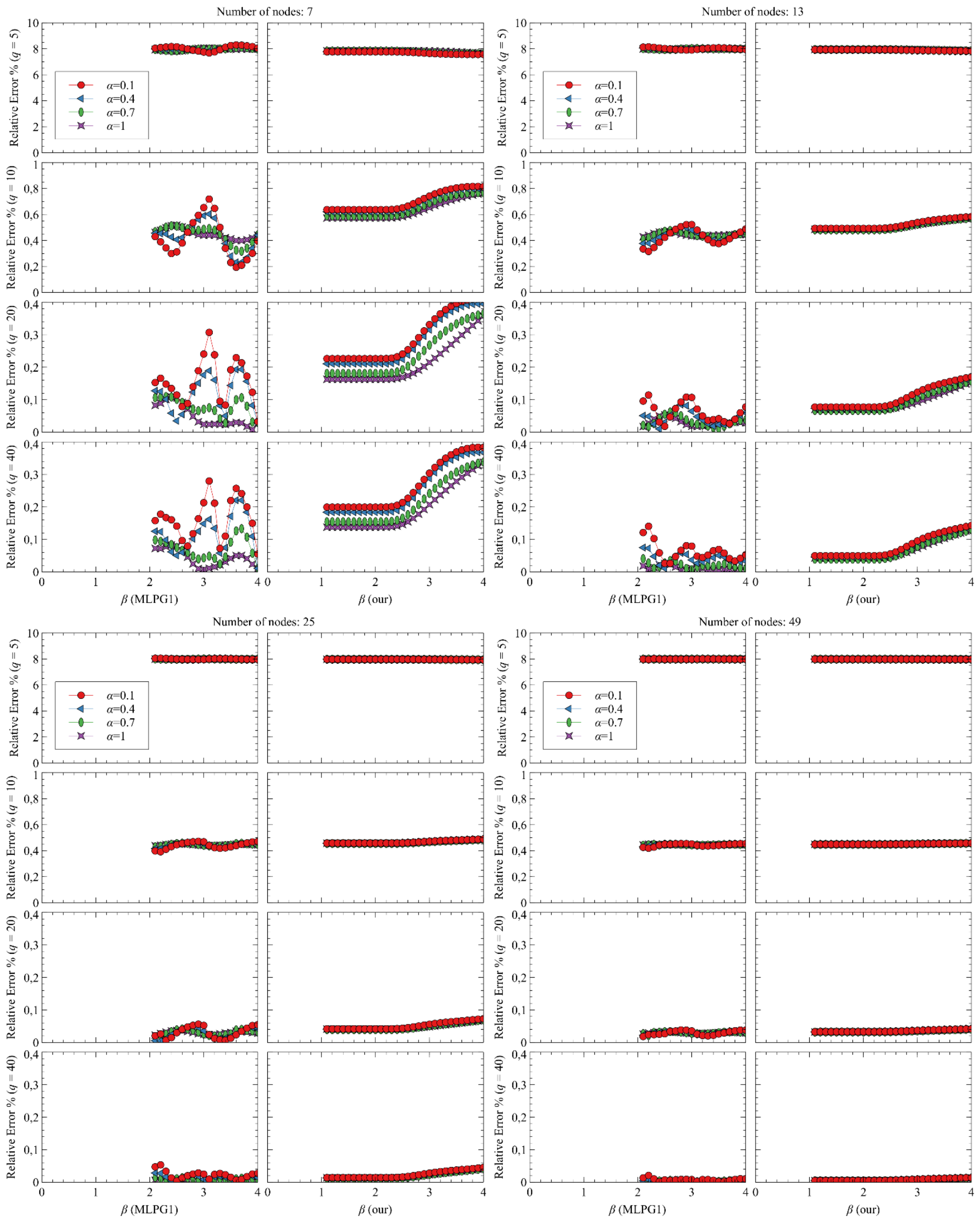


Figure 8: Exhaustive test with quadratic polynomial base using 7, 13, 25 and 49 nodes, where $\bar{x}$ is located in accordance with the pattern shown in Figure 6b. The graphs on the left columns are from the original MLPG1 method and on the right columns are from our approach. The tests are performed with 5, 10, 20 and 40 integration points as shown in each row. The y-axis represents the percentage relative error (Eq. (44)), the x-axis shows the values of β, and each curve corresponds to a specific value of α.

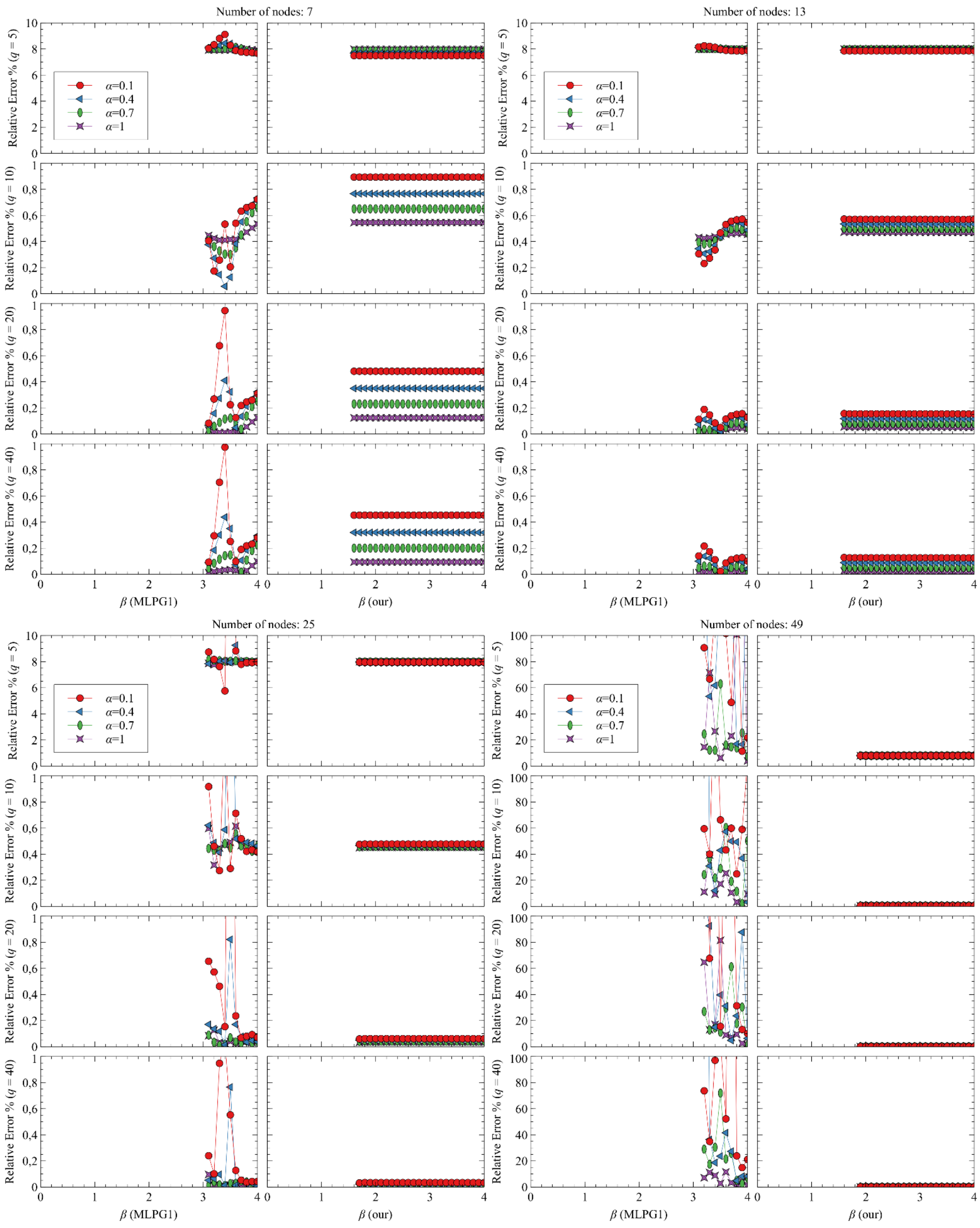


Figure 9: Exhaustive test with cubic polynomial base using 7, 13, 25 and 49 nodes, where $\bar{x}$ is located in accordance with the pattern shown in Figure 6c. The graphs on the left columns are from the original MLPG1 method and on the right columns are from our approach. The tests are performed with 5, 10, 20 and 40 integration points as shown in each row. The y-axis represents the percentage relative error (Eq. (44)), the x-axis shows the values of β, and each curve corresponds to a specific value of α.

3.2 Varying t with Constant $k(x)$

In this test, we use specific values of α and β based on the results shown in Section 3.1 and compare both our solution and MLPG1's solution against the exact solution given in Equation (43).

Once again, we separate the results according to the polynomial degree t of the trial functions. We present results for different numbers of integration points ($q = 5, 10, 20$ and 40) and nodes ($n = 7$, 13, 25 and 49).

The value of α was fixed at 0.7 for the sake of MLPG1 (most stable results), although for our method $\alpha = 1.0$ would be the best choice. Since the value of β determines the existence, or not, of the inverse of matrix $\boldsymbol{A}$ (Eq. (21)), it depends on the value of t. Thus, we chose the smallest value of β required by each method.

3.2.1 Trial function of second degree ($\boldsymbol{t = 2}$)

In these tests, we use $\beta = 2.1$ for the original MLPG1, and $\beta = 1.1$ for our method (those were the minimum required values to invert matrix $\boldsymbol{A}$). Notice (see Fig. 7) that, if we also used $\beta = 2.1$ in our method, the results would not be any different. For our method, we use the trial function distribution pattern shown in Figure 6a.

The results are shown in Figure 10. Notice that, with 10 integration points, all the results (any number of nodes) are visually indistinguishable from the exact solution. However, as we point out in Section 3.4, our method is much more efficient than MLPG1.

3.2.2 Trial functions of third degree ($\boldsymbol{t = 3}$)

In these tests, we use $\beta = 3.2$ for the original MLPG1, and $\beta = 1.9$ for our method (those were the minimum required values to invert matrix $\boldsymbol{A}$). For our method, we use the trial function distribution pattern shown in Figure 6c.

The results are shown in Figure 11. Notice that, with 10 integration points, all the results from 5 up to 25 nodes are visually indistinguishable from the exact solution. However, with 49 nodes, MLPG1 shows inconsistent behavior regardless of the number of integration points (see also Fig. 9, 49-node case). Our method, on the other hand, maintains consistency in all cases. Moreover, as we point out in Section 3.4, our method is much more efficient than MLPG1.

3.2.3 Summarized results at the midpoint

The midpoint in the current problem, $x = 1.5$m, has a displacement of 0.5142m (see the exact solution in Eq. (43)). Table 1 shows comparative results for the displacement of that point, including the value of the relative error, i.e.

$$\text{error} = \left|\frac{u - u^{\text{h}}}{u}\right| \times 100\,, \tag{45}$$

Table 1 Approximate displacements of the cable's midpoint. The exact displacement is 0.5142m. All the examples use 40 integration points and $\alpha = 0.7$. The values of β are: for the quadratic cases, $\beta = 2.1$ (original MLPG1 method) and $\beta = 1.1$ (our method); and, for the cubic case, $\beta = 3.2$ (original MLPG1 method) and $\beta = 1.9$ (our method).

	Nodes	Value MLPG1 (m)	Error MLPG1	Value our (m)	Error our
quadratic	7	0.5144	0.04%	0.5146	0.08%
	13	0.5144	0.04%	0.5143	0.02%
	25	0.5143	0.02%	0.5142	0.00%
	49	0.5142	0.00%	0.5142	0.00%
cubic	7	0.5147	0.10%	0.5131	0.21%
	13	0.5145	0.06%	0.5139	0.06%
	25	0.5144	0.04%	0.5141	0.02%
	49	-0.4633	190.10%	0.5142	0.00%

As expected, those quantitative results confirm the qualitative results shown in Figures 10 and 11. In the quadratic case, all the relative errors of both methods are less than 0.1%. However, in the cubic case, our method presented a consistent diminishing of the relative error as the number of nodes increased. MLPG1 showed a similar behavior up to 25 nodes. However, with a 49-node discretization, the diminishing trend was lost, and the results are clearly wrong.

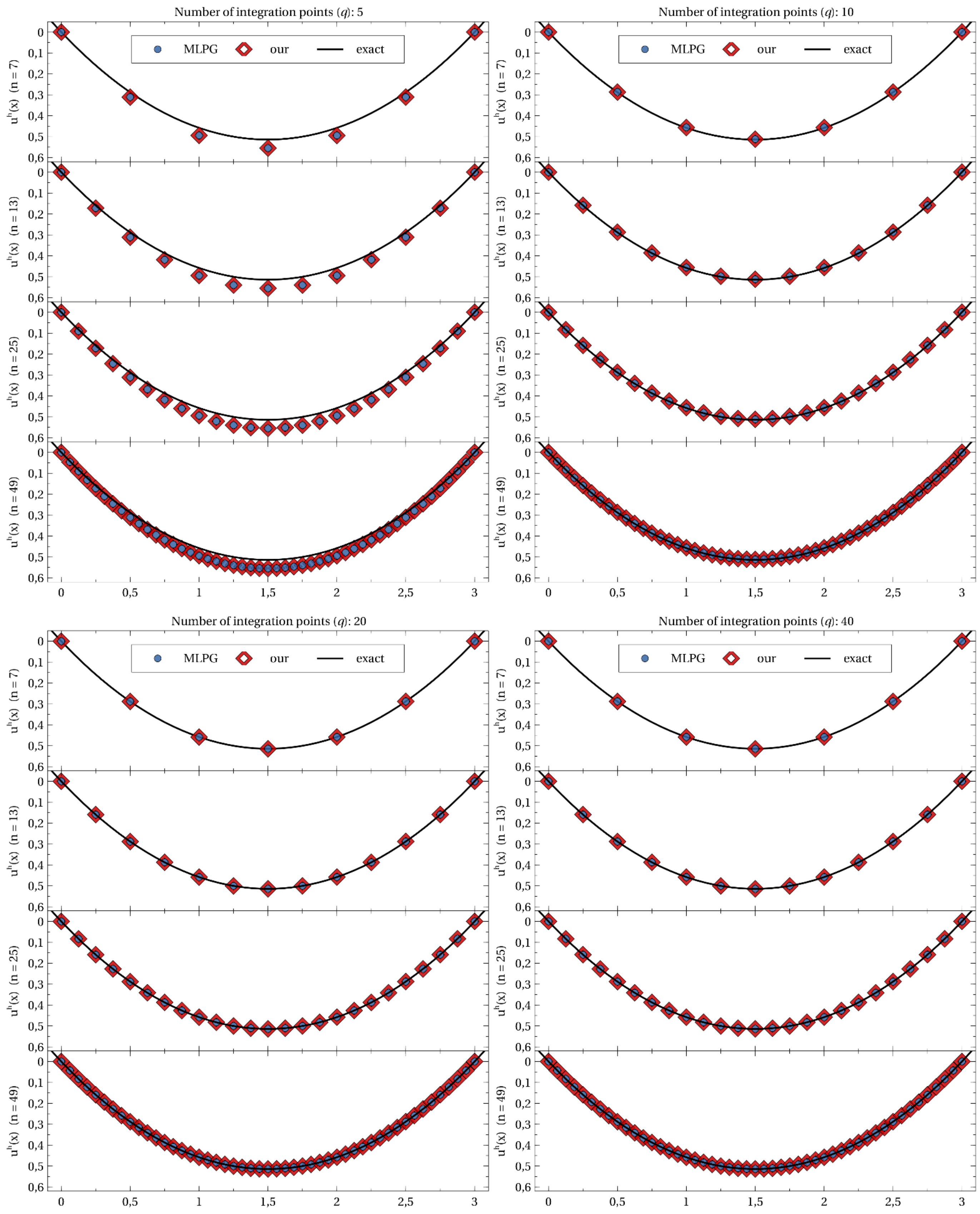


Figure 10: Comparison of the MLPG1 method with our approach for the quadratic polynomial order, where $\bar{x}$ is located in accordance with the pattern shown in Figure 6a. The displacements are shown at each node, using uniform discretization with 7, 13, 25 and 49 nodes; and 5, 10, 20 and 40 integration points. The support radii of the test functions were fixed at $\alpha = 0.7$, and the radii for the trial functions were: $\beta = 2.1$ for the original MLPG1 method and $\beta = 1.1$ for our method.

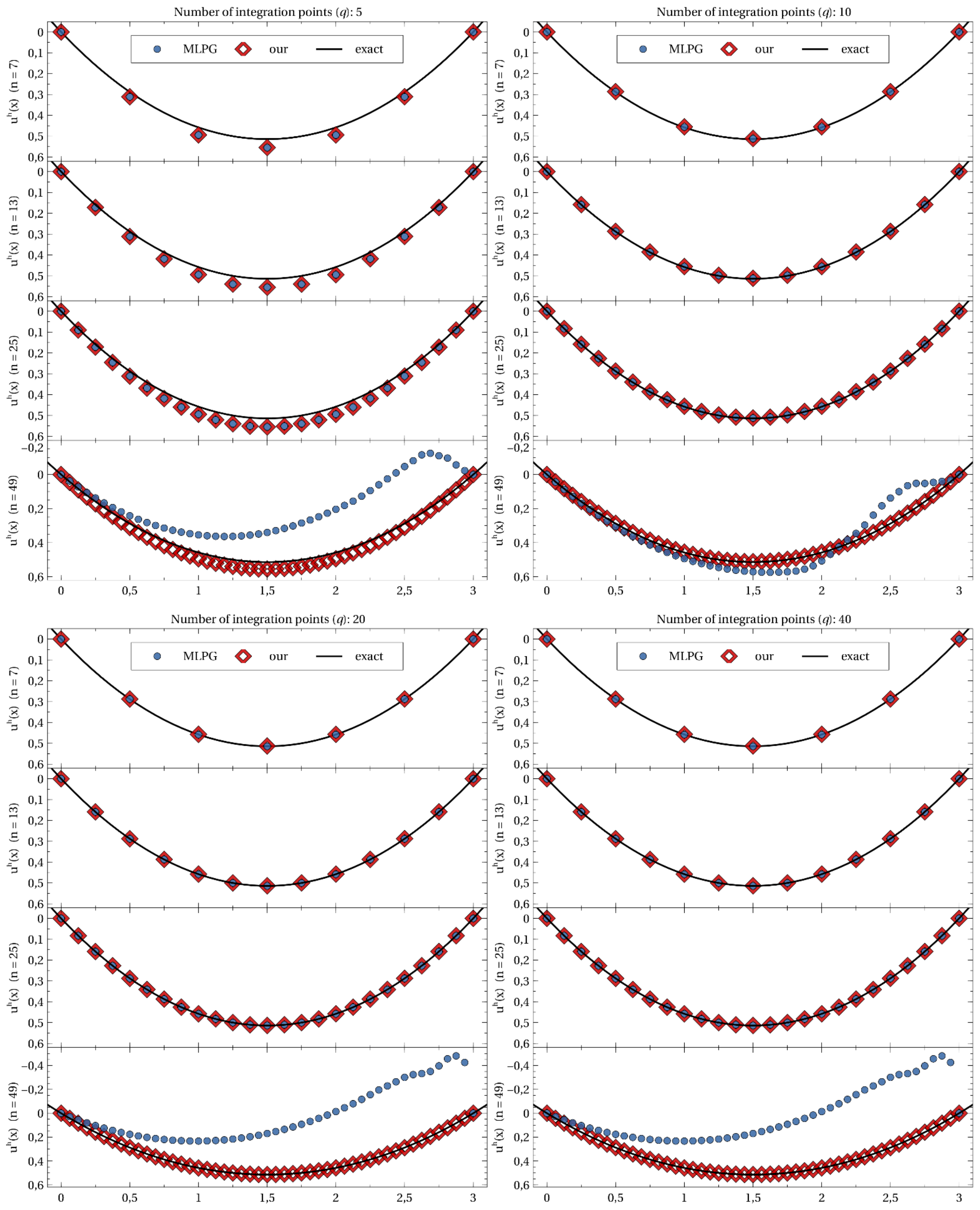


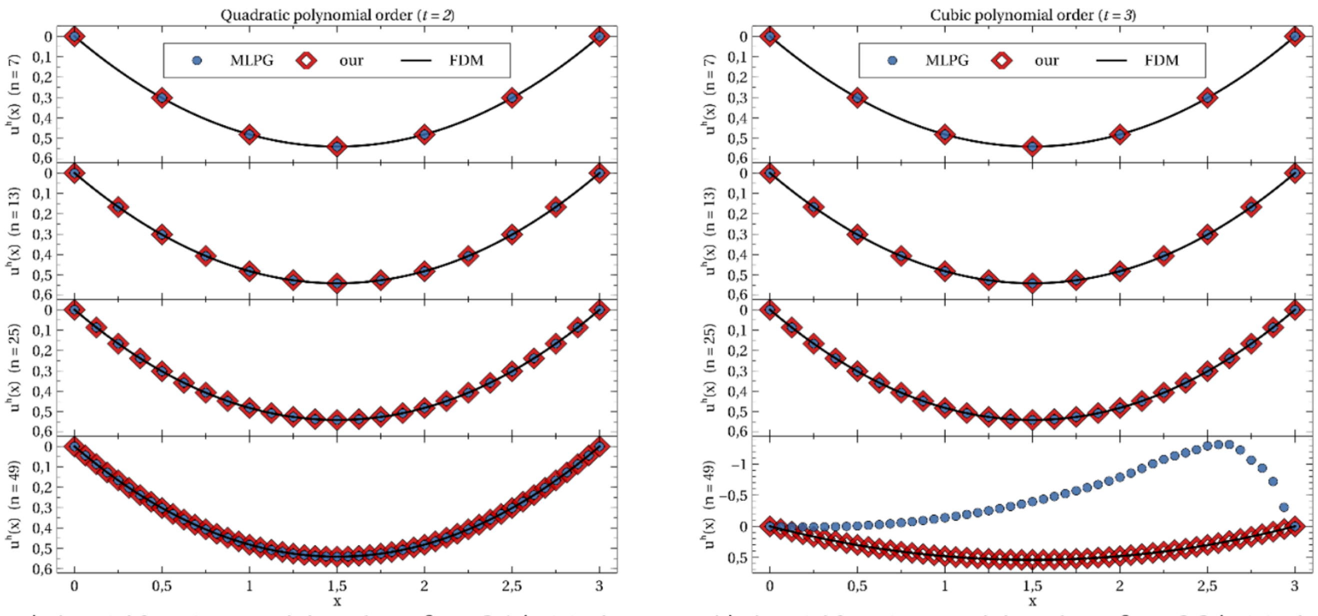
Figure 11: Comparison of the MLPG1 method with our approach for the cubic polynomial order. The displacements are shown at each node, using uniform discretization with 7, 13, 25 and 49 nodes; and 5, 10, 20 and 40 integration points. The support radii of the test functions were fixed at $\alpha = 0.7$, and the radii for the trial functions were: $\beta = 3.2$ for the original MLPG1 method and $\beta = 1.9$ for our method.

3.3 Varying t with Non-Constant $k(x)$

Since the analytical result shown in Equation (43) corresponds to the case in which $k(x)$ is constant, we cannot use it here because $k(x) = \sin(x\pi/L)$. Thus, we compare the results with a Finite Difference (FD) solution obtained with a very refined grid of 100 partitions. All other variables of the model are the same as the previous example.

The tests for this problem were exactly the same as those performed in Section 3.2. However, we present only the results with 40 integration points. Thus, Figures 12a and 12b show, respectively, the results for the quadratic and cubic cases ($t = 2$ and 3), using 40 integration points ($i = 40$) and discretizations of 7, 13, 25 and 49 nodes. Table 2 shows the midpoint's displacements and the corresponding relative errors. In this test, the original MLPG1 example, with cubic order trial functions and 49, nodes shows a deviation from the FDM's solution as in the previous test.

Both methods reach similar relative errors for the quadratic case. However, for the cubic case, our method always presents better results with consistent diminishing error trend as the discretization increases.



a) The trial functions used the values: $\beta = 2.1$ (original MLPG1 method) and $\beta = 1.1$ (our method).

b) The trial functions used the values: $\beta = 3.2$ (original MLPG1 method) and $\beta = 1.9$ (our method).

Figure 12: Tests using $k(x) = \sin(x\pi/L)$ for the quadratic (a) and cubic cases (b). The displacements are shown at each node, using uniform discretization with 7, 13, 25 and 49 nodes; and 40 integration points. The support radii of the test functions were fixed at $\alpha = 0.7$.

Table 2 Approximate displacements of the cable's midpoint. The "true" displacement of 0.5406m was computed using the finite difference method 100 domain partitions. All the examples use 40 integration points and $\alpha = 0.7$. The values of β are: for the quadratic cases, $\beta = 2.1$ (original MLPG1 method) and $\beta = 1.1$ (our method); and, for the cubic case, $\beta = 3.2$ (original MLPG1 method) and $\beta = 1.9$ (our method).

	Nodes	Value MLPG1 (m)	Error MLPG1	Value our (m)	Error our
quadratic	7	0. 5410	0.07%	0.5410	0.07%
	13	0.5409	0.06%	0.5407	0.02%
	25	0.5407	0.02%	0.5407	0.02%
	49	0.5406	0.00%	0.5406	0.00%
cubic	7	0.5414	0.15%	0.5402	0.08%
	13	0.5409	0.06%	0.5405	0.03%
	25	0.5408	0.04%	0.5406	0.00%
	49	-0.3952	173.10%	0.5407	0.02%

3.4 Computational Performances

This test compares the average time spent in computing the solution with the original MLPG1 method and with our method (using both $\bar{x}$ distributions illustrated in Figures 6a and 6b). The simulations were repeated ten thousand times and the average simulation time was computed. We implemented a program with both methods using C++ and the simulations were made in the same machine. The tests used 7, 13, 25 and 49 nodes with $t = 2$ (quadratic polynomial order), using 5 and 40 integration points.

The results plotted in Figure 13 indicate that our approach outperforms MLPG1. Considering, for example, the 40-integration-point cases, which deliver the most accurate results, our method is approximately 30 times faster. Even if we were satisfied with the results obtained with 5 integration points and 49 nodes, our method was approximately 5 times faster.

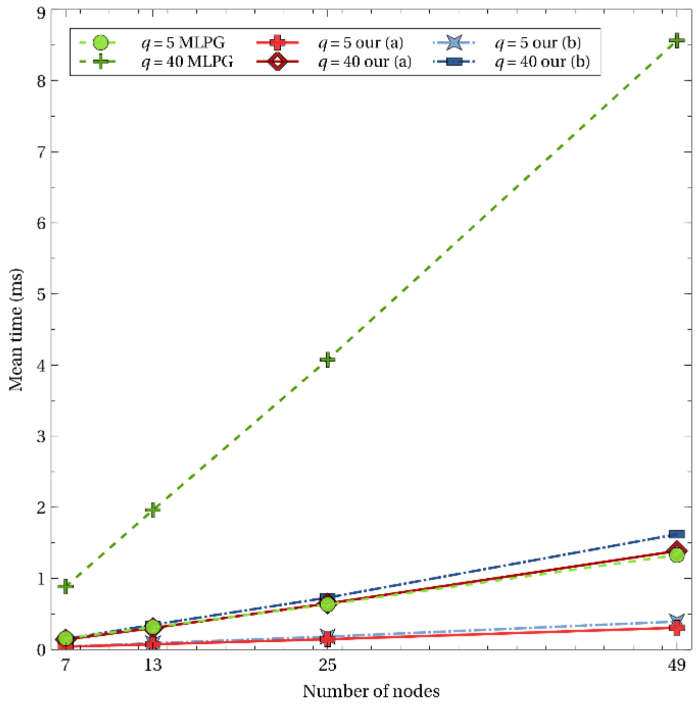


Figure 13: Average run time vs discretization. All simulations were performed for the quadratic polynomial order ($t = 2$), considering the two distribution patterns for $\bar{x}$ shown in Figures 6a and 6b. The discretization used 7, 13, 25 and 49 nodes, and the integration used 5 and 40 integration points. The values of α and β were fixed ($\alpha = 0.7$ and $\beta = 2.1$ for the the MLPG1 method and $\beta = 1.1$ for our method).

Also, notice that the performance of our method with 40 integration points is comparable to the performance of MLPG1 with 5 integration points. So, for equivalent time performance, our approach delivers more accurate results. As shown in Figure 7, for 5 integration points, regardless of the number of nodes, MLPG1 delivers a relative error of approximately 8%. However, with the same computational times for each discretization, our method delivers results with relative errors less than 0.01% (see Table 1).

4 CONCLUSIONS

We investigated the behavior of a one-dimensional problem of a cable with fixed ends using the meshless local Petrov-Galerkin method using the same weight function as the test function. That method is also known as MLPG1 (Atluri and Shen, 2002) and uses the Moving Least Square (MLS) to construct local trial functions. The MLS collocation method was used to enforce the displacement boundary conditions.

The problem was sufficiently simple for us to assess the pitfalls of the original MLPG formulation and to propose a consistent MLPG formulation that does not suffer of the same drawbacks. Our development shows that the location around which the approximating polynomial's coefficients are optimized should be dissociated from the place where the polynomial is evaluated, and its derivatives are computed. This is truly consistent with the error minimization associated with the MLS method, and makes the method simple and more powerful. We demonstrated, through a series of tests, that the consistent formulation delivers accurate results in an efficient manner. Moreover, with our method, it is easy to recommend sound values of α and β to deliver accurate results. In fact, although this is a very important issue for the practical application of the method, such recommendation is often overlooked in the literature.

As future work, the proposed approach will be extended to two-dimensional problems next and then, to three-dimensional ones.

Author's Contributions: Conceptualization, SMF Oliveira and LLC Sousa and CA Vidal; Data curation, SMF Oliveira; Formal Analysis, CA Vidal; Investigation, SMF Oliveira and LLC Sousa and CA Vidal and JB Cavalcante-Neto; Methodology, SMF Oliveira and LLC Sousa and CA Vidal; Software, SMF Oliveira and LLC Sousa; Supervision, CA Vidal and JB Cavalcante-Neto; Visualization, SMF Oliveira; Writing – original draft, SMF Oliveira and LLC Sousa and CA Vidal; Writing – review & editing, CA Vidal and JB Cavalcante-Neto.

Editor: Marco L. Bittencourt.

References

Amini, R., Akbarmakoui, M. and Nezhad, S. M. M. (2018), Fluid flow modeling in channel using meshless local petrov-galerkin (mlpg) method by radial basis function, Modares Mechanical Engineering 18(8), 241-249.

Atluri, S. N. and Shen, S. (2002), The Meshless Local Petrov-Galerkin (MLPG) Method: A Simple and Less-costly Alternative to the Finite Element and Boundary Element Methods, CMES-Computer Modeling in Engineering & Sciences, 11-51.

Atluri, S. N. and Zhu, T. (1998), A New Meshless Local Petrov-Galerkin (MLPG) Approach in Computational Mechanics, Computational Mechanics 22(2), 117-127.

Atluri, S. N. and Zhu, T. (2000), The meshless local petrov-galerkin (mlpg) approach for solving problems in elasto-statics, Computational Mechanics 25, 169-179.

Atluri, S. N., Cho, J. Y. and Kim, H.-G. (1999a), Analysis of thin beams, using the meshless local petrov-galerkin method, with generalized moving least squares interpolations, Computational Mechanics 24, 334-347.

Atluri, S. N., Cho, J. Y. and Kim, H.-G. (1999b), A critical assessment of the truly meshless local petrov-galerkin (mlpg), and local boundary integral equation (lbie) methods, Computational Mechanics 24, 348-372.

Babuska, I. and Melenk, J. M. (1997), The Partition of Unity Method, International Journal for Numerical Methods in Engineering 40(4), 727-758.

Belytschko, T., Lu, Y. Y. and Gu, L. (1994), Element-Free Galerkin Methods, International Journal for Numerical Methods in Engineering 37(2), 229-256.

Buchanan, G. (1994), Schaum's Outline of Finite Element Analysis, Schaum's Outline Series, McGraw-Hill Education.

Gingold, R. A. and Monaghan, J. J. (1977), Smoothed Particle Hydrodynamics: Theory and Application to Non-Spherical Stars, Mon. Not. Roy. Astron. Soc. 181, 375-389.

Guennebaud, G., Jacob, B. et al. (2019), Eigen v3.3.7, http://eigen.tuxfamily.org.

Han, Z. D. and Atluri, S. N. (2004), Meshless Local Petrov-Galerkin (MLPG) Approaches for Solving 3D Problems in Elasto-Statics, CMES-Computer Modeling in Engineering & Sciences 6, 129-140.

Han, Z., Rajendran, A. and Atluri, S. N. (2005), Meshless local petrov-galerkin (mlpg) approaches for solving nonlinear problems with large deformations and rotations, CMES-Computer Modeling in Engineering & Sciences 10, 1-12.

Hu, D. and Sun, Z. (2011), The meshless local petrov-galerkin method for large deformation analysis of hyperelastic materials, ISRN Mechanical Engineering 2011.

Kamranian, M., Dehghan, M. and Tatari, M. (2017), An adaptive meshless local petrov-galerkin method based on a posteriori error estimation for the boundary layer problems, Applied Numerical Mathematics111, 181-196.

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

Liu, N., He, X., Li, S. and Wang, G. (2011), Meshless Simulation of Brittle Fracture, Computer Animation and Virtual Worlds 22(2-3), 115-124.

Liu, W. K., Jun, S. and Zhang, Y. F. (1995), Reproducing Kernel Particle Methods, International Journal for Numerical Methods in Fluids 20(8-9), 1081-1106.

Mirzaei, D. (2015), A new low-cost meshfree method for two and three dimensional problems in elasticity, Applied Mathematical Modelling 39(23), 7181-7196.

Mirzaei, D. and Schaback, R. (2013), Direct meshless local petrov-galerkin (DMLPG) method: A generalized MLS approximation, Applied Numerical Mathematics 68, 73-82.

Sheikhi, N., Najafi, M. and Enjilela, V. (2019), Extending the meshless local petrov-galerkin method to solve stabilized turbulent fluid flow problems, International Journal of Computational Methods 16(01).

Sladek, J., Sladek, V., Solek, P. and Zhang, C. (2010), Fracture analysis in continuously nonhomogeneous magneto-electro-elastic solids under a thermal load by the MLPG, International Journal of Solids and Structures 47(10), 1381-1391.

Sladek, J., Stanak, P., Han, Z., Sladek, V. and Atluri, S. N. (2013), Applications of the MLPG Method in Engineering & Sciences: A Review, CMES: Computer Modeling in Engineering & Sciences 92(5), 423-475.

Sukumar, N., Moran, B. and Belytschko, T. (1998), The Natural Element Method in Solid Mechanics, International Journal for Numerical Methods in Engineering 43(5), 839-887.

Zhang, X., Yao, Z. and Zhang, Z. (2006), Application of MLPG in Large Deformation Analysis, Acta Mechanica Sinica22(4), 331-340.