Department of Computing

Federal University of Ceará

DC

A One-Dimensional Least-Square-Consistent Displacement-based Meshless Local Petrov-Galerkin Method

by

Laise Lima de Carvalho Sousa,

Suzana Matos França de Oliveira,

Creto Augusto Vidal,
and Joaquim Bento Cavalcante Neto

2019

A One-Dimensional Least-Square-Consistent Displacement-based Meshless Local Petrov-Galerkin Method

Laise Lima de Carvalho Sousa,
Suzana Matos França de Oliveira,
Creto Augusto Vidal,
and Joaquim Bento Cavalcante Neto

2019

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.

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 ([13], [23]). 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) [7]; element free Galerkin (EFG) [5]; reproducing kernel particle method (RKPM) [15]; partition of unity finite element method (PUFEM) [4]; natural element method (NEM) [24]. However, as Atluri and Zhu [3] 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 [3] 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,

2

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 ([1], [9], [11], [12], [14], [21], [25]). However, the shape functions used in this method have very complex forms, which makes their computation and their derivatives' computation quite costly ([13], [17]). 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 II, 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 III, we present some tests. Finally, in Section IV, we present our conclusions.

2 Least-Square-Consistent MLPG Formulation (LSC-MLPG)

In this section, we explain the LSC-MLPG formulation (see the overview shown in Figure 1), focusing on how we propose to compute a shape function's derivative. To achieve this goal, we consider a two-point boundary value problem.
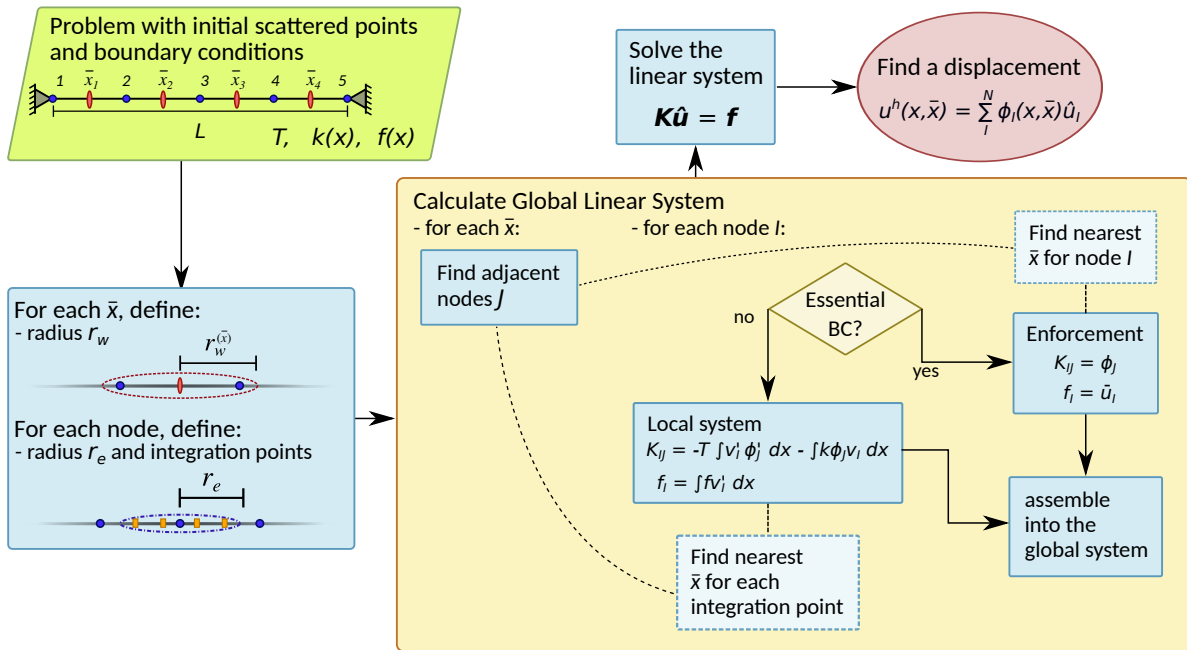

Figure 1: Overview of the technique.

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. 2).
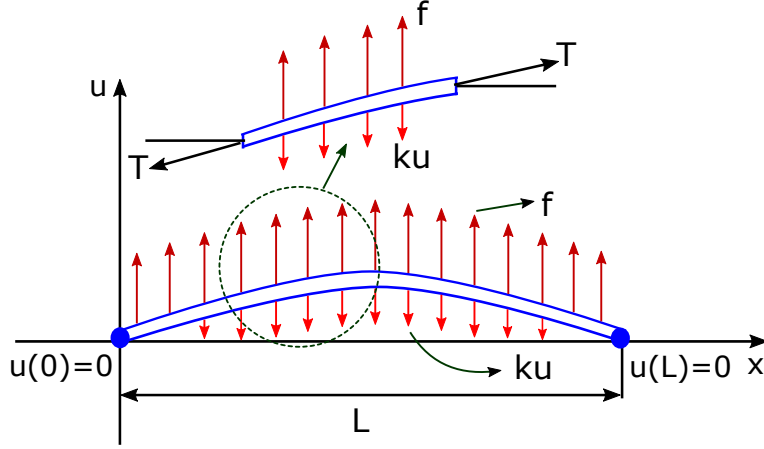

Figure 2: Cable with Dirichlet boundary conditions in the interval $[0, L]$, deflecting upward in the positive y direction. $u(x)$ is the transverse displacement of the cable (deflection), $f(x)$ is the external transverse load, $k(x)$ is the stiffness offered by the medium in which the cable is embedded; 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. 3). 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 [3].
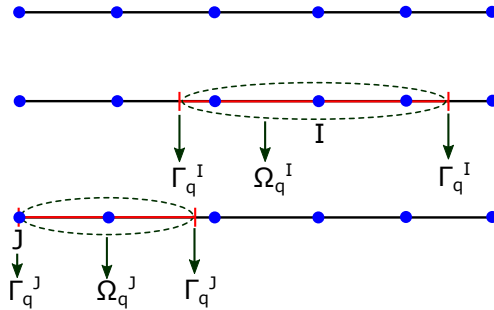

Figure 3: The domain of the cable is covered by nodes scattered arbitrarily. All nodes will have a local subdomain Ω_q with its local boundary Γ_q. Ω_q^I and Ω_q^J represent, in that order, the local subdomains of nodes I and J, and their boundaries are Γ_q^I and Γ_q^J.

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

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

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 [2], [3], [10], [19], [18].

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

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

where $\mathbf{p}^T(x) = [p_1(x) \quad p_2(x) \ldots p_m(x)]$ is a complete monomial basis and $\mathbf{a}(\bar{x}) = [a_1(\bar{x}) \quad a_2(\bar{x}) \ldots a_m(\bar{x})]^T$ 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 [2]. For example, if $t = 2$, then $m = 3$ and $\mathbf{p}^T(x) = [1 \quad x \quad x^2]$.

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. 4). 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

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

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

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

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. 4). 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

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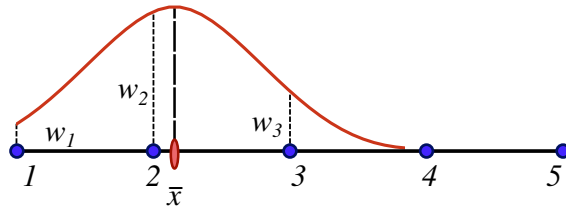

Figure 4: Influences of nodes in the vicinity of $\bar{x}$: nodes 1, 2 and 3. Node 2 has the strongest influence on $\bar{x}$ and node 1 has the least influence on $\bar{x}$.

In order to find the coefficients $\mathbf{a}(\bar{x})$ of the approximating polynomial, we minimize the squared L_2 norm of the vector of weighted errors (Notice that, in [3], the weighted squared errors are minimized while we minimize the squared weighted errors). Thus, defining the functional

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

we pose the following unconstrained minimization problem:

$$\underset{\mathbf{a}(\bar{x}))}{\text{minimize}} \ \mathbf{J}(\mathbf{a}(\bar{x})). \tag{14}$$

After solving the unconstrained minimization problem, the optimized coefficients are

$$\mathbf{a}(\bar{x}) = \mathbf{A}^{-1}(\bar{x})\mathbf{B}(\bar{x})\hat{\mathbf{u}}, \tag{15}$$

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

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}^T(x_1) \\ \mathbf{p}^T(x_2) \\ \vdots \\ \mathbf{p}^T(x_N) \end{bmatrix}, \tag{16}$$

$$\mathbf{W}(\bar{x}) = \begin{bmatrix} w_1^2 & 0 & \cdots & 0 \\ 0 & w_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_N^2 \end{bmatrix}, \tag{17}$$

and

$$\hat{\mathbf{u}} = [\hat{u}_1 \quad \hat{u}_2 \quad \cdots \quad \hat{u}_N]^T. \tag{18}$$

2.3.1 Shape Functions

Substituting Equation (15) into Equation (6) yields

$$u^h(x, \bar{x}) = \mathbf{p}^T(x)\mathbf{A}^{-1}(\bar{x})\mathbf{B}(\bar{x})\hat{\mathbf{u}} = \boldsymbol{\phi}^T(x, \bar{x})\hat{\mathbf{u}}, \tag{19}$$

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

$$\begin{aligned} \boldsymbol{\phi}^T(x, \bar{x}) &= \mathbf{p}^T(x)\mathbf{A}^{-1}(\bar{x})\mathbf{B}(\bar{x}) = \\ &= [\phi_1(x, \bar{x}) \quad \phi_2(x, \bar{x}) \quad \cdots \quad \phi_N(x, \bar{x})], \end{aligned} \tag{20}$$

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

Considering Equation (20), Equation (19) can be rewritten as

$$u^h(x, \bar{x}) = \sum_{I=1}^{N} \phi_I(x, \bar{x})\hat{u}_I, \tag{21}$$

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

$$\phi_I(x, \bar{x}) = \sum_{j=1}^{m} p_j(x)[\mathbf{A}^{-1}(\bar{x})\mathbf{B}(\bar{x})]_{jI}, \tag{22}$$

where $p_j(x)$ is the jth term of the polynomial basis $\mathbf{p}(x)$ and $[\mathbf{A}^{-1}(\bar{x})\mathbf{B}(\bar{x})]_{jI}$ is the element (j, I) of matrix $\mathbf{A}^{-1}(\bar{x})\mathbf{B}(\bar{x})$. Note that, in order to compute the shape functions, the $\mathbf{A}$ 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 [2] and [13]).

2.3.2 Derivatives of the Shape Functions

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

$$\phi_I'(x, \bar{x}) = \sum_{j=1}^{m} p_j'(x)[\mathbf{A}^{-1}(\bar{x})\mathbf{B}(\bar{x})]_{jI}, \tag{23}$$

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 (23). This is totally consistent with the formulation developed for the approximating polynomial and its coefficients (Eqs.(6) to (21)). On the other hand, to substitute x for $\bar{x}$ in Equation (15) makes the formulation inconsistent.

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 forth order spline with compact support as weight function ([2, 20, 22]), i.e.,

$$w(r) = \begin{cases} 1 - 6\left(\dfrac{r}{r_s}\right)^2 + 8\left(\dfrac{r}{r_s}\right)^3 - 3\left(\dfrac{r}{r_s}\right)^4, & 0 \leq r \leq r_s \\ 0, & r > r_s \end{cases} \tag{24}$$

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 $r_w^{(\bar{x})}$ should be sufficiently large, as mentioned in section 2.3.1, to contain the N required points to ensure that matrix A is invertible ($N \geq m$). However, $r_w^{(\bar{x})}$ should also be small enough to maintain the local nature of the MLS approach [2].

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 [2]. In this work, the test functions are also bell-shaped functions centered at the nodes (the same as the weight function of Equation (24) with $r = ||x - x_I||$ and $r_s = r_e^{(I)}$), resulting in the so-called MLPG1. However, $r_w^{(\bar{x})}$ need not be equal to r_e^I (Fig. 5). 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.
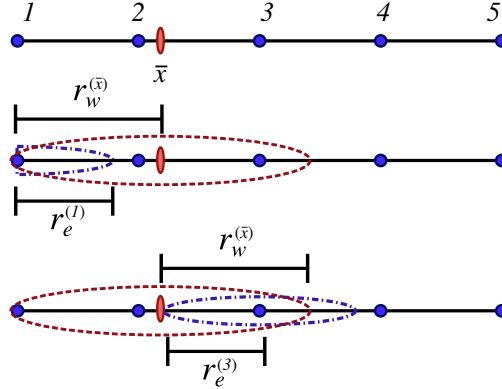

Figure 5: Support zones for the test functions, r_e^I, and for the trial functions, $r_w^{(\bar{x})}$: r_e^I of nodes 1 and 3 and $r_w^{(\bar{x})}$ centered at $\bar{x}$).

2.5 Discretization

Substituting equation (21) into equation (5) the local weak form for a node I can be rewritten as

$$\sum_{J=1}^{N} \left\{ T\left[v^I(x_r)\phi'_J(x_r, \bar{x})\hat{u}_J - v^I(x_l)\phi'_J(x_l, \bar{x})\hat{u}_J - \int_{\Omega_q^I} v'^I(x)\phi'_J(x, \bar{x})\hat{u}_J dx \right] \right.$$
$$\left. - \int_{\Omega_q^I} k(x)\phi_J(x, \bar{x})\hat{u}_J v^I(x)dx \right\} + \int_{\Omega_q^I} f(x)v^I(x)dx = 0. \tag{25}$$

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

$$\sum_{J=1}^{N} K_{IJ}\hat{u}_J = f_I, \tag{26}$$

where

$$K_{IJ} = T\left[v^I(x_r)\phi'_J(x_r,\bar{x}) - v^I(x_l)\phi'_J(x_l,\bar{x}) - \int_{\Omega_q^I} v'^I(x)\phi'_J(x,\bar{x})dx\right] - \int_{\Omega_q^I} k(x)\phi_J(x,\bar{x})v^I(x)dx \tag{27}$$

and

$$f_I = -\int_{\Omega_q^I} f(x)v^I(x)dx. \tag{28}$$

The test function $v^I(x)$ is chosen to vanish in the support boundary, then equation (27) can be simplified. Therefore, for internal nodes (Fig. 5, nodes 2 to 4), we have

$$K_{IJ} = -T\int_{\Omega_q^I} v'^I(x)\phi'_J(x,\bar{x})dx - \int_{\Omega_q^I} k(x)\phi_J(x,\bar{x})v^I(x)dx; \tag{29}$$

for the left node $x_l = 0$ (Fig. 5, node 1), then

$$K_{IJ} = T\left[-\phi'_J(0,\bar{x}) - \int_{\Omega_q^I} v'^I(x)\phi'_J(x,\bar{x})dx\right] - \int_{\Omega_q^I} k(x)\phi_J(x,\bar{x})v^I(x)dx; \tag{30}$$

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

$$K_{IJ} = T\left[\phi'_J(L,\bar{x}) - \int_{\Omega_q^I} v'^I(x)\phi'_J(x,\bar{x})dx\right] - \int_{\Omega_q^I} k(x)\phi_J(x,\bar{x})v^I(x)dx. \tag{31}$$

Applying Equation (26) 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

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

After solving this system, the displacement at any point in the domain can be obtained through Equation (21), 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 (equations (29) and (28)), making the necessary parameterization,

$$K_{IJ} = -r_e\sum_{q=1}^{LP}\omega_q\left[Tv'^I(x(\varepsilon_q))\phi'_J(x(\varepsilon_q),\bar{x}) + k(x(\varepsilon_q))\phi_J(x(\varepsilon_q),\bar{x})v^I(x(\varepsilon_q))\right] \tag{32}$$

and

$$f_I = -r_e\sum_{q=1}^{LP}\omega_q f(x(\varepsilon_q))v^I(x(\varepsilon_q)), \tag{33}$$

where ε_q, $q = 1,..,LP$, are the points of Legendre, ω_q are their associated weights, and $x(\varepsilon_q) = x_I + r_e\varepsilon_q$; for the left node (equations (30) and (28)),

$$K_{IJ} = -\frac{r_e}{2} \sum_{q=1}^{LP} \omega_q \left[T v'^{I}(x(\varepsilon_q)) \phi'_J(x(\varepsilon_q), \bar{x}) + k(x(\varepsilon_q)) \phi_J(x(\varepsilon_q), \bar{x}) v^I(x(\varepsilon_q)) \right] - T\phi'_J(0, \bar{x}) \tag{34}$$

and

$$f_I = -\frac{r_e}{2} \sum_{q=1}^{LP} \omega_q f(x(\varepsilon_q)) v^I(x(\varepsilon_q)), \tag{35}$$

where $x(\varepsilon_q) = \frac{r_e}{2}(1 + \varepsilon_q)$; and for the right node (equations (31) and (28)),

$$K_{IJ} = -\frac{r_e}{2} \sum_{q=1}^{L.P} \omega_q \left[T v'^{I}(x(\varepsilon_q)) \phi'_J(x(\varepsilon_q), \bar{x}) + k(x(\varepsilon_q)) \phi_J(x(\varepsilon_q), \bar{x}) v^I(x(\varepsilon_q)) \right] + T\phi'_J(L, \bar{x}), \tag{36}$$

where f_I is computed using equation (35), 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 that 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. 6). 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 [13] and [16].

$$u^h(0) = \sum_{J=1}^{N} \phi_J(0, \bar{x}) \hat{u}_J = 0 \tag{37}$$

and

$$u^h(L) = \sum_{J=1}^{N} \phi_J(L, \bar{x}) \hat{u}_J = 0. \tag{38}$$

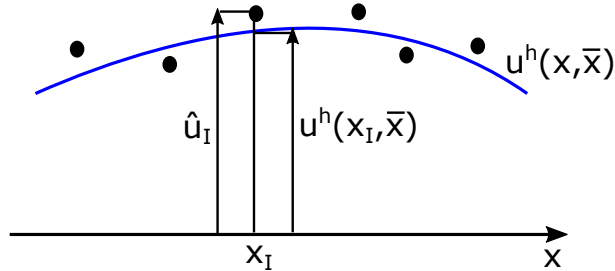Equations (37) and (38) replace equations (34), (35) and (36) for the left and right nodes.

Figure 6: Distinction between $\hat{u}_I$ and $u^h(x_I, \bar{x})$ in the MLS approximation.

2.8 Algorithm

The implementation of the Least-Square-Consistent Meshless Local Petrov-Galerkin (LSC-MLPG) Method followed Algorithm 1.

Algorithm 1: (LSC-MLPG)

Choose a finite number of nodes in the domain Ω and on the boundary Γ.

Choose a finite number of of $\bar{x}$ in the domain Ω.

Define the monomial basis and the radii r_e^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 $A^{-1}B$ (Eq. (15)).

Loop over all nodes I without prescribed displacement:

    Loop over all integration points q:

        Calculate local force vector f_I (Eq. (33)).
        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. (32)).
        Assemble local contributions into the global linear system $\mathbf{K}$ and $\mathbf{f}$.

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. (37) and (38)).

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

Calculate the displacement at any point x using Equation (21)

---

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 [8] 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 [3], [2], using a simple problem of a cable with fixed ends illustrated in Figure 2. 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 (39). 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 2, 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 [6]

$$u(x) = \frac{f[\cosh(\gamma L) - 1]\sinh(\gamma x)}{k\sinh(\gamma L)} - \frac{f\cosh(\gamma x)}{k} + \frac{f}{k}, \tag{39}$$

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

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

and
$$r_w(\beta) = \beta d_{min},$$
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 presented in Figures 10 and 12 are computed as

$$error = \frac{||\mathbf{u} - \mathbf{u}^h||}{||\mathbf{u}||} \times 100 \tag{40}$$

where $\mathbf{u}$ and $\mathbf{u}^h$ 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 the linear, quadratic and the cubic order trial functions.

3.1.1 Trial functions of first degree $(t = 1)$

In our method, $\bar{x}$ is located as illustrated in Figure 7 for the 7-node case, using a radius that ensures an inverse for the $\mathbf{A}$ matrix ($N \geq 2$). First, we consider the minimum number of trial functions, i.e., with $\bar{x}$ located at every other node (Fig. 7b). Then, we tested trial functions with $\bar{x}$ located at each internal node (Fig. 7a) 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 Figure 7.
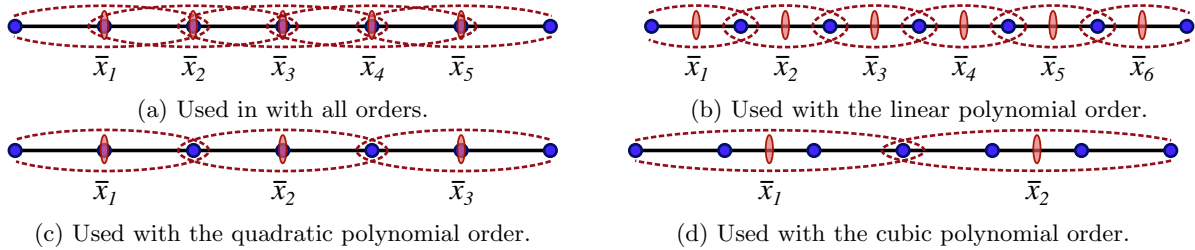

(a) Used in with all orders.

(b) Used with the linear polynomial order.

(c) Used with the quadratic polynomial order.

(d) Used with the cubic polynomial order.

Figure 7: Location of $\bar{x}$ for the 7-node case.

In the original MLPG1, $\bar{x}$ is fused with x, requiring a new computation of the shape functions for every integration point (see [3] and [20]).

First, we tested $\bar{x}$ located as shown in Figure 7b. The MLPG1 with linear order can obtain less error than our method as shown in Figures 8. However, due the instabilities of MLPG1, it's impossible to determine easily the parameters α and β that can be used to obtain a good approximated result. Our method remains with the stability as presented in the previous examples, making easy to choose the α and β parameters, although the error is high. The number of integration points do not influence the result as was obtained in the next examples (quadratic and cubic order).

However, the second test (Fig. 9), using Figure 7a to place $\bar{x}$, presented results that could not be used since the error was extremely high.

3.1.2 Trial functions of second degree ($t = 2$)

The quadratic order is tested with $\bar{x}$ positioned as illustrated in Figures 7c and 7a for the 7-node case. To ensure the inverse o matrix $\mathbf{A}$, we need $N \geq 3$. The distribution of trial functions for the 13, 25 and 49-node cases follows the same pattern shown in Figure 7. In the original MLPG1, $\bar{x}$ is fused with x, requiring a new computation of the shape functions for every integration point.

The results plotted in Figures 10 and 11 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 $\mathbf{A}$. 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 10 and 11: 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 $\mathbf{A}$ 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 an 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.

3.1.3 Trial functions of third degree ($t = 3$)

The the trial functions of third degree were tested with $\bar{x}$ is located as illustrated in Figures 7d and 7a for the 7-node case, ensuring that matrix $\mathbf{A}$ 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 7. 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 first test placed $\bar{x}$ as in Figure 7d. The results plotted in Figure 12 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 $\mathbf{A}$. 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 12: 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 $\mathbf{A}$ to have an inverse, while our method requires $\beta = 1.6$. With 49 nodes, MLPG1 requires β to be at least 3.2 for $\mathbf{A}$ 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.

The second test, using $\bar{x}$ as shown in Figure 7a, presented similar results as the previous test (Fig. 7d), particularly at the initial range of β values. Furthermore, we can observe that the larger the number of nodes and integration points are, the more stable the result becomes.
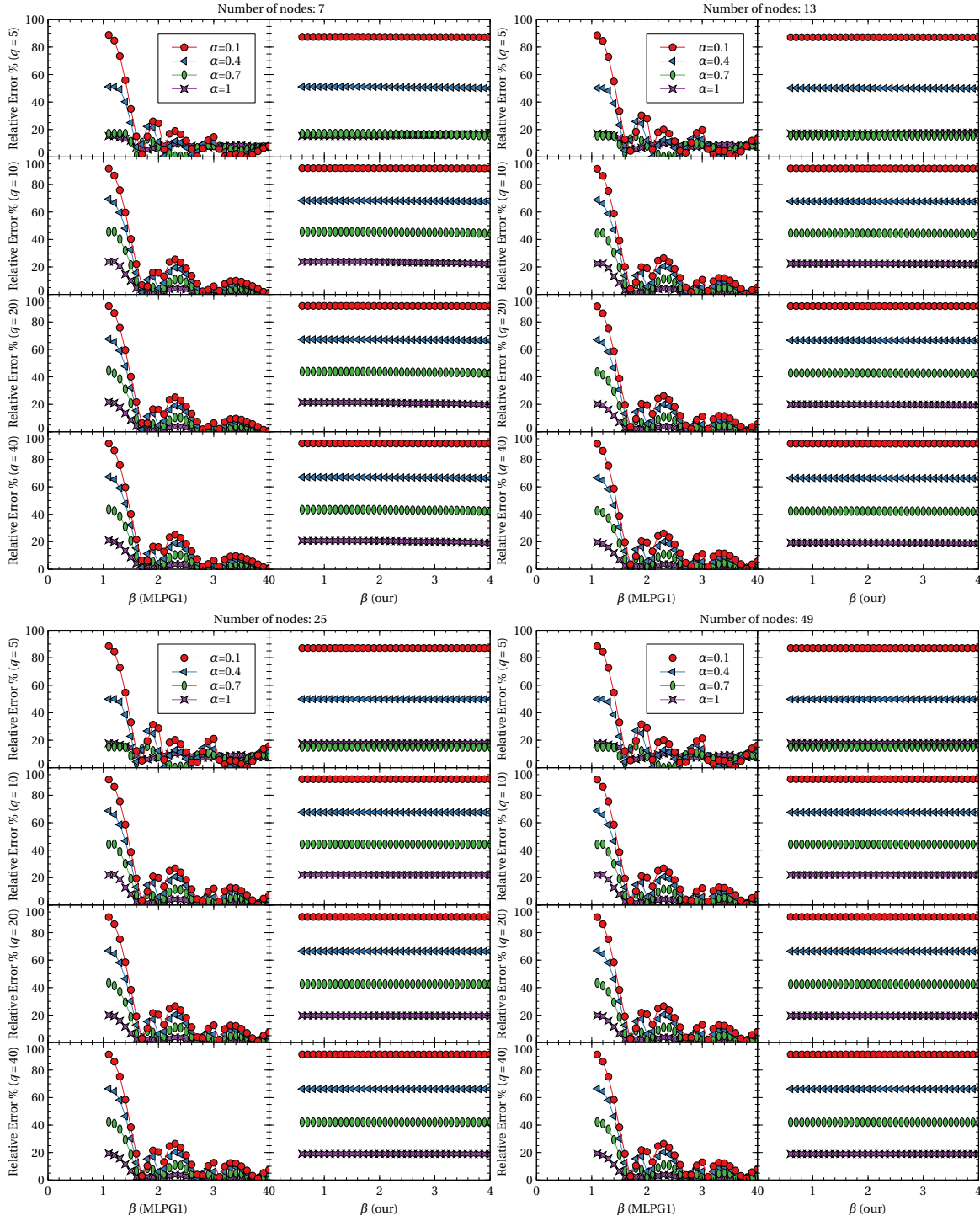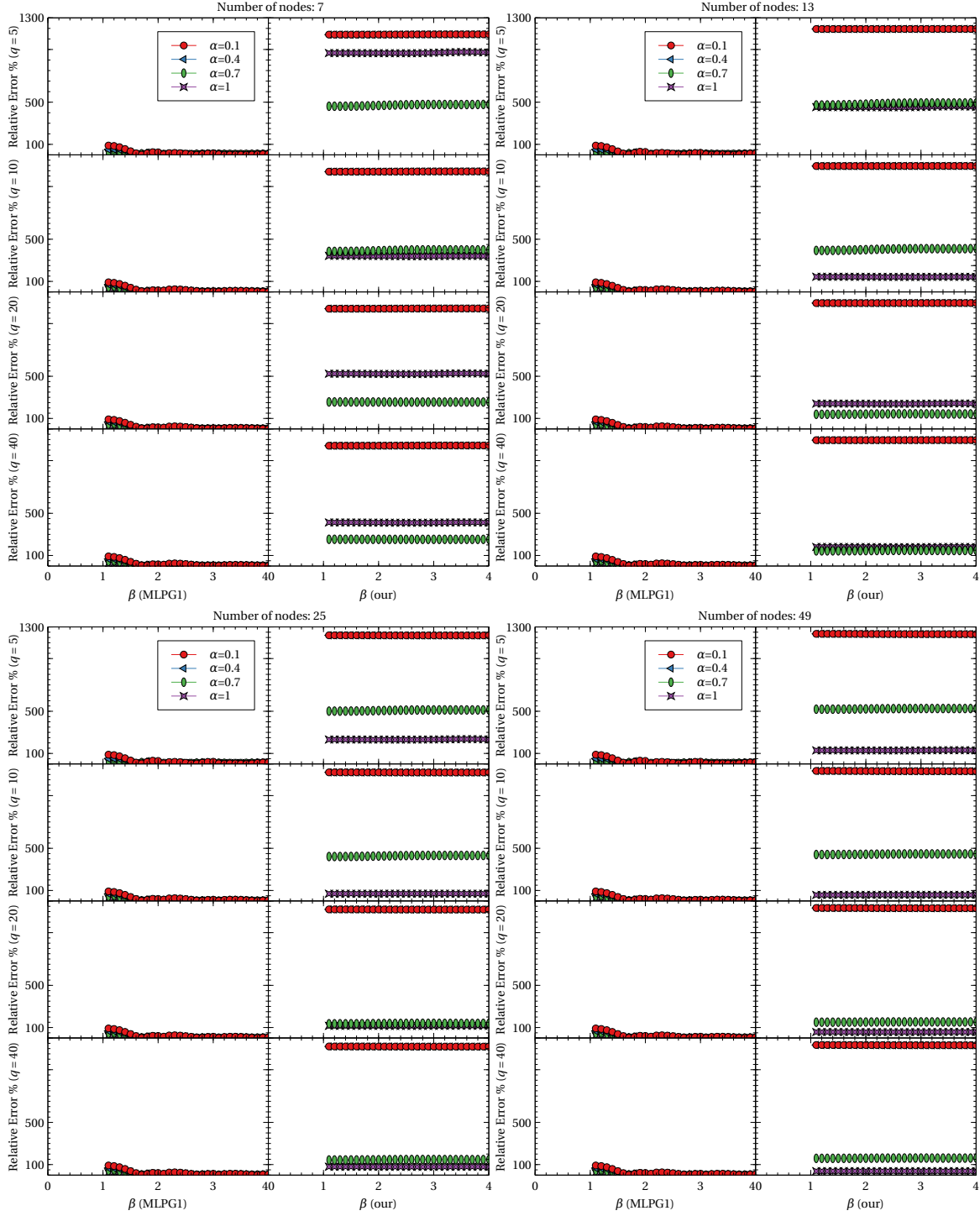
Figure 8: Exhaustive test with linear polynomial base using 7, 13, 25 and 49 nodes, where $\bar{x}$ is located in accordance with the pattern shown in Figure 7b. The graphs on the left columns are from the original MLPG1 method and on the right columns are from our approach. The tests are performed with 5, 10, 20 and 40 integration points as shown in each row. The y-axis represents the percentage relative error (Eq. (40)), the x-axis shows the values of β, and each curve corresponds to a specific value of α.

Figure 9: Exhaustive test with linear polynomial base using 7, 13, 25 and 49 nodes, where $\bar{x}$ is located in accordance with the pattern shown in Figure 7a. The graphs on the left columns are from the original MLPG1 method and on the right columns are from our approach. The test are performed with 5, 10, 20 and 40 integration points as shown in each row. The y-axis represents the percentage relative error (Eq. (40)), the x-axis shows the values of β, and each curve corresponds to a specific value of α.
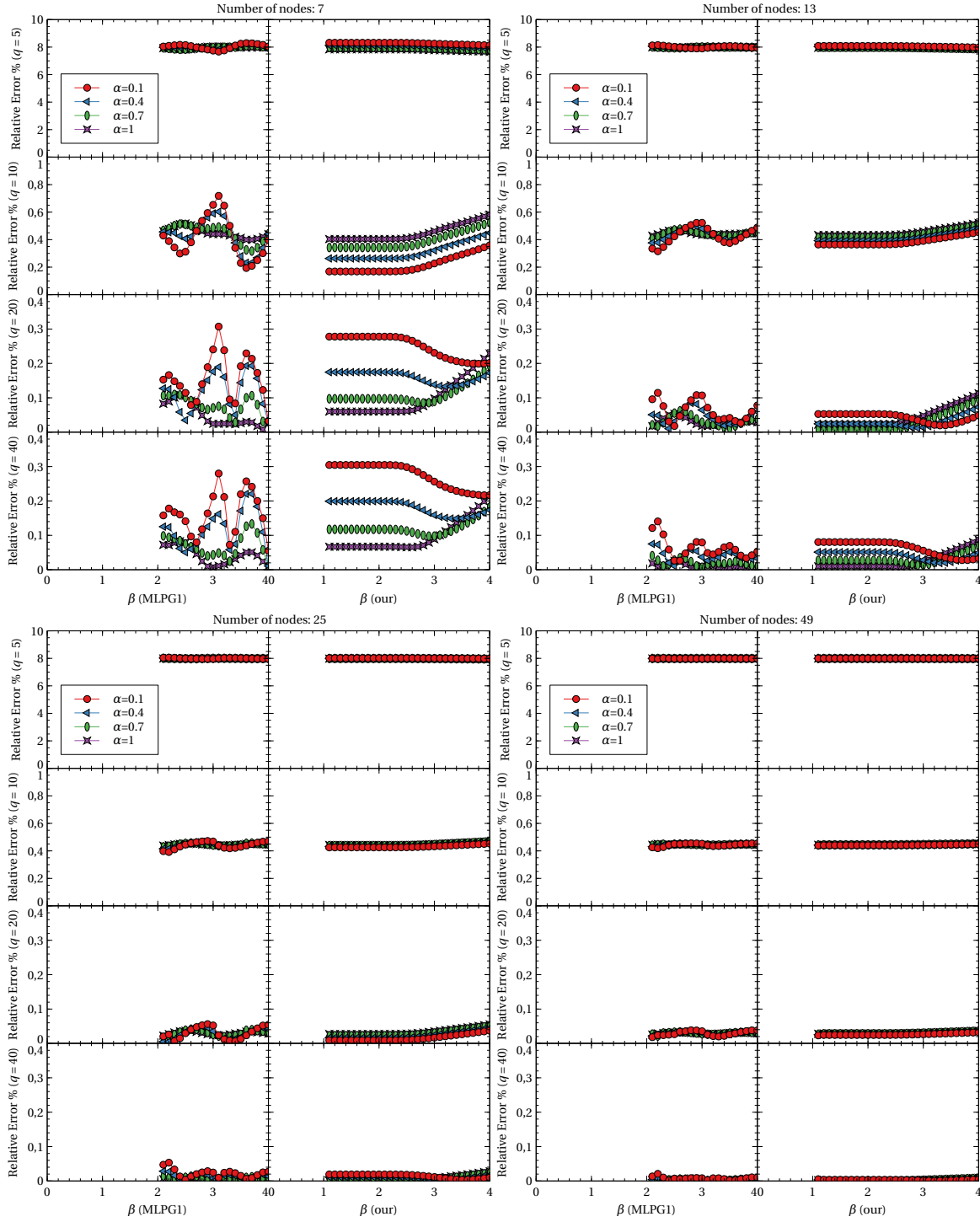
Figure 10: Exhaustive test with quadratic polynomial base using 7, 13, 25 and 49 nodes, where $\bar{x}$ is located in accordance with the pattern shown in Figure 7c. The graphs on the left columns are from the original MLPG1 method and on the right columns are from our approach. The tests are performed with 5, 10, 20 and 40 integration points as shown in each row. The y-axis represents the percentage relative error (Eq. (40)), the x-axis shows the values of β, and each curve corresponds to a specific value of α.
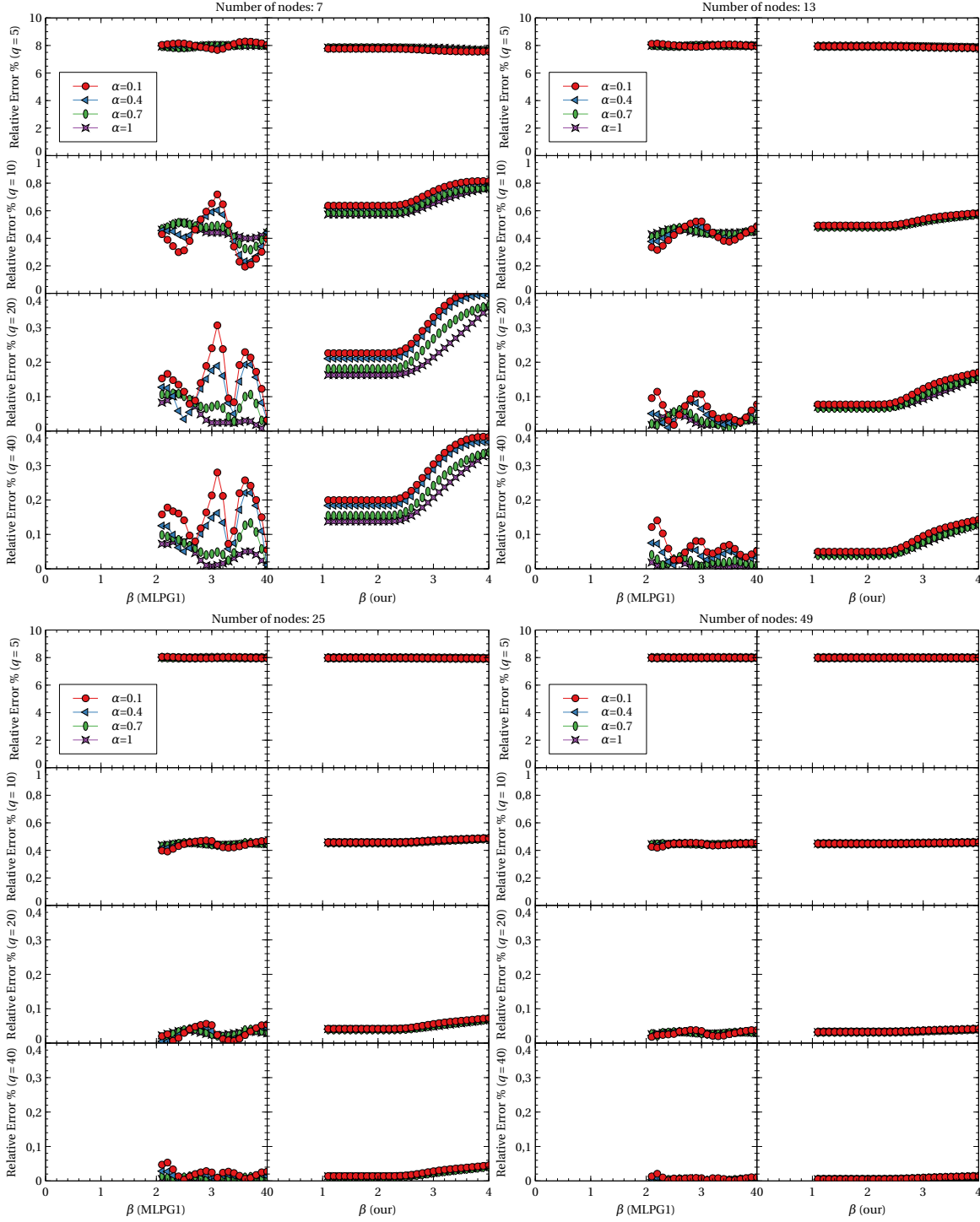
Figure 11: Exhaustive test with quadratic polynomial base using 7, 13, 25 and 49 nodes, where $\bar{x}$ is located in accordance with the pattern shown in Figure 7a. The graphs on the left columns are from the original MLPG1 method and on the right columns are from our approach. The test are performed with 5, 10, 20 and 40 integration points as shown in each row. The y-axis represents the percentage relative error (Eq. (40)), the x-axis shows the values of β, and each curve corresponds to a specific value of α.
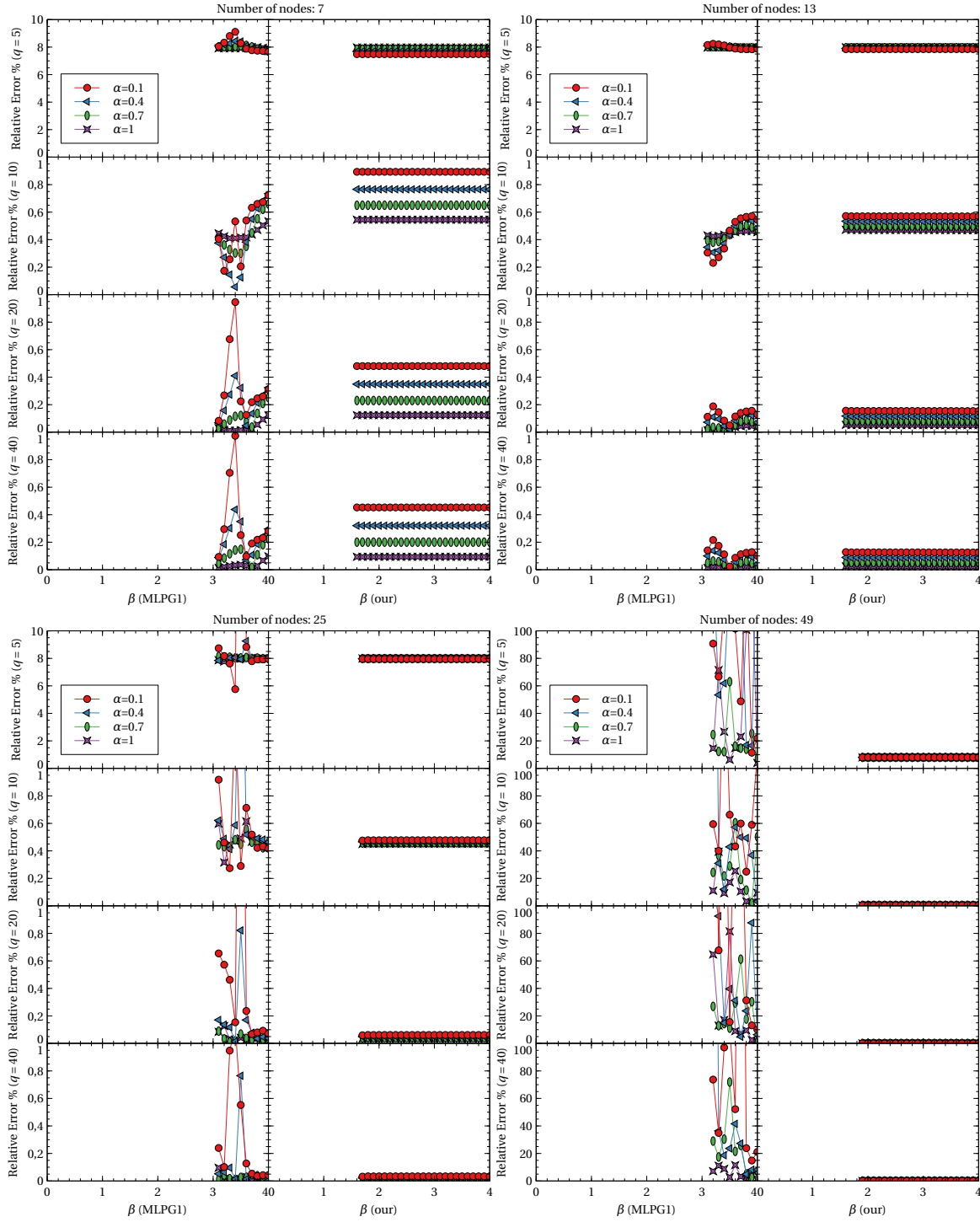
Figure 12: Exhaustive test with cubic polynomial base using 7, 13, 25 and 49 nodes, where $\bar{x}$ is located in accordance with the pattern shown in Figure 7d. The graphs on the left columns are from the original MLPG1 method and on the right columns are from our approach. The tests are performed with 5, 10, 20 and 40 integration points as shown in each row. The y-axis represents the percentage relative error (Eq. (40)), the x-axis shows the values of β, and each curve corresponds to a specific value of α.
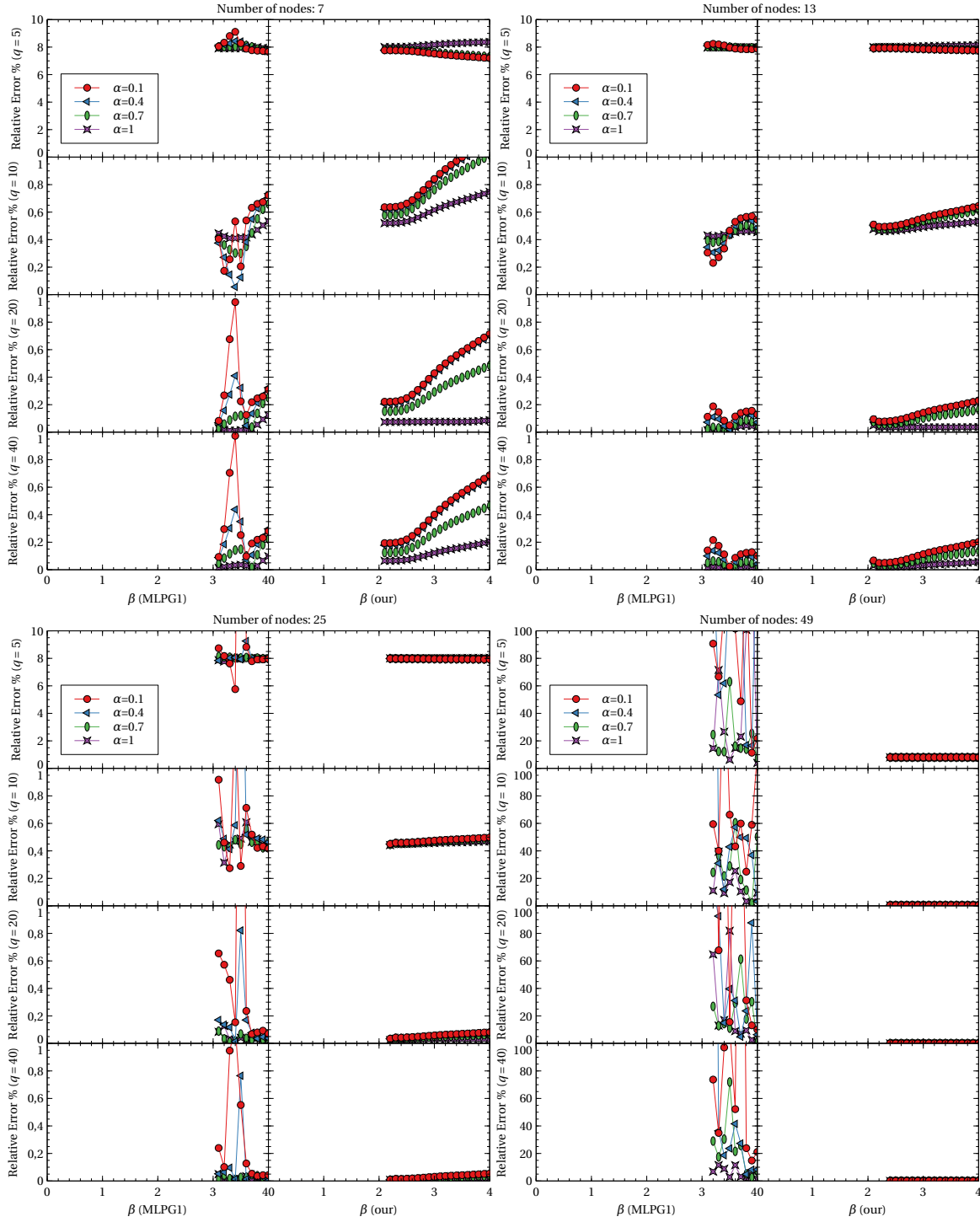
Figure 13: Exhaustive test with quadratic polynomial base using 7, 13, 25 and 49 nodes, where $\bar{x}$ is located in accordance with the pattern shown in Figure 7a. The graphs on the left columns are from the original MLPG1 method and on the right columns are from our approach. The test are performed with 5, 10, 20 and 40 integration points as shown in each row. The y-axis represents the percentage relative error (Eq. (40)), 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 (39).

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 $\mathbf{A}$ (Eq. (15)), it depends on the value of t. Thus, we chose the smallest value of β required by each method.

The results obtained with quadratic and cubic polynomial order (Figs. 11 and 13), using the distribution of $\bar{x}$ as shown in Figure 7a, were similar to those obtained with the minimum number of $\bar{x}$ (Figs. 10 and 12). However, the linear polynomial presented very poor results (see Figure 9). Therefore, in the following tests, we decided to use the minimum amount of $\bar{x}$ (see figures 7b to 7d).

3.2.1 Trial functions of first degree ($t = 1$)

In this tests, the original MLPG1 uses $\beta = 1.1$, and our method uses $\beta = 0.6$ (those were the minimum required values to invert matrix $\mathbf{A}$).

The results are shown in Figure 14. Notice that, each method has approximately the same error, with almost no influence of the number of integration points or of the number of nodes.

3.2.2 Trial function of second degree ($t = 2$)

In this 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 $\mathbf{A}$). Notice (see Fig. 10) 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 7c.

The results are shown in Figure 15. 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.

3.2.3 Trial function of cubic degree ($t = 3$)

In this 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 $\mathbf{A}$). For our method, we use the trial function distribution pattern shown in Figure 7d.

The results are shown in Figure 16. 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. 12, 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.

3.2.4 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. (39)). Table 1 shows comparative results for the displacement of that point, including the absolute value of the relative error, i.e.

$$error = \left| \frac{u - u^h}{u} \right| \times 100. \tag{41}$$

As expected, those quantitative results confirm the qualitative results shown in Figures 15 and 16. In the quadratic case, all the relative errors of both methods are less than 0.1%. However, in the cubic case, our
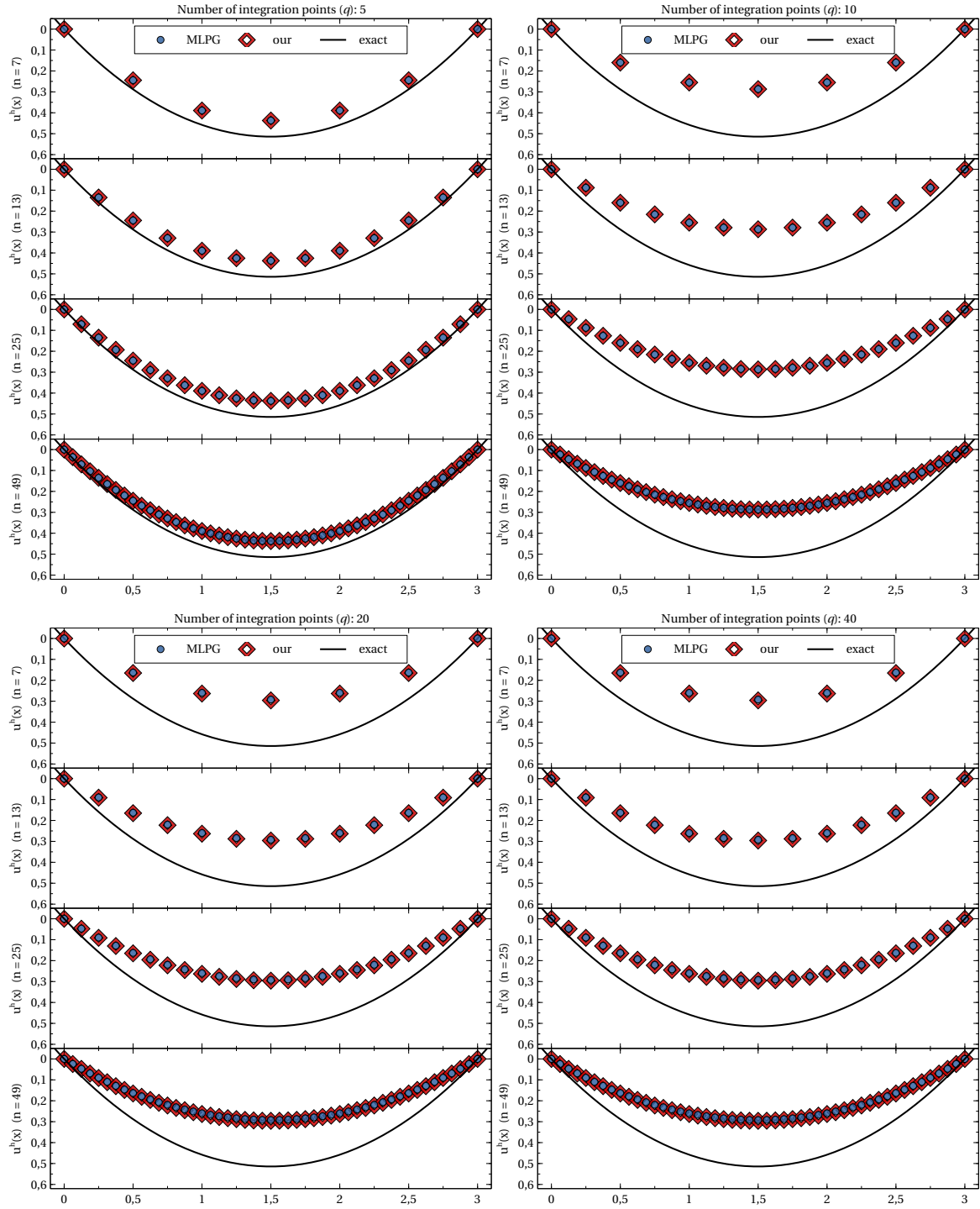
Figure 14: Comparison of MLPG1 method with our approach for the linear polynomial order. The displacements are shown at each node, using uniform discretization with 7, 13, 25 and 49 nodes; and 5, 10, 20 and 40 integration points. The support radii of the test functions were fixed at $\alpha = 0.7$, and the radii for the trial functions were: $\beta = 1.1$ for the original MLPG1 method and $\beta = 0.6$ for our method.
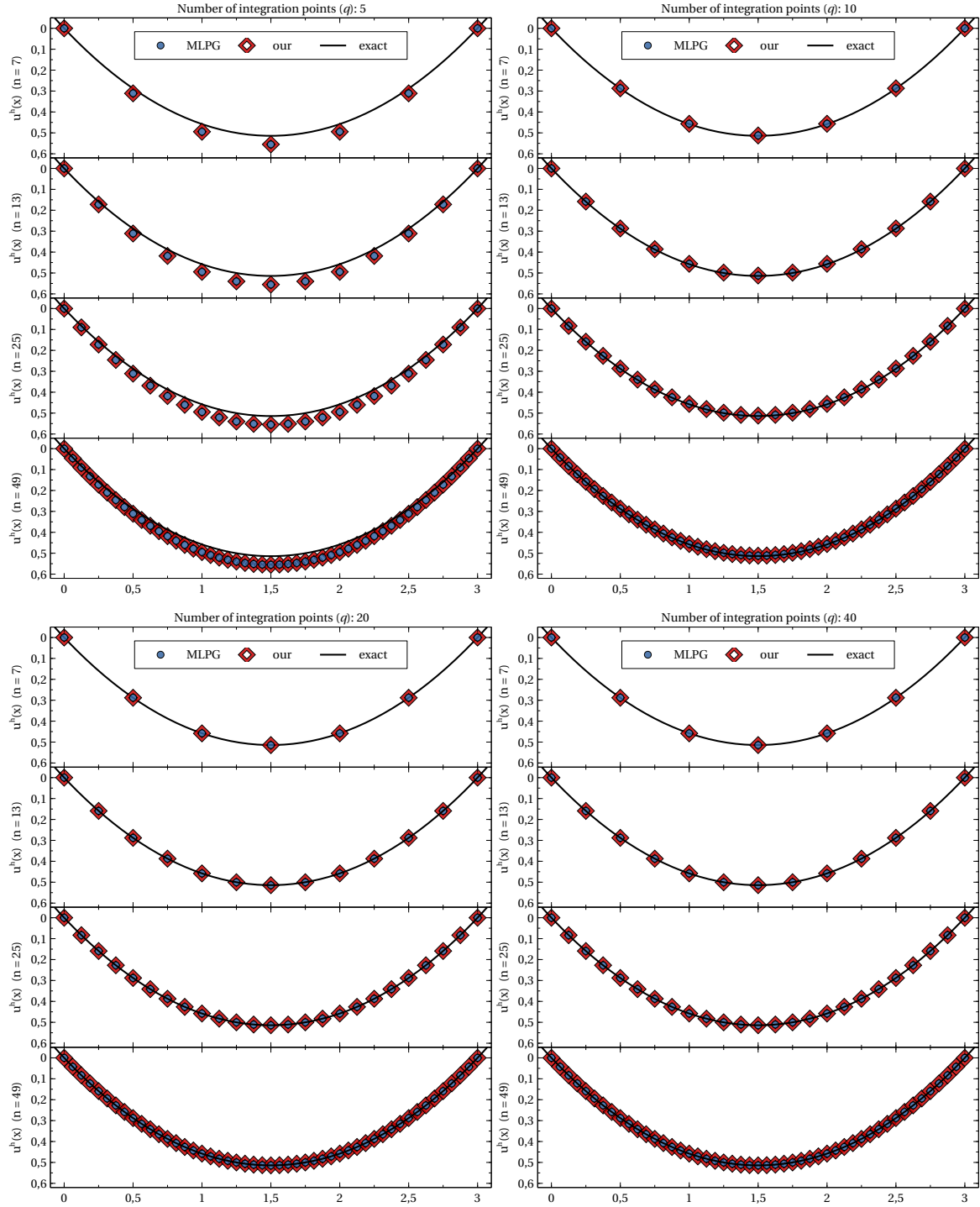
Figure 15: Comparison of the MLPG1 method with our approach for the quadratic polynomial order, where $\bar{x}$ is located in accordance with the pattern shown in Figure 7c. The displacements are shown at each node, using uniform discretization with 7, 13, 25 and 49 nodes; and 5, 10, 20 and 40 integration 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.
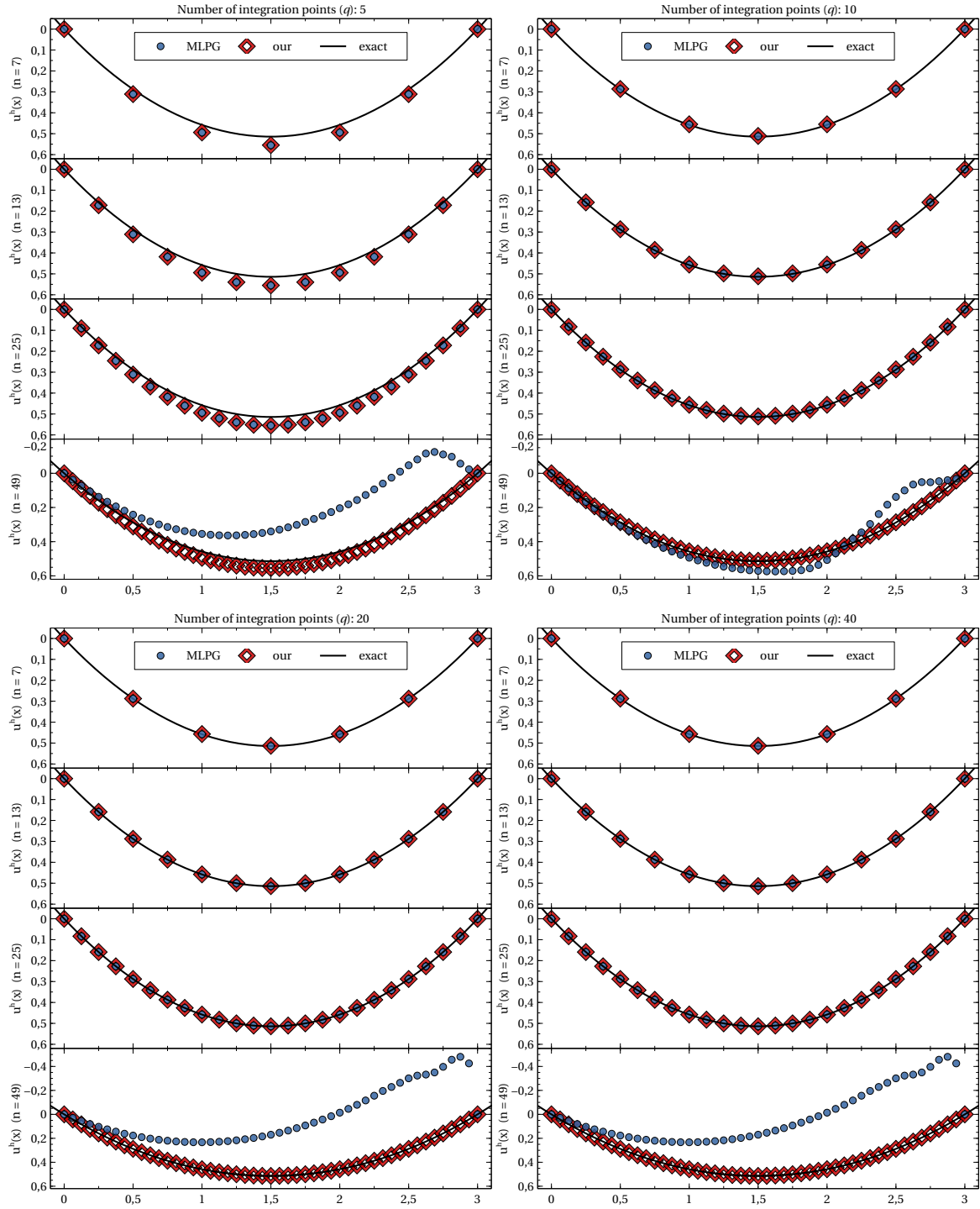
Figure 16: Comparison of MLPG1 method with our approach for the cubic polynomial order. The displacements are shown at each node, using uniform discretization with 7, 13, 25 and 49 nodes; and 5, 10, 20 and 40 integration points. The support radii of the test functions were fixed at $\alpha = 0.7$, and the radii for the trial functions were: $\beta = 3.2$ for the original MLPG1 method and $\beta = 1.9$ for our method.

Table 1: Approximate displacements of the cable's midpoint. The exact displacement is 0.514187m. 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
linear	7	0.297458	42.150%	0.298432	41.960%
	13	0.297583	42.126%	0.298419	41.963%
	25	0.297614	42.120%	0.298416	41.964%
	49	0.297622	42.118%	0.298415	41.964%
quadratic	7	0.514408	0.043%	0.514620	0.084%
	13	0.514428	0.047%	0.514326	0.027%
	25	0.514269	0.016%	0.514214	0.005%
	49	0.514196	0.002%	0.514190	0.001%
cubic	7	0.514710	0.102%	0.513120	0.208%
	13	0.514526	0.066%	0.513914	0.053%
	25	0.514403	0.042%	0.514112	0.015%
	49	−0.463295	190.102%	0.514195	0.002%

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. As in the previous results, the linear test is the least reliable, with errors bigger than 40%.

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

Since the analytical result shown in Equation (39) 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 17 to 19 show, respectively, the results for the linear, quadratic and cubic cases ($t = 1, 2$ 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. In the tests with linear trial functions, both methods did not achieve the expected results, i.e, similar to those presented in Figure 8.

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

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 Figs. 7a and 7c). The simulations were repeated ten thousand times and the average simulation time was computed. The simulations used 7, 13, 25 and 49 nodes with $t = 2$ (quadratic polynomial order), using 5 and 40 integration points.

The results plotted in Figure 20 indicate that our approach outperforms MLPG1. Considering, for example, the 40-integration-point cases, which deliver the most accurate results, our method is approximately
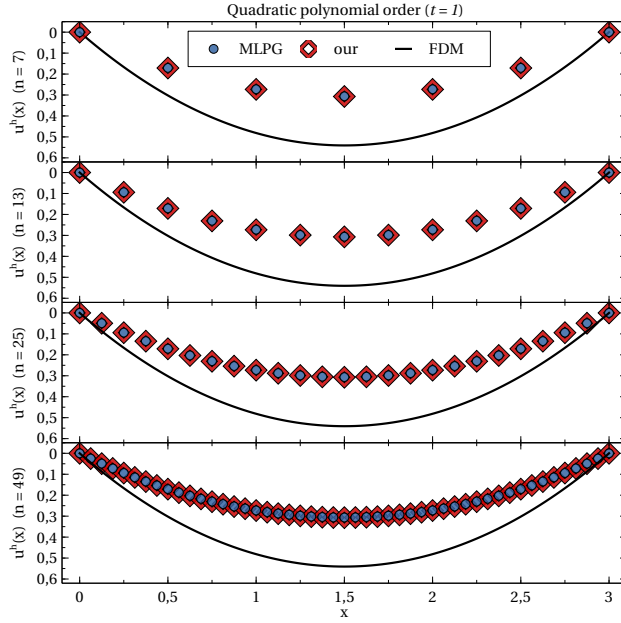
Figure 17: Tests using $k(x) = \sin(x\pi/L)$ for the linear case. The displacements are shown at each node, using uniform discretization with 7, 13, 25 and 49 nodes; and 40 integration points. The support radii of the test functions were fixed at $\alpha = 0.7$ and for the trial functions the values of β are $\beta = 1.1$ (original MLPG1 method) and $\beta = 2.1$ (our method).
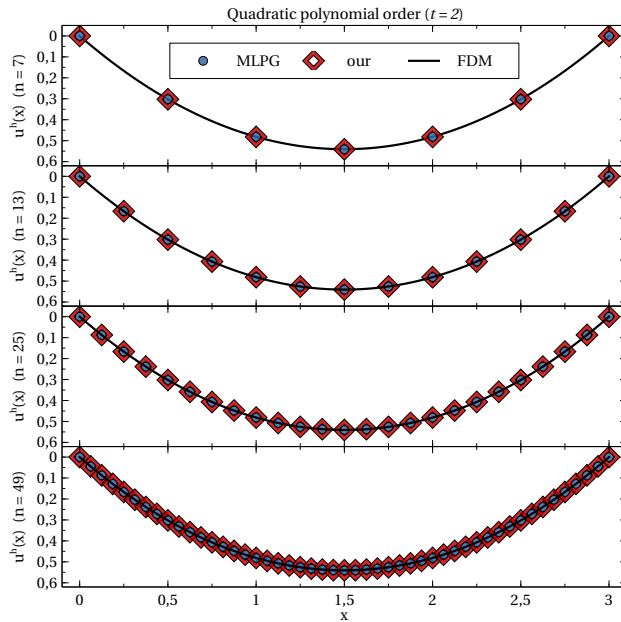

Figure 18: Tests using $k(x) = \sin(x\pi/L)$ for the quadratic case. The displacements are shown at each node, using uniform discretization with 7, 13, 25 and 49 nodes; and 40 integration points. The support radii of the test functions were fixed at $\alpha = 0.7$ and for the trial functions the values of β are $\beta = 2.1$ (original MLPG1 method) and $\beta = 1.1$ (our method).
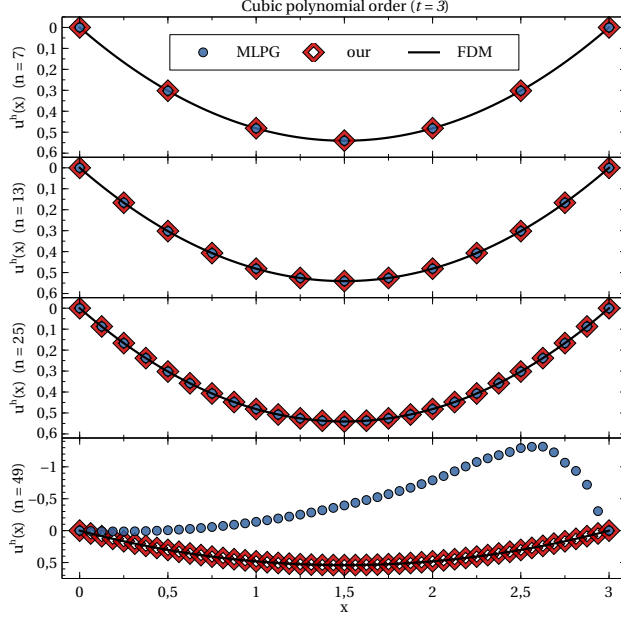
Figure 19: Tests using $k(x) = \sin(x\pi/L)$ for the cubic case. The displacements are shown at each node, using uniform discretization with 7, 13, 25 and 49 nodes; and 40 integration points. The support radii of the test functions were fixed at $\alpha = 0.7$ and for the trial functions the values of β are $\beta = 3.2$ (original MLPG1 method) and $\beta = 1.9$ (our method).

Table 2: Approximate displacements of the cable's midpoint. The "true" displacement of 0.540648m was computed using the finite difference method 1,000 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
linear	7	0.306081	43.386%	0.307106	43.197%
	13	0.306239	43.360%	0.307123	43.194%
	25	0.306278	43.350%	0.307127	43.193%
	49	0.306288	43.348%	0.307128	43.193%
quadratic	7	0.541038	0.072%	0.540961	0.058%
	13	0.540855	0.038%	0.540732	0.016%
	25	0.540703	0.010%	0.540659	0.002%
	49	0.540646	0.000%	0.540647	0.000%
cubic	7	0.541381	0.136%	0.540229	0.077%
	13	0.540906	0.048%	0.540500	0.027%
	25	0.540840	0.036%	0.540603	0.008%
	49	-0.395159	173.090%	0.540668	0.004%

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.

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 10, for 5 integration points, regardless of the number of nodes,
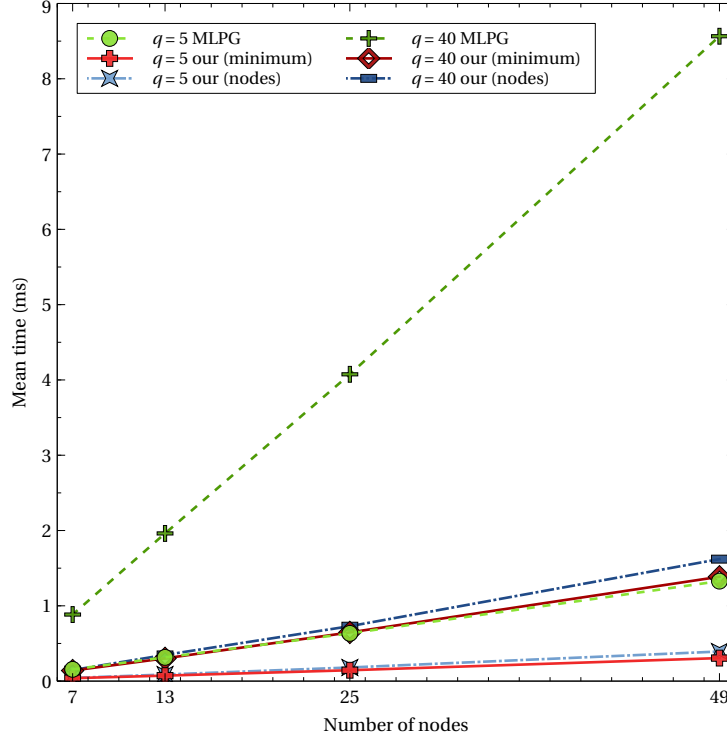
Figure 20: Average run time vs discretization. All simulations were performed for the quadratic polynomial order ($t = 2$), considering the two distribution patterns for $\bar{x}$ shown in Figure 7c. The discretizaitons used 7, 13, 25 and 49 nodes, and the integration used 5 and 40 integration points. 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).

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 [2] 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 with the approximating polynomial's coefficients are optimized should be dissociated from the place were 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. In the linear case, MLPG1 suffers of the same sensitivity problems to α and β variations which makes it difficult to give a proper recommendation. On the other hand, with our method, placing $\bar{x}$ at every node over-constrains the trial functions and the results do not improve. For $\bar{x}$ placed halfway between nodes allows us to find proper values of α and β (for

the problem with a constant $k(x)$, $\alpha = 0.6$ and $\beta = 0.75$ delivered a result with percentage error of 0.667%) but the solution was also very sensitive to variations of those parameters. It seems that the integration by parts does not relax the requirement that the trial function for the second-order boundary value problem has to be at least a quadratic polynomial.

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

References

[1] Ramin Amini, mohammad akbarmakoui, and Seyed Mojtaba and Mosavi Nezhad. Fluid flow modeling in channel using meshless local petrov-galerkin (mlpg) method by radial basis function. *Modares Mechanical Engineering*, 18(8), 2018.

[2] S. N. Atluri and S. Shen. 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*, 2002.

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

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

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

[6] G.R. Buchanan. *Schaum's Outline of Finite Element Analysis*. Schaum's Outline Series. McGraw-Hill Education, 1994. ISBN 9780071502887.

[7] R. A. Gingold and J. J. Monaghan. Smoothed Particle Hydrodynamics: Theory and Application to Non-Spherical Stars. *Mon. Not. Roy. Astron. Soc.*, 181:375, 1977.

[8] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3.3.7. http://eigen.tuxfamily.org, 2019.

[9] Z. D. Han and S. N. Atluri. Meshless Local Petrov-Galerkin (MLPG) Approaches for Solving 3D Problems in Elasto-Statics. *CMES- Computer Modeling in Engineering & Sciences*, 6:169–188, 2004.

[10] Zhidong Han, Arunachalam Rajendran, and S n. Atluri. Meshless local petrov-galerkin (mlpg) approaches for solving nonlinear problems with large deformations and rotations. *CMES. Computer Modeling in Engineering & Sciences*, 10, 10 2005.

[11] Dean Hu and Zhanhua Sun. The meshless local petrov-galerkin method for large deformation analysis of hyperelastic materials. *ISRN Mechanical Engineering*, 2011, 09 2011.

[12] Maryam Kamranian, Mehdi Dehghan, and Mehdi Tatari. An adaptive meshless local petrov–galerkin method based on a posteriori error estimation for the boundary layer problems. *Applied Numerical Mathematics*, 111:181 – 196, 2017.

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

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

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

[16] Davoud Mirzaei. A new low-cost meshfree method for two and three dimensional problems in elasticity. *Applied Mathematical Modelling*, 39(23):7181–7196, 2015.

[17] Davoud Mirzaei and Robert Schaback. Direct meshless local petrov–galerkin (dmlpg) method: A generalized mls approximation. *Applied Numerical Mathematics*, 68:73 – 82, 2013.

[18] S N. Atluri and Tulong Zhu. The meshless local petrov-galerkin (mlpg) approach for solving problems in elasto-statics. *Computational Mechanics*, 25:169–179, 03 2000. doi: 10.1007/s004660050467.

[19] S N. Atluri, J Y Cho, and Hyoung-Gon Kim. Analysis of thin beams, using the meshless local petrov–galerkin method, with generalized moving least squares interpolations. *Computational Mechanics*, 24:334–347, 11 1999.

[20] S. N. Atluri, J Y Cho, and Hyoung-Gon Kim. A critical assessment of the truly meshless local petrov-galerkin (mlpg), and local boundary integral equation (lbie) methods. *Computational Mechanics*, 24: 348–372, 11 1999.

[21] Nasrin Sheikhi, Mohammad Najafi, and Vali Enjilela. Extending the meshless local petrov–galerkin method to solve stabilized turbulent fluid flow problems. *International Journal of Computational Methods*, 16(01), 2019.

[22] J. Sladek, V. Sladek, P. Solek, and Ch. Zhang. 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, 2010.

[23] J Sladek, P Stanak, ZD Han, V Sladek, and SN Atluri. Applications of the MLPG Method in Engineering & Sciences: A Review. *CMES: Computer Modeling in Engineering & Sciences*, 92(5):423–475, 2013.

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

[25] Xi Zhang, Z. Yao, and Z. Zhang. Application of MLPG in Large Deformation Analysis. *Acta Mechanica Sinica*, 22(4):331–340, 2006.

UFC

Department of Computing
Block 910 - Campus of Pici
Humberto Monte Avenue
60.455-760 - Rodolfo Teófilo
Fortaleza - CE - Brazil
55 85 3366-9847 - 55 85 3366-9837