User's Manual of RKPM 2D Version 1.0 (Linear Static Analysis)

Prepared by

Jiun-Shyan Chen[1] *(Principal Investigator)*

Tsung-Hui Huang[1] *(Graduate Student Researcher)*

Haoyan Wei[1] *(Postdoctoral Researcher)*

Michael Hillman[2] *(Co-Principal Investigator)*

[1]University of California, San Diego, La Jolla, CA, USA

[2]The Pennsylvania State University, State College, PA, USA

Prepared for

Short Course of Introduction to Meshfree Methods: Fundamentals and Applications
Interdisciplinary Training and Networking in Engineering and Next Generation in Simulation and Experimentation (INTENSE)

May, 2019

Table of Contents

Disclaimer

Issue Date: May, 2019

1 Introduction

In recent years, the Reproducing Kernel Particle Method (RKPM) [1, 2, 3] has been recognized as an effective numerical method for solving partial differential equations (PDEs). Compared to conventional mesh-based numerical methods such as the Finite Element Method (FEM), the reproducing kernel (RK) approximations in RKPM are constructed based on a set of scattered points without any mesh connectivity, thus the strong tie between the quality of the discretization and the quality of approximation in conventional mesh-based methods is relaxed. This "meshfree" feature makes RKPM well-suited for solving large deformation and multiphysics problems where FEM suffers from mesh-distortion or element entanglement [1, 4, 5]. In addition, RKPM provides controllable orders of continuity and completeness, independent from one another, which enables effective solutions of PDEs involving high-order smoothness or discontinuities, and accordingly, implementation of h- and p-adaptive refinement [6, 7, 8, 9, 10] becomes straightforward. Furthermore, the wavelet-like multi-resolution properties can be reproduced in RK approximation, making it suitable for multi-resolution and multi-scale modeling [6, 7, 8]. Recently, accelerated and convergent RKPM formulations have been developed with the employment of variationally consistent and stabilized nodal integration techniques, for effective numerical solution of PDEs [11, 12]. With abovementioned advantages, RKPM has been successfully applied to a number of challenging engineering problems, including thin shells [13, 14], manufacturing processes [15, 16, 17], image-based biomechanics [18], geomechanics [19, 20], fracture/damage mechanics [21, 22, 23], shock dynamics [24, 25] and penetration/fragmentation phenomena [26, 27, 28], to name a few. Interested readers can refer to [2, 29] for a comprehensive review of RKPM and its applications.

A public domain RKPM-based source code is in high demand. Therefore, we present an RKPM-based open-source computational software (named RKPM2D) that can effectively solve linear PDEs in a 2-D domain with an arbitrary geometry.

1.1 Overview, formulation, and code capabilities

The open-source software, RKPM2D version 1.0, is a two-dimensional RKPM-based code developed for the static analysis of linear partial differential problems. The code is based on RKPM with the following features:

- User-friendly MATLAB program for straightforward meshfree analysis and easy implementation and modification for new functionalities.
- Subroutine for discretization of two-dimensional domains of arbitrary geometry and nodal representative domain creation through Voronoi diagram partitioning.
- A complete meshfree Galerkin equation solver with two types of domain integration: stabilized nodal integration, and conventional background Gauss integration.
- Built-in visualization tools for post-processing of the numerical results.

The RKPM2D code is implemented under a MATLAB environment [30] with pre-processing, solver, and post-processing functions fully integrated, for supporting reproducible research and serving as an efficient test platform for further development of meshfree methods. Both the MATLAB built-in mesh generator and standard neutral files exported by other mesh generators can be used to obtain the point-based domain discretization for meshfree analysis. A meshfree Galerkin equation solver for 2-dimensional elastostatics, and visualization tools for post-processing are provided. Nitsche's method [31, 32] is adopted for imposition of essential boundary conditions. Spatial domain integration includes background Gauss integration developed in [33, 34], direct nodal integration, and variationally consistent stabilized conforming nodal integration (SCNI) [11, 35]. For nodal integration, two different types of stabilized techniques are implemented in RKPM2D, Modified Stabilized Conforming Nodal Integration (MSCNI) [36] and Naturally Stabilized Nodal Integration (NSNI) [12], in addition to the conventional Gauss quadrature scheme.

For demonstration purposes, linear elasticity is chosen as the model problem, but slight modification of the subroutines can allow one to solve other types of PDEs as well. For instance, extensions of the RKPM open-source software to solve Poisson equations are provided in the appendix.

2 Basic Theory

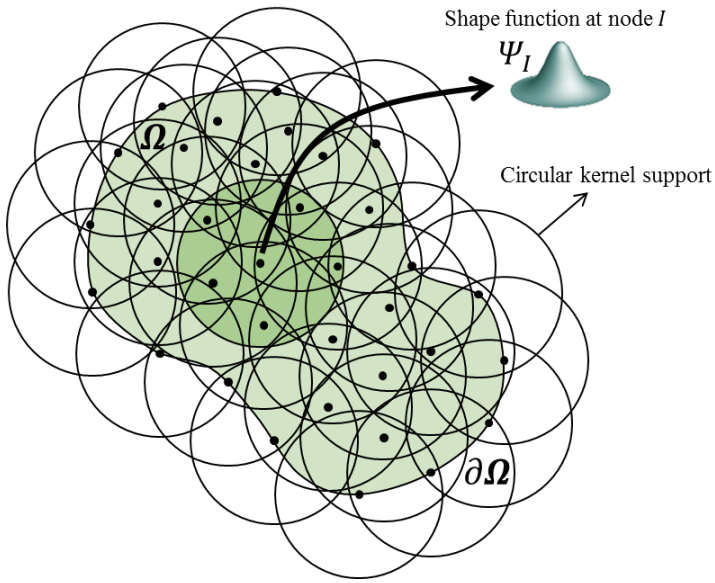

Figure 1. Illustration of 2D RK discretization, support coverage and nodal shape function, with circular kernel support employed.

In RKPM, the numerical approximation is constructed based upon a set of scattered nodes (also called points) [37]. A domain Ω is discretized by a set of nodes $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots \boldsymbol{x}_{NP}\}$ as shown in Figure 1, where $\boldsymbol{x}_I$ is the position vector of node I, and NP is the total number of nodes. The RK approximation of a function u is expressed as

$$u(\boldsymbol{x}) \approx u^h(\boldsymbol{x}) = \sum_{I \in G_x} \Psi_I(\boldsymbol{x}) u_I \tag{1}$$

where $\boldsymbol{x}$ is the spatial coordinates, u_I is the associated nodal coefficient to be determined, and $\Psi_I(\boldsymbol{x})$ is the reproducing kernel (RK) shape function of node I expressed as:

$$\Psi_I(\boldsymbol{x}) = \boldsymbol{H}^T(\boldsymbol{0})\boldsymbol{M}^{-1}(\boldsymbol{x})\boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}_I)\Phi_a(\boldsymbol{x} - \boldsymbol{x}_I) \tag{2}$$

where the basis vector $\boldsymbol{H}(\boldsymbol{x}-\boldsymbol{x}_I)$ is defined as

$$\boldsymbol{H}^T(\boldsymbol{x}-\boldsymbol{x}_I) = [1, x_1 - x_{1I}, x_2 - x_{2I}, x_3 - x_{3I}, (x_1 - x_{1I})^2, \dots, (x_3 - x_{3I})^n\,] \tag{3}$$

and $\boldsymbol{M}(\boldsymbol{x})$ is the moment matrix:

$$\boldsymbol{M}(\boldsymbol{x}) = \sum_{I \in G_{\boldsymbol{x}}} \boldsymbol{H}(\boldsymbol{x}-\boldsymbol{x}_I)\boldsymbol{H}^T(\boldsymbol{x}-\boldsymbol{x}_I)\Phi_a(\boldsymbol{x}-\boldsymbol{x}_I) \tag{4}$$

The set $G_{\boldsymbol{x}} = \{I | \Phi_a(\boldsymbol{x}-\boldsymbol{x}_I) \neq 0\,\}$ shown in equation (1) and (4) contains the nodal indices of a point $\boldsymbol{x}'s$ neighbors, and $\Phi_a(\boldsymbol{x}-\boldsymbol{x}_I)$ is the kernel function centered at $\boldsymbol{x}_I$ with compact support size a_I defined as

$$a_I = \tilde{c}h_I \tag{5}$$

In the above equation, $\tilde{c}$ is the normalized support size, and h_I is the nodal spacing associated with nodal point $\boldsymbol{x}_I$ defined as:

$$h_I = max\left(\left\|\boldsymbol{x}_J - \boldsymbol{x}_I\right\|\right), \;\; \forall \boldsymbol{x}_J \in B_I \tag{6}$$

in which the set B_I is chosen to contain the four nodes that are closest to point $\boldsymbol{x}_I$ for 2D problems. As an example of a kernel function, a C^2 cubic B-spline kernel function can be written as:

$$\Phi_a(\boldsymbol{x}-\boldsymbol{x}_I) = \begin{cases} 2/3 - 4z_I^2 + 4z_I^3 & \text{for} \quad 0 \le z_I \le 1/2\,, \\ 4/3 - 4z_I + 4z_I^2 - 4/3\, z_I^3 & \text{for} \quad 1/2 \le z_I \le 1, \\ 0 & \text{for} \quad z_I > 1, \end{cases} \tag{7}$$

in which z_I is defined as $z_I = \frac{\|x - x_I\|}{a_I}$.

By construction, the RK shape functions satisfy the following n^{th} order reproducing conditions:

$$\sum_{I \in G_{\boldsymbol{x}}} \Psi_I(\boldsymbol{x}) x_{1I}^i x_{2I}^j x_{3I}^k = x_1^i x_2^j x_3^k, \qquad 0 \le i + j + k \le n \tag{8}$$

where n is the specified order of completeness, which determines the order of consistency in the solution of PDEs.

2.1 Galerkin Formulation

Consider the following linear elasticity problem:

$$\begin{aligned} \sigma_{ij,j} + b_i &= 0 \quad \text{on } \Omega \\ \sigma_{ij} n_j &= t_i \quad \text{on } \partial\Omega_t \\ u_i &= g_i \quad \text{on } \partial\Omega_g \end{aligned} \tag{9}$$

Where u_i is the displacement, $\sigma_{ij} = C_{ijkl}\varepsilon_{kl}$ is the Cauchy stress, C_{ijkl} is the elasticity tensor, $\varepsilon_{ij} = \frac{1}{2}\left(u_{i,j} + u_{j,i}\right)$ is the strain, n_j is the surface normal on $\partial\Omega$, b_i is the body force, and t_i and g_i denote the prescribed traction and displacement on $\partial\Omega_t$ and $\partial\Omega_g$, respectively. Using Nitsche's method [38] for the enforcement of essential boundary conditions, the weak form of Eq. (9) is written as follows

$$\begin{aligned} &\int_{\Omega} \delta\varepsilon_{ij} C_{ijkl} \varepsilon_{kl} d\Omega \\ &\qquad = \int_{\Omega} \delta u_i b_i d\Omega + \int_{\partial\Omega_t} \delta u_i t_i d\Gamma + \int_{\partial\Omega_g} \delta u_i \lambda_i d\Gamma \\ &\qquad + \int_{\partial\Omega_g} \delta\lambda_i (u_i - g_i) d\Gamma + \beta \int_{\partial\Omega_g} \delta u_i (u_i - g_i) d\Gamma \end{aligned} \tag{10}$$

where λ_i is the Lagrange multiplier, and in Nitsche's method it is taken as the surface traction for elasticity problems, i.e., $\lambda_i = \sigma_{ij} n_j$, and $\beta = \beta_{nor} E/\bar{h}$, with β_{nor} the normalized penalty parameter, E the Young's modulus, and $\bar{h}$ the average of nodal spacing. Considering the following RK approximation for $\boldsymbol{u}$ and $\delta\boldsymbol{u}$:

$$\boldsymbol{u}^h = \sum_{I \in G_{\boldsymbol{x}}} \Psi_I(\boldsymbol{x}) \boldsymbol{u}_I, \quad \delta\boldsymbol{u}^h = \sum_{I \in G_{\boldsymbol{x}}} \Psi_I(\boldsymbol{x}) \delta\boldsymbol{u}_I, \tag{11}$$

Eq. (10) yields the following matrix equations:

$$\sum_J \boldsymbol{K}_{IJ}\boldsymbol{u}_J - \boldsymbol{F}_I = \boldsymbol{0}, \qquad \forall I \tag{12}$$

where

$$\boldsymbol{K}_{IJ} = \boldsymbol{K}_{IJ}^c + \boldsymbol{K}_{IJ}^\beta - \left(\boldsymbol{K}_{IJ}^g + \boldsymbol{K}_{IJ}^{g\,T}\right) \tag{13}$$

$$\boldsymbol{F}_I = \boldsymbol{F}_I^b + \boldsymbol{F}_I^t + \boldsymbol{F}_{IJ}^\beta - \boldsymbol{F}_I^g \tag{14}$$

in which each matrices and vectors for 2-dimensional elasticity are expressed as

$$\boldsymbol{K}_{IJ}^c = \int_\Omega \boldsymbol{B}_I^T(\boldsymbol{x})\boldsymbol{C}\boldsymbol{B}_J(\boldsymbol{x})d\Omega \tag{15}$$

$$\boldsymbol{K}_{IJ}^\beta = \beta \int_{\partial\Omega_g} \boldsymbol{\Psi}_I^T(\boldsymbol{x})\boldsymbol{S}\boldsymbol{\Psi}_J(\boldsymbol{x})d\Gamma \tag{16}$$

$$\boldsymbol{K}_{IJ}^g = \int_{\partial\Omega_g} \boldsymbol{B}_I^T(\boldsymbol{x})\boldsymbol{C}\boldsymbol{\eta}\boldsymbol{S}\boldsymbol{\Psi}_J(\boldsymbol{x})d\Gamma \tag{17}$$

$$\boldsymbol{F}_I^b = \int_\Omega \boldsymbol{\Psi}_I^T(\boldsymbol{x})\boldsymbol{b}(\boldsymbol{x})d\Omega \tag{18}$$

$$\boldsymbol{F}_I^t = \int_{\partial\Omega_t} \boldsymbol{\Psi}_I^T(\boldsymbol{x})\boldsymbol{t}(\boldsymbol{x})d\Gamma \tag{19}$$

$$\boldsymbol{F}_{IJ}^\beta = \beta \int_{\partial\Omega_g} \boldsymbol{\Psi}_I^T(\boldsymbol{x})\boldsymbol{S}\boldsymbol{g}d\Gamma \tag{20}$$

$$\boldsymbol{F}_I^g = \int_{\partial\Omega_g} \boldsymbol{B}_I^T(\boldsymbol{x})\boldsymbol{C}\boldsymbol{\eta}\boldsymbol{S}\boldsymbol{g}d\Gamma \tag{21}$$

$$\boldsymbol{B}_I(\boldsymbol{x}) = \begin{bmatrix} \Psi_{I,1}(\boldsymbol{x}) & 0 \\ 0 & \Psi_{I,2}(\boldsymbol{x}) \\ \Psi_{I,2}(\boldsymbol{x}) & \Psi_{I,1}(\boldsymbol{x}) \end{bmatrix}, \qquad \boldsymbol{\Psi}_I(\boldsymbol{x}) = \begin{bmatrix} \Psi_I(\boldsymbol{x}) & 0 \\ 0 & \Psi_I(\boldsymbol{x}) \end{bmatrix}, \tag{22}$$

$$\boldsymbol{\eta} = \begin{bmatrix} n_1 & 0 \\ 0 & n_2 \\ n_2 & n_1 \end{bmatrix}, \quad \boldsymbol{S} = \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \quad \boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}, \quad \boldsymbol{g} = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}. \tag{23}$$

where n_i is the component of the surface unit normal on the essential boundary, and $s_i = 0$ or 1 serves as a switch for imposing the boundary displacement g_i. For instance, if $s_1 = 1$ and $s_2 = 0$, then g_1 is imposed.

2.2 Domain Integration

Domain integration plays an important role in accuracy, stability and convergence of meshfree methods. Quadrature domains for meshfree methods can be chosen either as background cells that are independent from the point discretization or associated with the nodal representative domains. The former scheme is commonly adopted in conjunction with the Gauss quadrature scheme and the latter is used for the nodal integration schemes; both of which have been implemented in RKPM2D as discussed in this section.

2.2.1 Gauss Integration

When Gauss quadrature is adopted, quadrature points are generated based upon the background cells [39, 40] as shown in Figure 2, where only the quadrature points inside the physical domain are considered for domain integration. Gauss points for contour integrals are generated along the natural and essential boundaries as shown in Figure 2.

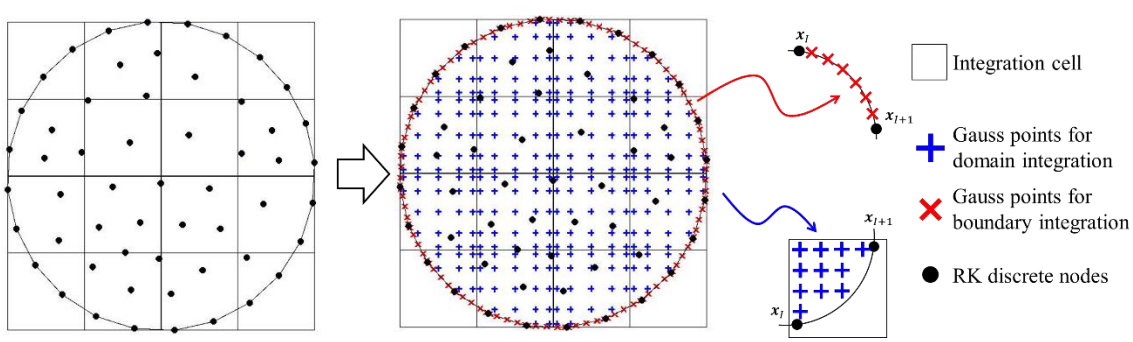

Figure 2. Meshfree nodes and generation of background Gauss quadrature points in an arbitrary two-dimensional domain Ω.

2.2.2 Stabilized Conforming Nodal Integration

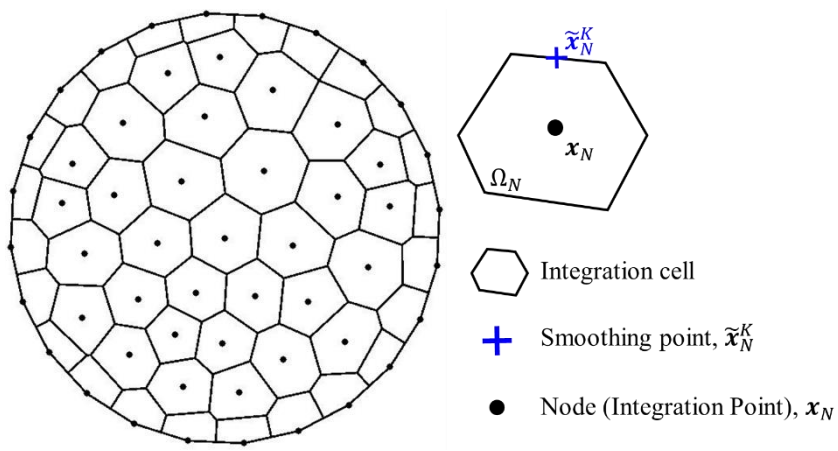

Figure 3. Voronoi cell diagram in a two-dimensional domain Ω.

Direct nodal integration (DNI) is notorious for spurious zero-energy modes and non-convergent numerical solutions. To ensure linear variational consistency, also known as passing the linear patch test, Chen et al. [35] showed that the quadrature rules in the Galerkin formulation need to meeting the following first order integration constraint on the gradient of shape function:

$$\hat{\int_{\Omega}} \Psi_{I,i} d\Omega = \hat{\int_{\partial\Omega}} \Psi_I n_i d\Gamma \tag{24}$$

In (24), ^ over the integral symbol denotes numerical integration. If nodal integration is introduced as the quadrature rule for the domain integration on the left hand side of (24), Chen et al. [35] introduced the following nodally smoothed gradient $\widetilde{\Psi}_{I,i}$ at the nodal point $\boldsymbol{x}_N$ for nodal integration:

$$\widetilde{\Psi}_{I,i}(\boldsymbol{x}_N) = \frac{1}{A_N} \int_{\Omega_N} \Psi_{I,i}(\boldsymbol{x}) d\Omega = \frac{1}{A_N} \int_{\partial\Omega_N} \Psi_I(\boldsymbol{x}) n_i(\boldsymbol{x}) d\Gamma \tag{25}$$

where A_N denotes the area of the nodal representative domain Ω_N associated with node N, and n_i denotes the ith component of the outward unit normal vector to the smoothing

domain boundary as shown in Figure 3. It is shown in [35] that integrating (24) with nodal integration and with the nodally smoothed gradient of shape function in (25), the first order integration constraint in (24) is exactly satisfied as long as the same boundary integral quadrature rule is used for the right hand side of both (24) and (25). As discussed in [11], for linear consistency of the smoothed gradient of a linearly consistent shape function, a simple one-point Gauss integration rule used for the contour integral of Eq. (25) is sufficient:

$$\widetilde{\Psi}_{I,i}(\boldsymbol{x}_N) \approx \frac{1}{A_N} \sum_{K \in S_N} \Psi_I(\widetilde{\boldsymbol{x}}_N^K) n_i(\widetilde{\boldsymbol{x}}_N^K) L_K \tag{26}$$

where $S_N = \{K | \widetilde{\boldsymbol{x}}_N^K \in \partial\Omega_N\}$ contains all center points of each boundary segment associated with node $\boldsymbol{x}_N$, and the integration weight L_K is the length of the Kth segment of the smoothing cell boundary.

2.2.3 Stabilized Nodal Integration Schemes

Spurious oscillatory modes can be triggered in nodal integration methods. Therefore, additional stabilization techniques are needed to eliminate these low-energy modes, which will be described in this following.

2.2.3.1 Modified Stabilized Nodal Integration

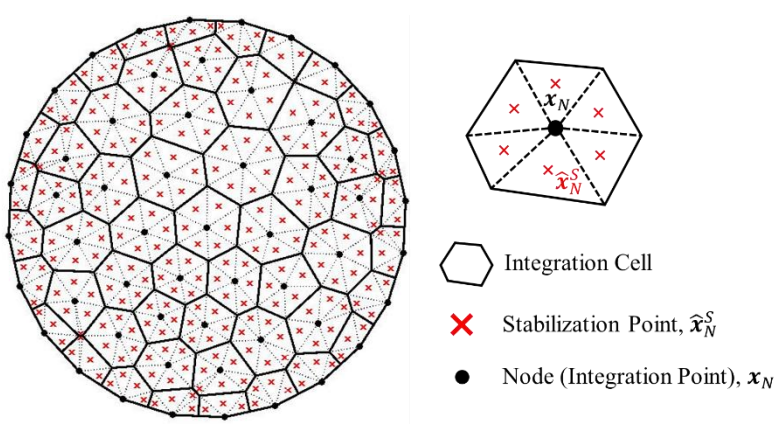

Figure 4. Illustration of nodal integration cells of the modified stabilized nodal integration.

The first stabilization technique employed here for eliminating spurious low-energy modes is called modified stabilized nodal integration [36, 41], where a least-squares type stabilization term is introduced into the stiffness matrix:

$$\boldsymbol{K}_{IJ}^{c} = \sum_{N=1}^{NP} \left(\underbrace{\widetilde{\boldsymbol{B}}_I^T(\boldsymbol{x}_N)\boldsymbol{C}\widetilde{\boldsymbol{B}}_J(\boldsymbol{x}_N)A_N}_{\text{SCNI}} + \underbrace{c_{stab}\sum_{S=1}^{NS}\left(\widetilde{\boldsymbol{B}}_I^T(\boldsymbol{x}_N) - \widetilde{\boldsymbol{B}}_I^T(\widehat{\boldsymbol{x}}_N^S)\right)\boldsymbol{C}\left(\widetilde{\boldsymbol{B}}_J(\boldsymbol{x}_N) - \widetilde{\boldsymbol{B}}_J(\widehat{\boldsymbol{x}}_N^S)\right)A_N^S}_{\text{stabilization}} \right) \tag{27}$$

where NS denotes the number of sub-cells associated with each nodal integration cell (as shown in Figure 4), $\widehat{\boldsymbol{x}}_N^S$ denotes the centroid of the Sth sub-cell, $\widetilde{\boldsymbol{B}}_J(\widehat{\boldsymbol{x}}_N^S)$ is the smoothed gradient evaluated by Eq. (26) for the Sth sub-cell and A_N^S denotes the area of the Sth sub-cell of Nth nodal smoothing-cell. Here, $0 \leq c_{stab} \leq 1$ is a stabilization parameter, which is chosen to be $c_{stab} = 1$ based on the study of Puso et al. [41] for elasticity.

2.2.3.2 Naturally Stabilized Nodal Integration

The other stabilized integration technique employed in RKPM2D is the *naturally stabilized nodal integration* (NSNI) proposed in [12], where an implicit gradient expansion of the strain field is introduced as:

$$\boldsymbol{\varepsilon}\left(\boldsymbol{u}^h(\boldsymbol{x})\right) \approx \boldsymbol{\varepsilon}_N\left(\boldsymbol{u}^h(\boldsymbol{x}_N) + \sum_{i=1}^{d}(x_i - x_{iI})\widehat{\boldsymbol{u}}_{,i}^h(\boldsymbol{x}_N)\right) \tag{28}$$

where $\widehat{\boldsymbol{u}}_{,i}^h(\boldsymbol{x}_N) = \sum_{N=1}^{NP}\Psi_{iI}^{\nabla}(\boldsymbol{x}_N)\boldsymbol{u}_N$ is the implicit gradient of the displacement with Ψ_{iI}^{∇} the implicit RK gradient function [22]:

$$\Psi_{iI}^{\nabla} = \boldsymbol{H}_i^T\boldsymbol{M}^{-1}(\boldsymbol{x})\boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}_I)\Phi_a(\boldsymbol{x} - \boldsymbol{x}_I) \tag{29}$$

where $\boldsymbol{H} = [1 \quad x_1 - x_{1I} \quad x_2 - x_{2I}]^T$ and the vector $\boldsymbol{H}_i$ takes on the following values for linear basis:

$$\begin{aligned} \boldsymbol{H}_1 &= [0 \quad -1 \quad 0]^T \\ \boldsymbol{H}_2 &= [0 \quad 0 \quad -1]^T \end{aligned} \tag{30}$$

Introducing the gradient expansion terms (28) into the variational equations, the stiffness matrix is obtained as

$$\begin{aligned} \boldsymbol{K}_{IJ}^c = \sum_{N=1}^{NP} \Bigg(& \underbrace{\widetilde{\boldsymbol{B}}_I^T(\boldsymbol{x}_N)\boldsymbol{C}\widetilde{\boldsymbol{B}}_J(\boldsymbol{x}_N)A_N}_{\text{SCNI}} \\ & + \underbrace{\boldsymbol{B}_{1I}^{\nabla^T}(\boldsymbol{x}_N)\boldsymbol{C}\boldsymbol{B}_{1J}^{\nabla}(\boldsymbol{x}_N)M_{1N} + \boldsymbol{B}_{2I}^{\nabla^T}(\boldsymbol{x}_N)\boldsymbol{C}\boldsymbol{B}_{2J}^{\nabla}(\boldsymbol{x}_N)M_{2N}}_{\text{stabilization}} \Bigg) \end{aligned} \tag{31}$$

where $\boldsymbol{B}_{1I}^{\nabla}(\boldsymbol{x}_N)$ and $\boldsymbol{B}_{2I}^{\nabla}(\boldsymbol{x}_N)$ are defined as follows:

$$\boldsymbol{B}_{1I}^{\nabla}(\boldsymbol{x}_N) = \begin{bmatrix} \Psi_{I1,1}^{\nabla}(\boldsymbol{x}_N) & 0 \\ 0 & \Psi_{I1,2}^{\nabla}(\boldsymbol{x}_N) \\ \Psi_{I1,2}^{\nabla}(\boldsymbol{x}_N) & \Psi_{I1,1}^{\nabla}(\boldsymbol{x}_N) \end{bmatrix}, \boldsymbol{B}_{2I}^{\nabla}(\boldsymbol{x}_N) = \begin{bmatrix} \Psi_{I2,1}^{\nabla}(\boldsymbol{x}_N) & 0 \\ 0 & \Psi_{I2,2}^{\nabla}(\boldsymbol{x}_N) \\ \Psi_{I2,2}^{\nabla}(\boldsymbol{x}_N) & \Psi_{I2,1}^{\nabla}(\boldsymbol{x}_N) \end{bmatrix} \tag{32}$$

and M_{1N}, M_{2N} are the second moments of inertia in each nodal integration domain:

$$M_{1N} = \int_{\Omega_N} (x_1 - x_{1N})^2 d\Omega, \quad M_{2N} = \int_{\Omega_N} (x_2 - x_{2N})^2 d\Omega \tag{33}$$

From Eqns. (31) - (33), no subdivision of integration cells is required in the stabilization.

3 Get started

In this section, the set-up of RKPM2D for solving linear elasticity will be explained, including the set-up of material properties, domain geometry, boundary conditions, domain discretization method, RK shape function parameters, quadrature rule and control of post-processing. All the source codes and sample scripts (.m files) are provided.

In order to compile all the MATLAB scripts and functions, the code requires the following MATLAB environment:

- MATLAB version (R2018a) or higher
- MATLAB Partial Differential Equation Toolbox™
- MATLAB Mapping Toolbox™

If any error message shows up in the command window during the simulation due to undefined built-in functions, please search for the built-in function name in MATLAB to install the latest version of that function.

3.1 Overall program structure

The general flowchart of RKPM2D is given in Figure 5. Unlike in the FEM procedures where the element type is tied with mesh/nodes generation, the order of basis and smoothness are independent to the domain discretization in RKPM, while the general program functionalities (such as matrix assembly and solver) of a meshfree Galerkin method are similar to that of FEM.

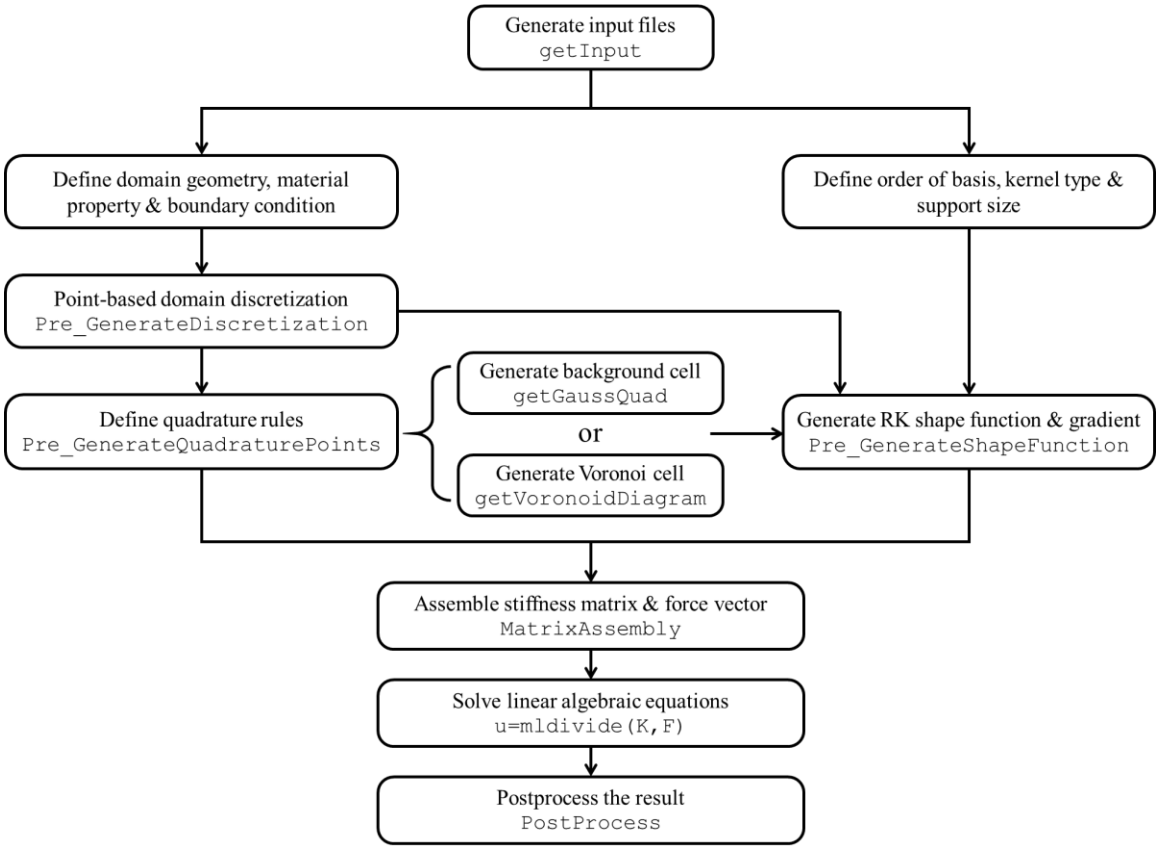

Figure 5. Flowchart of the RKPM2D procedures.

Shown in Figure 5 are the numerical procedures for the model input file generation, domain discretization, quadrature rule definition, shape function construction, matrix assembly, solver, and post-processing. For general purpose, only the input files `getInput.m` needs to be modified for modeling a problem. The input files for solving different linear elasticity problems are provided in the folders of "01_LinearPatchTest" to "06_TensileTest" as shown in Figure 6. The rest of the subroutines are located in the folder "MAIN_PROGRAM". The description for each subroutine will be given in section 3.3. To run the simulation, one needs to execute the main script `MAIN.m` after setting up the model parameters in `getInput.m`

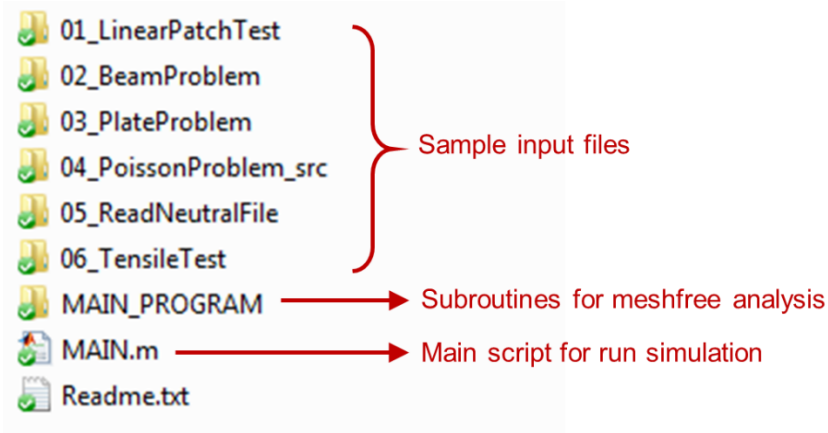

Figure 6. Folders containing input files and subroutines for meshfree analysis.

3.2 Preparation of Input files

To illustrate the procedure to run RKPM2D, let us consider a tensile test for a tapered specimen in a 2D domain $\Omega \subset \mathbb{R}^2$, as shown in Figure 7:

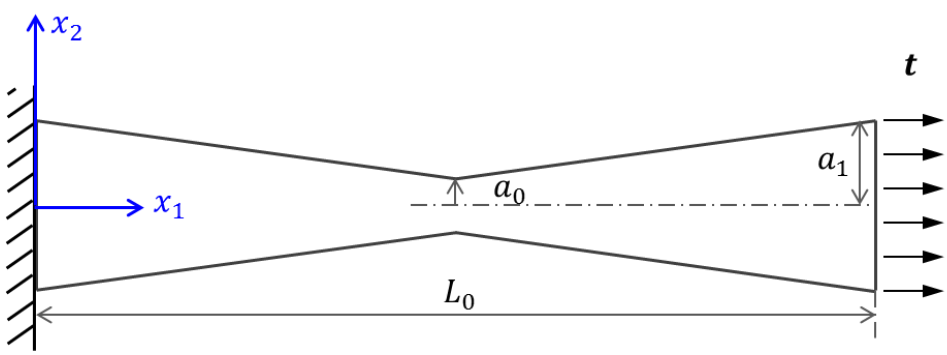

Figure 7. Illustration of the tensile test problem.

The dimensions, material properties and the boundary conditions are given as follows:

- Dimensions
  - $L_0 = 5.3334 \times 10^{-2}$ m, $a_0 = 0.32 \times 10^{-2}$ m, $a_1 = 0.64 \times 10^{-2}$ m.
- Material model: Linear Elasticity
  - E: Young's modulus, 206.9 GPa
  - v : Poisson ratio, 0.29
  - Plane Strain Condition
- Boundary conditions

- Prescribed traction $\boldsymbol{t} = [20.69,0]^T$ GPa at $x_1 = L_0$
- Prescribed displacement $\boldsymbol{g} = [0,0]^T$ m at $x_1 = 0$

To set up the input files, the following steps are needed

1. Create a new folder for input files (e.g. "06_TensileTest").
2. Copy any existing `getInput.m` from other folder and paste it in the new input folder
3. Open `getInput.m`, and edit the input parameters

The preparation of `getInput.m` is discussed in detail in the following subsection.

3.2.1 Setting up the material properties

In `getInput.m`, we first need to edit section (1) Material shown in Listing 1 to set up material parameters:

1. Set `Model.E = 206.9E9` for Young's modulus as $E = 206.9$ GPa
2. Set `Model.nu = 0.29` for Poisson ratio as $v = 0.29$
3. Set `Model.Condition = 'PlaneStrain'` to specify the plane strain condition. For plane stress condition, please use `'PlaneStress'`.

```
%% Read Input Files from Trelis Neutral file
%% (1) Material
% Linear Elasticity
% Lame Parameters for Young's modulus and poisson ratio
Model.E = 206.9E9; Model.nu = 0.29;
Model.Condition = 'PlaneStrain'; % PlaneStress, or PlaneStrain
Model.ElasticTensor =
getElasticTensor(Model.E,Model.nu,Model.Condition);
Model.DOFu = 2;                    % two dimensional problem
```

Listing 1. Defining material properties in the input file.

3.2.2 Setting up the domain geometry

Next, go to section (2) Geometry shown in Listing 2 to define the coordinates of all vertices of the domain. For example, for the tensile test problem shown in Figure 7, set

`x1_vertices = 1E-3*[0, 26.667, 53.334, 53.334, 26.667, 0]'` and `x2_vertices = 1E-3*[-6.2, -3.2, -6.2, 6.2, 3.2, 6.2]'`, where the 2D domain vertices are given in a counter-clock-wise (CCW) order.

```
%% (2) Geometry
% user input
% geometry of the bar
x1_vertices = 1E-3*[0 26.667 53.334 53.334 26.667 0]';
x2_vertices = 1E-3*[-6.2 -3.2 -6.2 6.2 3.2 6.2]';
% ensure the boundary segments to be counter clockwise
[x1_vertices, x2_vertices] = poly2ccw(x1_vertices, x2_vertices);
Model.xVertices = [x1_vertices, x2_vertices];
Model.DomainArea = polyarea(x1_vertices,x2_vertices);
```

Listing 2. Defining domain geometry in the input file.

3.2.3 Setting up the boundary conditions

In section (3) Boundary condition shown in Listing 3, we can define the boundary conditions, including the prescribed traction $\boldsymbol{t}$, body force $\boldsymbol{b}$, displacement $\boldsymbol{g}$, through the following steps:

1. Define the criteria of finding the essential boundary coordinates and natural boundary coordinates (the number `1E-7` is a user-defined tolerance for locating the boundary edges/points)
   a. `Model.CriteriaEBC = @(x1,x2) find(x1 < 0 + 1E-7)` indicates essential boundary edge with coordinates $x_1 < 10^{-7}$
   b. `Model.CriteriaNBC = @(x1,x2) find(x1 > 5.3334E-2 - 1E-7)` indicates natural boundary edges with coordinates $x_1 > 0.053334 - 10^{-7}$
   c. For boundary edges that do not meet the above criteria, they are set to be a zero traction boundary conditions, by default
2. Set `Model.Beta_Nor = 1E2` for normalized Nitsches parameter β_{nor}. It is recommended to set this parameter to be between 10~100.
3. For evaluation of traction $\boldsymbol{t}$, body force $\boldsymbol{b}$, displacement $\boldsymbol{g}$ and the essential boundary switch $\boldsymbol{S}$, RKPM2D provides two ways:

a. If an analytical expression of displacement $\boldsymbol{u}_{exact}$ exists, then we can provide the expression of displacement $\boldsymbol{u}_{exact}$ in a symbolic format in terms of x_1 and x_2 for imposing the boundary condition. In this case, the corresponding boundary conditions will be calculated as $\boldsymbol{t} = \boldsymbol{\eta}^T \boldsymbol{C} \boldsymbol{\varepsilon}^{exact}$, $\boldsymbol{g} = \boldsymbol{u}^{exact}$, $\boldsymbol{b} = \boldsymbol{\sigma}_{ij,j}^{exact}$, $\boldsymbol{S} = \boldsymbol{I}_{2\times 2}$ in the code automatically.

b. Usually, the exact expression of displacement is unknown, and in this case, one needs to the specify the value or expression (in terms of x_1 and x_2) for the prescribed traction $\boldsymbol{t}$, displacement $\boldsymbol{g}$, body force $\boldsymbol{b}$, and the essential boundary switch $\boldsymbol{S}$ in the input files shown in Listing 4.

```
%% (3) Boundary condition
% If an edge is not specified, natural BC with zero traction is
imposed.
% displacement driven boundary conditions
Model.CriteriaEBC = @(x1,x2) find(x1 <= 0); % user input
Model.CriteriaNBC = @(x1,x2) find(x1>=5.3334E-2); % user input

% beta parameter for Nitches Method
Model.Beta_Nor = 1E2;

% give the expression of function handle of switch S, essential
% boundary conditions g, traction t, and body force b
if isfield(Model,'ExactSolution') % if given analytical displacement
    [Model.ExactSolution.S,Model.ExactSolution.g,...
     Model.ExactSolution.t,Model.ExactSolution.b] = ...
    getBoundaryConditions(Model);
    Model.ExactSolution.Exist = 1; % exist exact solution
else % if S, g, t, b are defined in functions
    Model.ExactSolution.S = @getSebc; % function getSebc
    Model.ExactSolution.g = @getGebc; % function getGebc
    Model.ExactSolution.t = @getTraction; % function getTraction
    Model.ExactSolution.b = @getBodyForce; % function getBodyForce
    Model.ExactSolution.Exist = 0; % no existence of exact solution
end
```

Listing 3. Defining boundary conditions and area in the input file.

```
%% Create Boundary Conditions and body forces
function [ t ] = getTraction(x1,x2,n1,n2)
% Input:
%   x1 x2: Cartesian coordinate
%   n1,n2: Normal vector at X on boundary
% Output:
%       t: a 2 by 1 vector for the traction
t = [20.69E9; 0];
end

function [ SEBC ] = getSebc(x1,x2)
% Input:
%  x1 x2: Cartesian coordinate
% Output:
%   SEBC: a 2 by 2 matrix for the switch matrix on EBC
SEBC = diag([1 1]);
End

function [ gEBC ] = getGebc(x1,x2)
%  Input:
%  x1 x2: Cartesian coordinate
% Output:
%   gEBC: a 2 by 1 vector of prescribed displacement on EBC
gEBC =[0; 0];
end

function [ b ] = getBodyForce(x1,x2)
%  Input:
%  x1 x2: Cartesian coordinate
% Output:
%      b: a 2 by 1 vector for the body force
b = [0; 0;];
end
```

Listing 4. Defining traction $\boldsymbol{t}$, body force $\boldsymbol{b}$, imposed displacement $\boldsymbol{g}$, and switch matrix $\boldsymbol{S}$ in the input file.

3.2.4 Setting up the discretization

RKPM2D provides four ways to discretize the domain defined in Section 3.2.2. We can edit section (4) Discretization Method shown in Listing 5 to specify the discretization method:

1. Define the `Model.Discretization.Method`, and associated parameters.
2. The illustration of each discretization and required parameters are listed below:
    a. `'A'`: MATLAB built-in mesh generator

i. `Hmax`: max discretized nodal distance, if `Hmax` is set to be zero, MATLAB will generate the discretization by a default nodal distance value.

b. `'B'`: Uniform/Non-uniform discretization for a rectangular domain

i. `nx1`: number of nodes in x_1 direction.

ii. `nx2`: number of nodes in x_2 direction.

iii. `Randomness`: random perturbation for nodal coordinates.

c. `'C'`: Shestakov distorted discretization for the plate with a hole problem in [42]

i. `nc`: denotes a refinement parameter (`nc > 1`).

ii. `Distortion`: distortion level between 0 ~ 0.5.

d. `'D'`: Read in a neutral file exported by other mesh generators (e.g. PATRAN, TRELIS, FEMAP, etc.)

i. `InputFileName`: The neutral file name (name in *.dat)

3. In general, it is recommended to use method `'A'`.

```
%% (4) Discretization Method
% For general purposes, one can always use A
Model.Discretization.Method = 'A';

% (...A) MATLAB built-in FE mesh generator: Default
% if A is chosen,
% define Hmax: max nodal distance for MATLAB built-in mesh generator
% if Hmax is <=0, then mesh is generated automatically
% by MATLAB built-in routine
% https://www.mathworks.com/help/pde/ug/pde.pdemodel.generatemesh.html
Model.Discretization.Hmax = 0.005;


% (...B) Uniform/Non-uniform discretization for rectangular domain:
% if A is chosen,
% define nx and ny: nx*ny is total number of nodes
% Randomness can be introduced to nodal distribution.
Model.Discretization.nx1 = 32;
Model.Discretization.nx2 = 8;
Model.Discretization.Randomness = 0.5;  % 0~1


% (...C) Shestakov distorted discretization for the plate problem:
% if C is chosen,
% define nc and randomness:
% nc>1 denotes a refinement parameter
% 0<Distortion<=0.5 is the distortion level parameter.
% This meshing option is specialized for "plate with a hole" geometry
% modifications are needed for other geometries
% as explained in the reference paper
Model.Discretization.nc = 3;             % >=1
Model.Discretization.Distortion = 0.1;   % 0~0.5

% (...D) FEA discretization by Trelis:
% if D is chosen,
% define the input file name from CAD/FEA model *.dat
% "please save the file in Patran format":
Model.Discretization.InputFileName =
'FE_Neutral_TensileTest_QuadMesh_Refined.dat';
```

Listing 5. Defining the domain discretization method in the input file.

3.2.5 Setting up RK shape function parameters

The RKPM2D provides several choices of basis, kernel geometry, kernel type, and normalized support size for the RK shape functions. These parameters are specified in section (5) RK shape function parameters shown in Listing 6:

1. Set up the kernel function type by defining `RK.KernelFunction`. The variable names for different kernel functions are listed in Table 1

Table 1. Abbreviation of the kernel function

Parameter	Continuity	Kernel Function
`HVSIDE`	C^{-1}	Heaviside
`SPLINE1`	C^{0}	Linear B-Spline (tent)
`SPLINE2`	C^{1}	Quadratic B-Spline
`SPLINE3`	C^{2}	Cubic B-Spline
`SPLINE4`	C^{3}	Quartic B-Spline
`SPLINE5`	C^{4}	Quintic B-Spline

2. Set up the kernel geometry `RK.KernelGeometry`, where two options are available:

Table 2. Abbreviation of the kernel geometry

Parameter	`CIR`	`REC`
Shape of Kernel	Circular	Rectangular

3. Set up the normalized support size `RK.NormalizedSupportSize`, where the support size is usually defined as $n + 1$, and n is the order of basis.
4. Set up the basis of the RK shape function `RK.Order`, where three basis are provided in the code as shown in Table 3. Linear order basis is recommended to use in conjunction with the SCNI-based nodal integration for general meshfree analysis.

Table 3. Abbreviation of basis

Parameter	`Constant`	`Linear`	`Quadratic`
Order of Basis	Constant	Linear	Quadratic

```
%% (5) RK shape function parameters
RK.KernelFunction = 'SPLIN3';        % SPLIN3
RK.KernelGeometry = 'CIR';           % CIR, REC
RK.NormalizedSupportSize = 2.01;     % suggested order n + 1;
RK.Order = 'Linear';                 % Constant, Linear, Quadratic
```

Listing 6. Defining RK shape function parameters in the input file

3.2.6 Setting up quadrature rules

RKPM2D provides different quadrature rules, which can be specified in section (6) Quadrature rule shown in Listing 7, as follows:

1. Set up the quadrature rule `Integration`, where three different integration rules are provided as shown in Table 4.

Table 4. The abbreviation of quadrature rule

Name	Represented integration method
`DNI`	Direct nodal integration
`SCNI`	Stabilized conforming nodal integration
`GAUSS`	Background Gauss integration

2. Set up the stabilization method for nodal integration, `Stabilization`, where two options for stabilization are available as shown in Table 5:

Table 5. The abbreviation of the stabilization for nodal integration

Name	Represented stabilization type
`N`	Naturally stabilized method [12]
`M`	Least squares stabilization method [36]
`WO`	Without any stabilization

3. Set up quadrature rule for boundary integration, `Option_BCintegration`, where one can choose `'NODAL'` for nodal integration or `'GAUSS'` for Gauss integration for the boundary integration.
4. For background Gauss integration, it is required to set up two additional parameters:
    a. `nGaussPoints`: the number of Gauss points $N_g \in \mathbb{N}$ in each background integration cell. E.g. $N_g = 6$ denotes 6×6 Gauss points in each cell
    b. `nGaussCells`: the parameter determines the number of background integration cells along x_1 or x_2 direction of the problem domain, depending on the problem domain dimension.

```
%% (6) Quadrature rule
Quadrature.Integration = 'SCNI';        % GAUSS, SCNI, DNI
Quadrature.Stabilization = 'N';         % M, N, WO
Quadrature.Option_BCintegration = 'NODAL'; % NODAL OR GAUSS
Quadrature.nGaussPoints = 6;
Quadrature.nGaussCells = 5;
% nGaussCells on the short side of the domain
```

Listing 7. Defining quadrature rules in the input file

3.2.7 Controlling the output

RKPM2D generates post-processing figures, based on the set-up in section (7) Control the output figures, where one can define each parameter shown in Table 6 to be 1 (on) or 0 (off).

Table 6. The abbreviation of outputs

Name	Represented output figures
`Discretization`	Discretization, nodal representative domains or background Gauss cells
`Displacement`	Displacement u_1 and u_2
`Strain`	Strain $[\varepsilon_{11}, \varepsilon_{22}, 2\varepsilon_{12}]$
`Stress`	Stress $[\sigma_{11}, \sigma_{22}, 2\sigma_{12}]$
`DeformedConfiguration`	Deformed configuration
`Error`	Absolute error in displacement and energy (active only when exact solution exists)

```
%% (7) Control the output figures
Model.Plot.Discretization = 1;
% plot discretization, nodal representative domains, or Gauss cells
Model.Plot.Displacement = 1;     % plot displacement
Model.Plot.Strain = 1;           % plot strain
Model.Plot.Stress = 1;           % plot stress
Model.Plot.DeformedConfiguration = 1;
% plot deformed configuration
Model.Plot.Error = 0;            % plot absolute error
```

Listing 8. Control the output fields in figures

3.3 Description of subroutines in RKPM2D

In addition to `getInput.m`, all subroutines for different functionalities of RKPM are included in the folder "MAIN_PROGRAM", as shown in Figure 8:

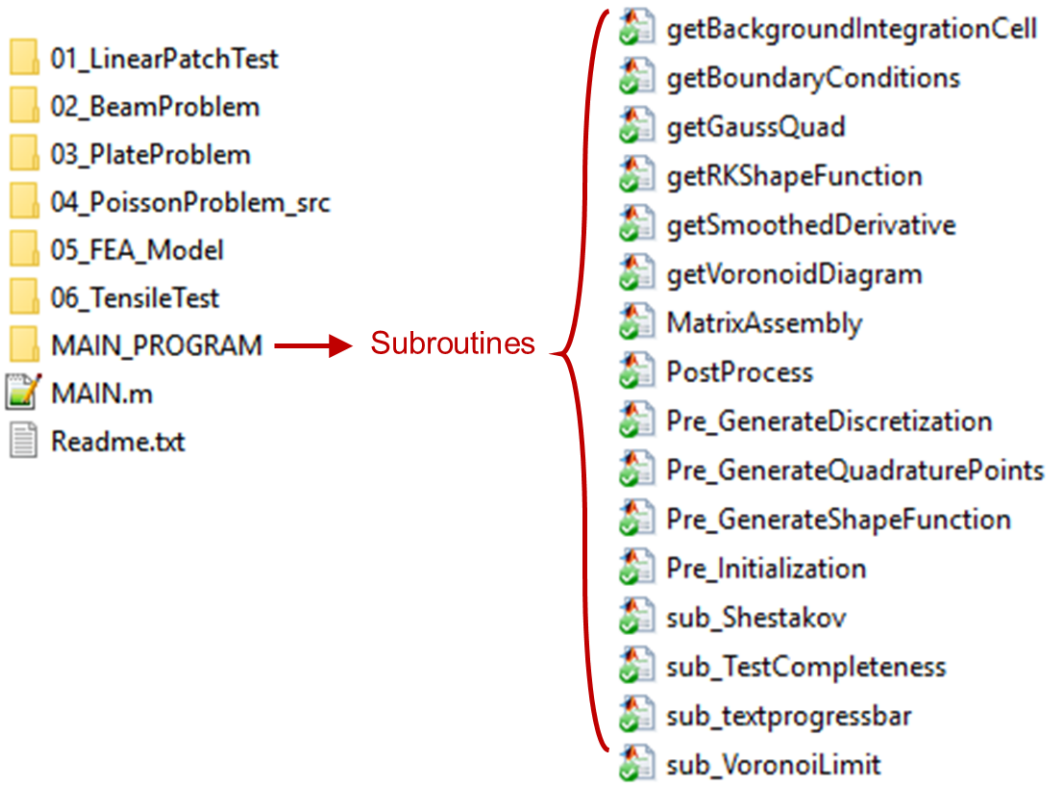

Figure 8. Subroutines for pre-processing, solver and post-processing.

The function of each subroutine is briefly described as follows, and users can check the reference [42] for more details of the code implementation:

`getBackgroundIntegrationCell.m`: This subroutine constructs the rectangular background Gauss integration cells if Gauss integration is adopted.

`getBoundaryConditions.m`: This subroutine outputs the functional of the prescribed boundary conditions $\boldsymbol{t} = \boldsymbol{\eta}^T \boldsymbol{C} \boldsymbol{\varepsilon}^{exact}$, $\boldsymbol{g} = \boldsymbol{u}^{exact}$, $\boldsymbol{S} = \boldsymbol{I}_{2\times 2}$ and the body force $\boldsymbol{b} = \boldsymbol{\sigma}_{ij,j}^{exact}$ according to given expressions of displacement $\boldsymbol{u}_{exact}$ in terms of x_1 and x_2 in a symbolic format.

`getGaussQuad.m`: This subroutine outputs Gaussian quadrature points and weights for any 1D domain. For example, `[Xgp,Wgp] = getGaussQuad(N,a,b)` gives with point location `Xgp` and weight `Wgp` of `N` Gauss points in a line segment with two ends `a` and `b`, where `a` and `b` denote the 1D coordinates of two end points.

`getRKShapeFunction.m`: This subroutine computes the 2D RK shape function and its direct gradients at a given evaluation point.

`getSmoothedDerivetive.m`: This subroutine computes SCNI smoothed gradients for a given nodal representative domain.

`getVoronoidDiagram.m`: This subroutine calls `sub_VoronoiLimit.m` to generate Voronoi cells and then re-arranges the Voronoi cell IDs and coordinates.

`MatrixAssembly.m`: This subroutine assembles the stiffness matrix and force vector.

`PostProcess.m`: This subroutine visualizes the calculated displacement, strain and stress fields.

`Pre_GenerateDiscratization.m`: This subroutine generates point-based domain discretization based on domain vertices' coordinates and the chosen discretization method.

`Pre_GenerateQuadraturePoint.m`: This subroutine computes the quadrature points for domain integration and boundary integration for nodal integration or Gauss integration.

`Pre_GenerateShapeFunction.m`: This subroutine generates the RK shape functions and gradients at quadrature points.

`Pre_Initialization.m`: This subroutine initializes the matrices and vectors for the simulation. It also displays the input information in the MATAB command window.

`sub_Shestakov.m`: This subroutine generates highly distorted meshes using Shestakov's algorithm [43] for a rectangular domain. This function is called in `Pre_GenerateDiscratization.m` when discretization method 'D' is used.

`sub_TestCompleteness.m`: This subroutine tests the reproducing condition of RK shape functions and gradients.

`sub_textprogressbar.m`: This subroutine generates the simulation progress bar in MATLAB command window.

`sub_VoronoiLimit.m`: This subroutine calls a library that computes vertices' coordinates of Voronoi cells and indices of vertices within each Voronoi cell.

3.4 Executing RKPM2D

After the input file `getInput.m` is prepared, one can execute RKPM2D by running `MAIN.m` through the following steps:

1. Open `MAIN.m`. shown in Figure 9
2. Press "Run" to execute the simulation in MALTAB.
3. Once the simulation is finished, the post-processing figures will be generated, and one can perform further analysis of the results.

A successful execution of the meshfree analysis will show basic information for the simulation, as shown in Figure 10.

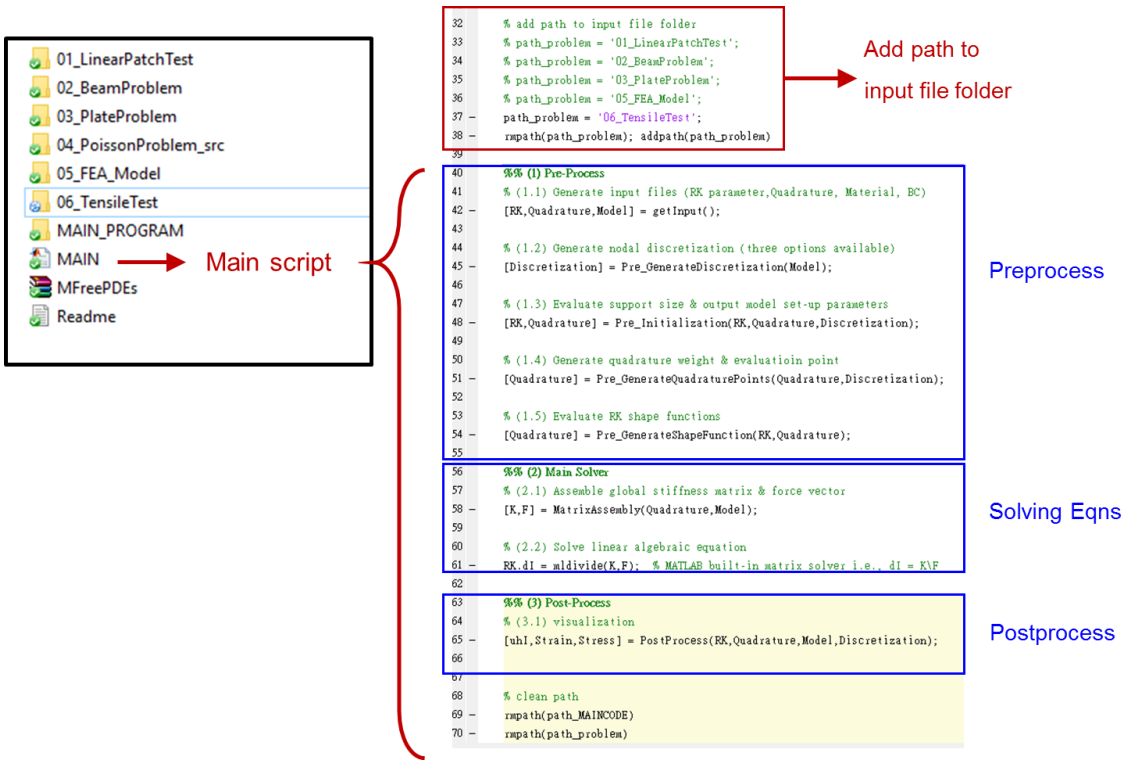

Figure 9. The main script (MAIN.m) for running meshfree analysis.

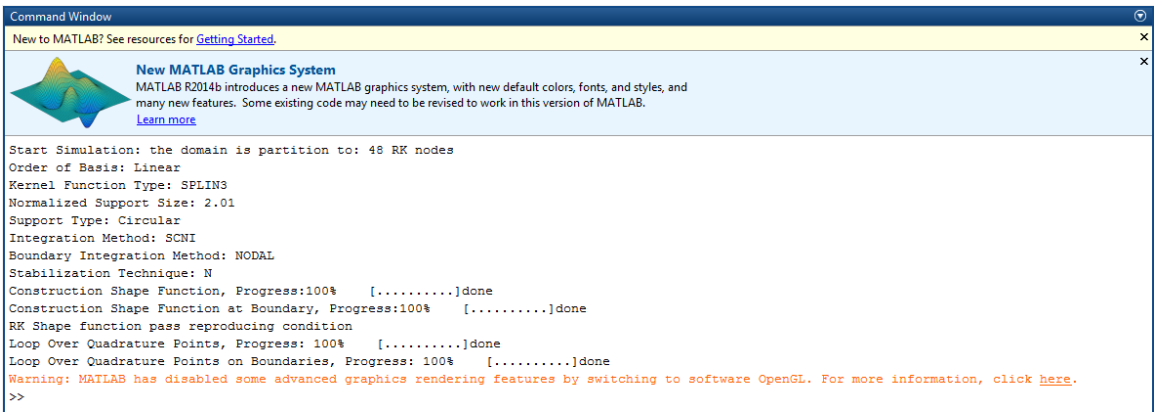

Figure 10. MATLAB Command Windows showing the progress of simulation.

3.5 Post-processing and analysis

When the simulation is finished, key parameters and data structures associated with the results are saved in the workspace of MATLAB, as shown in Figure 11 where the data are mainly included in the structures `RK`, `Quadrature`, and `Model`.

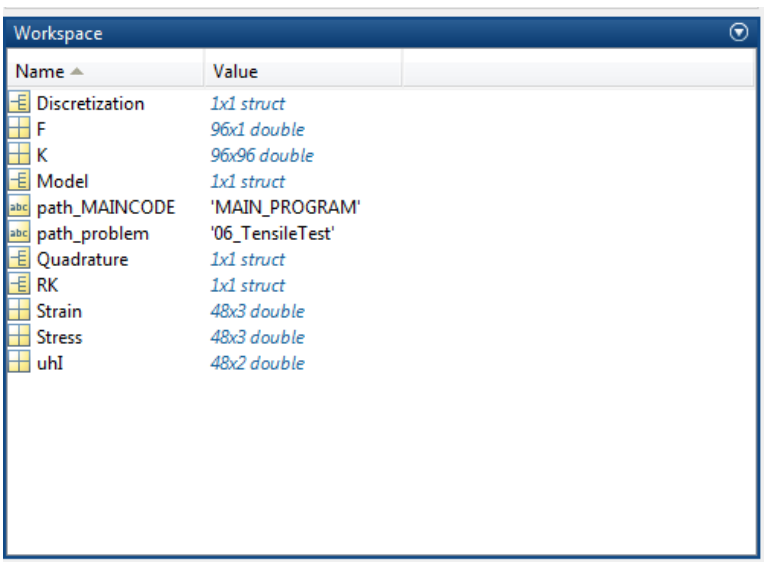

Figure 11. Data structure and parameters saved in the MATLAB Workspace area.

A brief description of structures and important sub-classes are given as follows:

`RK` contains the following fields:

- `KernelFunction`: kernel functions with different levels of continuity.
- `KernelGeometry`: nodal support shape.
- `NormalizedSupportSize`: normalized support size.
- `Order`: order of basis.
- `xI`: coordinates of RK nodes
- `nP`: number of RK nodes
- `dI`: solution coefficient $\boldsymbol{u}_I$

`Quadrature` contains the following fields:

- `Integration`: quadrature rules.
- `Stabilization`: type of stabilization technique for nodal integrations.
- `Option_BCintegration`: quadrature rule for boundary integrals.
- `Domain`: class that defines the variables required for domain integration.
  - `nQuad`: total number of quadrature points.
  - `xQuad`: coordinates of quadrature points.
  - `Weight`: quadrature weights for domain integral.
- `BC`: class that defines the variables required for boundary integration.
  - `nQuad_onBoundary`: number of quadrature points on the boundary.
  - `xQuad_onBoundary`: coordinates of quadrature points on the boundary.
  - `Weight_onBoundary`: quadrature weights for contour integral.
  - `Normal_onBoundary`: outward unit normal vectors at quadrature points
- `VoronoiDiagram`: structure defines the quadrature rules required for nodal integration.
  - `VerticeCoordinates`: all vertices' coordinates of Voronoi cells.
  - `VoronoiCell`: indexes of vertices within each Voronoi cell.
- `SHP`: matrix of size `nQuad`×`nP` that stores the shape functions $\Psi_I(\boldsymbol{x}_N)$ of all nodes.
- `nQuad`: number of quadrature points for domain integration, the value $= NG$ for Gauss integration and $= NP$ for nodal integration
- `SHPDX1`: matrix of size `nQuad`×`nP` that stores the shape function derivatives $\Psi_{I,1}(\boldsymbol{x}_N)$ or $\widetilde{\Psi}_{I,1}(\boldsymbol{x}_N)$ of all nodes.
- `SHPDX2`: matrix of size `nQuad`×`nP` that stores the shape function derivatives $\Psi_{I,2}(\boldsymbol{x}_N)$ or $\widetilde{\Psi}_{I,2}(\boldsymbol{x}_N)$ of all nodes.

`Model` contains the following fields:

- `E`, `nu`: Young's modulus E, Poisson ratio ν.
- `DOFu`: number of nodal degrees of freedom, `DOFu=2` for 2D elasticity.
- `Beta_Nor`: normalized penalty parameter.
- `xVertices`: physical coordinates of problem domain.
- `CriteriaEBC`: function handle to define the essential boundaries.
- `CriteriaNBC`: function handle to define the natural boundaries.
- `Discretization`: domain discretization method.
- `ExactSolution`: function handle to define the essential boundaries.
  - `t`: functional handle returns the traction $\boldsymbol{t}$.
  - `b`: functional handle returns the body force vector $\boldsymbol{b}$.
  - `g`: functional handle returns the imposed displacement $\boldsymbol{g}$.
  - `S`: functional handle returns the switch matrix $\boldsymbol{S}$.

In addition to the `RK`, `Quadrature`, and `Model`, other variables shown in Figure 11 are saved as

- `K`: global total stiffness matrix after assembly, Eq. (13)
- `F`: global total force vector after assembly, Eq. (14)
- `uhI`: matrix of size $NP \times 2$ that defines the physical displacements at RK nodes $\boldsymbol{u}^h(\boldsymbol{x}_I)$.
- `Strain`: double vector of size $NP \times 3$ that defines the strain at RK nodes $\boldsymbol{\varepsilon}(\boldsymbol{x}_I) = [\varepsilon_{11}(\boldsymbol{x}_I), \varepsilon_{22}(\boldsymbol{x}_I), 2\varepsilon_{12}(\boldsymbol{x}_I)]$.
- `Stress`: double vector of size $NP \times 3$ that defines the stress at RK nodes $\boldsymbol{\sigma}(\boldsymbol{x}_I) = [\sigma_{11}(\boldsymbol{x}_I), \sigma_{22}(\boldsymbol{x}_I), 2\sigma_{12}(\boldsymbol{x}_I)]$.

One can make use of the abovementioned information to perform further analysis of the simulation results, such as error analysis, convergence studies, etc.

4 Numerical Examples

In this section, numerical examples and corresponding input files are provided, which can be used as the starting point for the users to solve more complicated problems. The reproducing kernel approximation with linear basis and cubic B-spline kernel is adopted, for which circular support with a normalized support size $\tilde{c} = 2.0$ is used. The normalized penalty parameter for Nitsche's method is chosen as $\beta_{nor} = 100$. The abbreviation for domain integration methods are given in Table 7:

Table 7. The abbreviations for domain integration with different stabilization

	Naturally stabilized `N`	Least-squared-based stabilization `M`	No stabilization `WO`
`DNI`	`NDNI`	`MDNI`	`DNI`
`SCNI`	`NSCNI`	`MSCNI`	`SCNI`
`GAUSS`	`Not applicable`	`Not applicable`	`GI`

4.1 Plotting RK shape function in 1D/2D

In the first example, two scripts (Filename: PlotRKShapeFunction1D and PlotRKShapeFunction2D shown in Figure 12) are provided to plot 1D and 2D RK shape functions, where the subroutines `getRKShapeFunction` and the corresponding 1D version `getRKShapeFunction1D` are adopted, respectively.

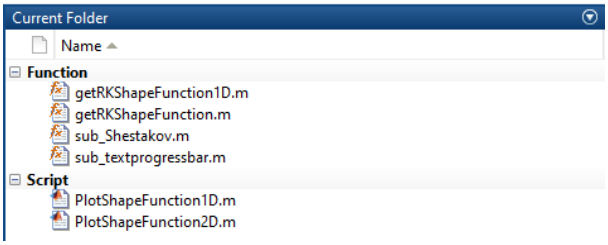

Figure 12. MATLAB script for plotting RK shape functions in 1D and 2D.

4.1.1 Plotting the RK shape function in 1D

One can follow the following steps to plot the RK shape functions in 1D:

1. Open the folder PlotRKShapeFunction.
2. Open the MATLAB script PlotRKShapeFunction1D.m
3. Specify the RK parameters, nodal distribution, and output figure type

The definition of RK parameters are identical to the discussions given in section 3.2.5, and the variable `output_figure` specifies the types of output figures, and `randomness_number` imposes the magnitude of nodal coordinate perturbation if one wants to test non-uniform nodal discretizations.

```
%% RKPM Setting Area
RK.KernelFunction = 'SPLIN3';        % SPLIN1-5, HVSIDE
RK.KernelGeometry = 'CIR';           % CIR, REC
RK.Order = 'Linear';                 % Constant, Linear, Quadratic
RK.nP = 11;                          % # of nodes

% Support Size
NormalzieSupportSize1 = 1.5;

% Choose the output
output_figure = 'Shape_Function';
% Shape_Function, RK_condition, RK_condition_error, Support_Comparison

%% Discretize RK nodes
xmin=0; xmax= 1;
nP = RK.nP;
L = xmax-xmin;

% randomness
randomness_number = 0.5;
```

Listing 9. RK parameters and nodal distribution set up in PlotRKShapeFunction1D.m

Running the script `PlotRKShapeFunction1D.m`, one can observe how the kernel function controls the smoothness of the approximation, as shown in Figure 13, where the C^0 tent kernel function is compared with the C^2 cubic B-spline kernel function.

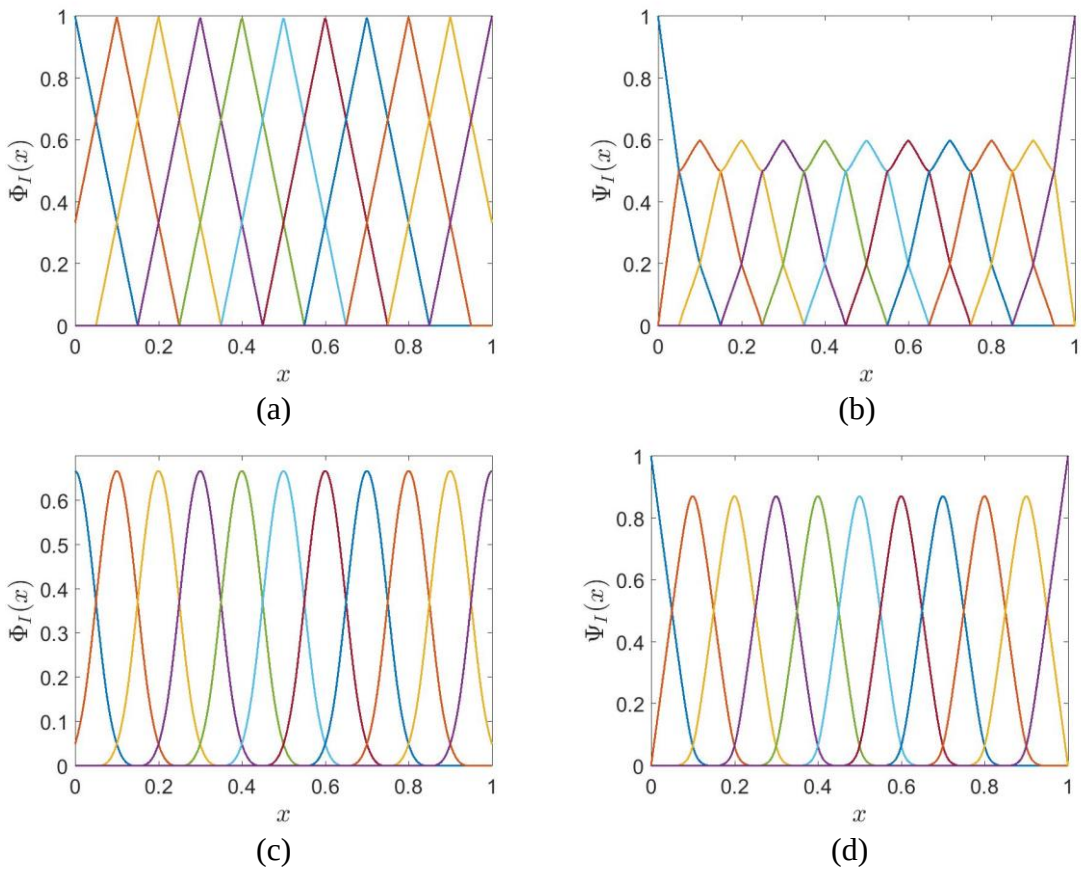

Figure 13. Kernel function and corresponding RK shape function with linear basis and support size $\tilde{c} = 1.5$: (a) Tent kernel and (b) corresponding RK shape function, (c) Cubic B-Spline kernel and (d) corresponding RK shape function.

4.1.2 Plotting the RK shape function in 2D

One can follow the following steps to plot the RK shape functions in 2D:

1. Open the folder PlotRKShapeFunction.
2. Open the MATLAB script PlotRKShapeFunction2D.m
3. Specify the RK parameters, nodal distribution, and output figure type

The definition of RK parameters are identical to the discussion given in section 3.2.5, and the variable `output_figure` determines the types of output figures. In this example, the nodal distributions are generated by the Shestakov meshing method discussed in section 3.2.4 where `nc` determines the refinement level and `randomness` determines

the level of distortion. With the RK nodes uniformly distributed, the RK shape function with circular cubic B-spline kernels is plotted in Figure 14.

```
%% Define RK Shape Function
RK.KernelFunction = 'SPLIN3';        % HVSIDE,SPLIN1-5
RK.KernelGeometry = 'CIR';           % CIR, REC
RK.NormalizedSupportSize = 1.5;      %
RK.Order = 'Linear';                 % Constant, Linear, Quadratic

%% define discrete RK nodes by Shestakov mesh
% we use the test distorted mesh generator to test
nc = 2; % the total number of nodes will be (2^nc+1)*(2^nc+1)
randomness = 0.1; % 0~0.5, where 0.5 is uniform case

% Choose the output
output_figure = 'Shape_Function'; % Shape_Function, RK_condition,
RK_condition_error
```

Listing 10. Set-up of RK parameters and nodal distribution in PlotRKShapeFunction2D.m

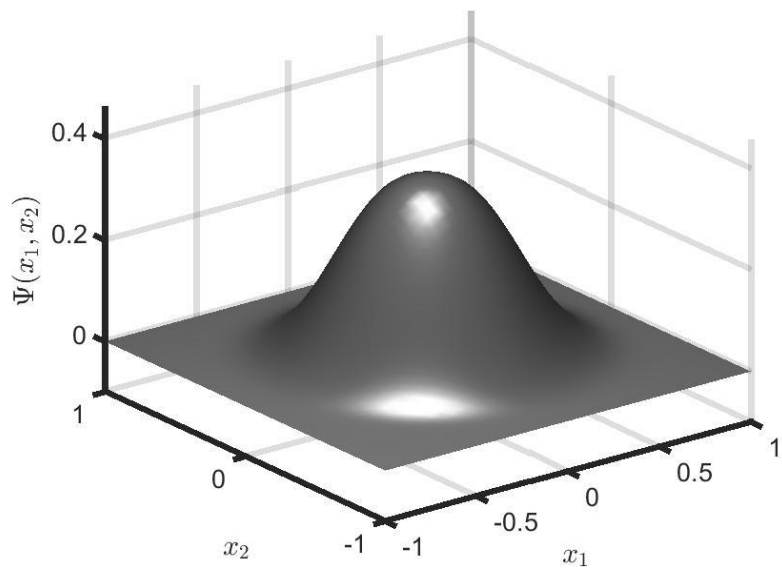

Figure 14. The RK shape function in the two-dimensional case with cubic B-spline and circular kernel.

4.2 Patch test

In the second example, the linear patch test is provided to verify the accuracy of RKPM2D using linear basis in the RK approximation. The elasticity equation in (9) is considered with the exact solution defined as a linear polynomial function:

$$\boldsymbol{u}^{exact} = \begin{bmatrix} 0.1 + 0.1x_1 + 0.2x_2 \\ 0.05 + 0.15x_1 + 0.1x_2 \end{bmatrix} \tag{34}$$

Accordingly, the traction $\boldsymbol{t} = \boldsymbol{\eta}^T \boldsymbol{C}\boldsymbol{\varepsilon}^{exact}$ is imposed on $\partial\Omega_t: (x_1, x_2) \in \partial\Omega, x_2 > 0.5$, where $\boldsymbol{\eta}$ is the collection of outward unit normal vector of the boundary surface, $\boldsymbol{C}$ is the matrix of elastic moduli with Young's modulus $E = 2.1 \times 10^{11}$ and Poisson's ratio $\nu = 0.3$, $\boldsymbol{\varepsilon}^{exact}$ is the exact strain $\boldsymbol{\varepsilon}^{exact} = [0.1,\ 0.1,\ 0.35]^T$; $\boldsymbol{g} = \boldsymbol{u}^{exact}$ is enforced on $\partial\Omega_g: (x_1, x_2) \in \partial\Omega, x_2 \leq 0.5$, and the body force is $\boldsymbol{b} = \boldsymbol{0}$. A square domain is considered, and its nodal discretization is shown in Figure 15.

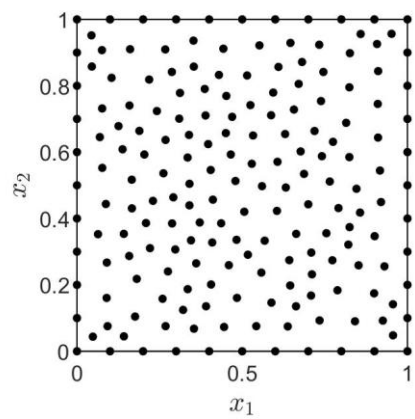

Figure 15. A square domain considered in the patch test.

The input file for solving the linear patch test is provided in the folder "01_LinearPatchTest". For demonstration purposes, we use discretization method `'A'` (MATLAB Built-In Mesh Generator). NSCNI is used as the quadrature rule, and circular, cubic B-Spline kernel (normalized support size 2.01) is used for the RK shape function construction. Detailed information for setting up the model can be referred to Listing 11. Since the exact solution is known, the boundary conditions $\boldsymbol{t}, \boldsymbol{g}, \boldsymbol{S}$ and body force $\boldsymbol{b}$ will be automatically computed according to the given exact solution (Eq. (34)).

```
function [RK,Quadrature,Model] = getInput()
%% INPUT FILE
% Sample Input File for Patch Test
%% (1) Material
% Linear Elasticity
% Lame Parameters for Young's modulus and Poisson ratio
Model.E = 2.1E11; Model.nu = 0.3;
Model.Condition = 'PlaneStress'; % PlaneStress, or PlaneStrain
Model.ElasticTensor =
getElasticTensor(Model.E,Model.nu,Model.Condition);
Model.DOFu = 2;                  % two dimensional problem
%% (2) Geometry
x1_vertices = [0 1 1 0]';
x2_vertices = [0 0 1 1]';
% ensure the boundary segments to be counter clockwise
[x1_vertices, x2_vertices] = poly2ccw(x1_vertices, x2_vertices);
Model.xVertices = [x1_vertices, x2_vertices];
Model.DomainArea = polyarea(x1_vertices,x2_vertices);
%% (3) Boundary condition
% If an edge is not specified, natural BC with zero traction is
imposed.
Model.CriteriaEBC = @(x1,x2) find(x2<=0.5); % user input
Model.CriteriaNBC = @(x1,x2) find(x2>0.5); % user input
% beta parameter for Nitches Method
Model.Beta_Nor = 1E2;

% For verification purpose, provide the exact displacement solution
syms x1 x2        % use x1 and x2 as x- & y- coordinates
% exact solution
Model.ExactSolution.u_exact =[0.1 + 0.1*x1 + 0.2*x2;
                              0.05 + 0.15*x1 + 0.1*x2;];
[Model.ExactSolution.S,...
 Model.ExactSolution.g,...
 Model.ExactSolution.t,...
 Model.ExactSolution.b] = getBoundaryConditions(Model);
 Model.ExactSolution.Exist = 1;

%% (4) Discretization Method
% For general purpose, one can always use A
Model.Discretization.Method = 'A';
Model.Discretization.Hmax = 0;
%% (5) RK shape function parameter
RK.KernelFunction = 'SPLIN3';       % SPLIN3
RK.KernelGeometry = 'CIR';          % CIR, REC
RK.NormalizedSupportSize = 2.01;    %
RK.Order = 'Linear';                % Constant, Linear, Quadratic
%% (6) Quadrature rule
Quadrature.Integration = 'SCNI';       % GAUSS, SCNI, DNI
Quadrature.Stabilization = 'N';        % M, N
Quadrature.Option_BCintegration = 'NODAL'; % NODAL OR GAUSS
Quadrature.nGaussPoints = 6;
Quadrature.nGaussCells = 10; % nGaussCells on the short side of the
domain
end
```

Listing 11. Input file for solving a linear patch test with a square domain.

Once the numerical results were obtained, it can be verified that NSCNI passes patch tests by checking the displacement and strain energy errors shown in Figure 16.

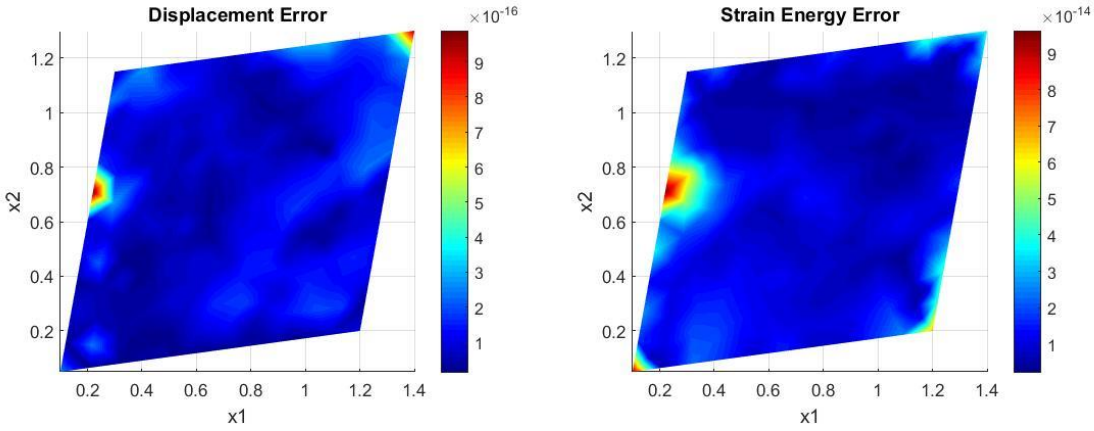

Figure 16. Absolute displacement and strain energy error.

4.3 Cantilever beam problem

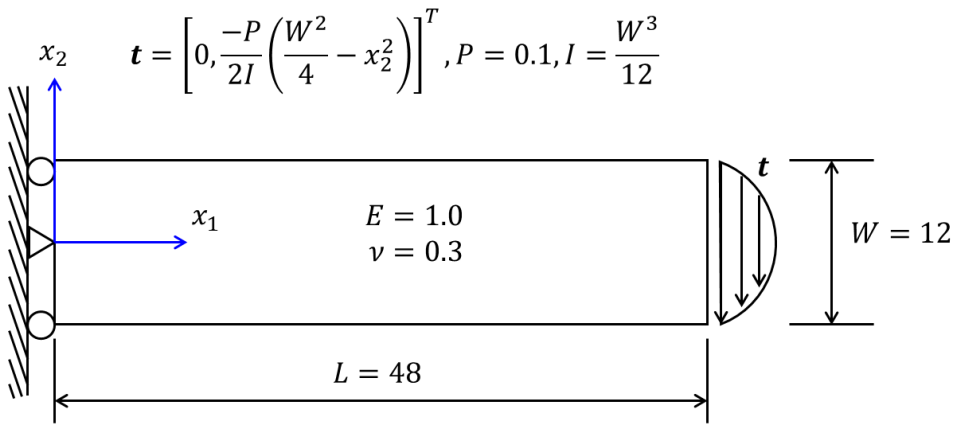

Figure 17. Problem setting of a cantilever beam under a shear force.

The last example shown here is the cantilever beam problem shown in Figure 17, where the exact solutions are:

$$
\begin{aligned}
u_1^{exact} &= \frac{Px_2}{6EI}\left[(6L - 3x_2)x_1 + (2+\nu)\left(x_2^2 - \frac{W^2}{4}\right)\right], \\
u_2^{exact} &= \frac{-P}{6EI}\left[3\nu x_2^2(L - x_1) + (4+5\nu)\frac{W^2 x_1}{4} + (3L - x_1)x_1^2\right]
\end{aligned}
\tag{35}
$$

in which the Young's modulus $E = 1$, Poisson ratio $\nu = 0.3$, and the geometry and loading is plotted in Figure 17. The exact displacement solution is prescribed on the left wall as the essential boundary condition, i.e., $\boldsymbol{g} = \boldsymbol{u}^{exact}$, and the traction is imposed on the right-side surface as the natural boundary condition.

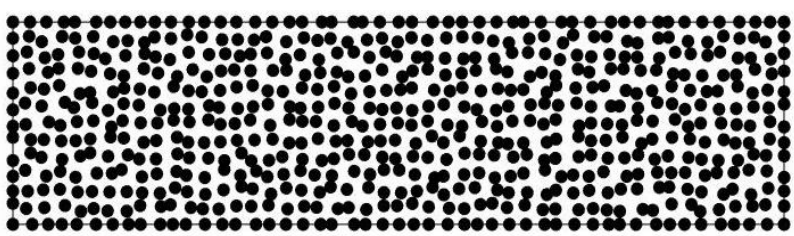

Figure 18. Non-uniform discretization with 49×13 nodes for the cantilever beam problem, where the discretization consists of a randomized nodal distribution generated from a uniform nodal distribution.

A non-uniform nodal discretization of the beam is shown in Figure 18. The input file is provided in the folder "02_BeamProblem". For demonstration purpose, we use discretization method `'B'` (rectangular domain) with randomness number 0.5. Cubic B-Spline kernel with circular support (normalized support size 2.01) is used for the RK shape function construction. Detailed information for setting up the model can be referred to Listing 12. Users are recommended to use different quadrature rules to solve this problem to examine their performance. As shown in Figure 19, both MSCNI and NSCNI perform well comparing with the exact solution, whereas DNI demonstrates spurious oscillations in the stress field.

```
function [RK,Quadrature,Model] = getInput()
% Sample Input File for Beam Problem
%% (1) Material
% Lame Parameters for Young's modulus and Poisson ratio
Model.E = 1.0E0; Model.nu = 0.3;  % user input
Model.Condition = 'PlaneStress';  % 'PlaneStress' or 'PlaneStrain'
Model.ElasticTensor =
getElasticTensor(Model.E,Model.nu,Model.Condition);
Model.DOFu = 2;                   % two dimensional problem
%% (2) Geometry
% rectangular polygon
x1_vertices = [0 48 48 0]';  % user input
x2_vertices = [-6 -6 6 6]';  % user input
% ensure the boundary segments to be counter clockwise
[x1_vertices, x2_vertices] = poly2ccw(x1_vertices, x2_vertices);
Model.xVertices = [x1_vertices, x2_vertices];
Model.DomainArea = polyarea(x1_vertices,x2_vertices);
%% (3) Boundary condition
Model.CriteriaEBC = @(x1,x2) find(x1<=0+1E-7);    % user input
Model.CriteriaNBC = @(x1,x2) find(x1>=48-1E-7);   % user input
% beta parameter for Nitches Method
Model.Beta_Nor = 1E2;
% For verification purpose, provide the exact displacement solution
syms x1 x2        % use x1 and x2 as x- & y- coordinates
H = 12; L = 48; D = H; trac = 0.1;   % user input
I_inertia = (H^3)/12;    % user input
E = Model.E; nu = Model.nu;
% exact solution exist, so use it
u1 = (trac*x2/(6*E*I_inertia)).*((6*L-3*x1).*x1+(2+nu)*(x2.^2-
(D^2)/4));
u2 = -(trac/(6*E*I_inertia)).*(3*nu*x2.^2.*(L-
x1)+(4+5*nu).*((D^2*x1)/4)+(3*L-x1).*x1.^2);
Model.ExactSolution.u_exact =[u1;u2;];
[Model.ExactSolution.S,Model.ExactSolution.g,...
 Model.ExactSolution.t,Model.ExactSolution.b] =
getBoundaryConditions(Model);
Model.ExactSolution.Exist = 1;
%% (4) Discretization Method
Model.Discretization.Method = 'B';
Model.Discretization.nx1 = 32;
Model.Discretization.nx2 = 8;
Model.Discretization.Randomness = 0.5;  % 0~1
%% (5) RK shape function parameter
RK.KernelFunction = 'SPLIN3';       % SPLIN3
RK.KernelGeometry = 'CIR';          % CIR, REC
RK.NormalizedSupportSize = 2.01;    % suggested order n + 1;
RK.Order = 'Linear';                % Constant, Linear, Quadratic
%% (6) Quadrature rule
Quadrature.Integration = 'SCNI';    % GAUSS, SCNI, DNI
Quadrature.Stabilization = 'N';     % M, N
Quadrature.Option_BCintegration = 'NODAL'; % NODAL OR GAUSS
Quadrature.nGaussPoints = 6;
Quadrature.nGaussCells = 10;
end
```

Listing 12. Sample input file for solving cantilever beam problem.

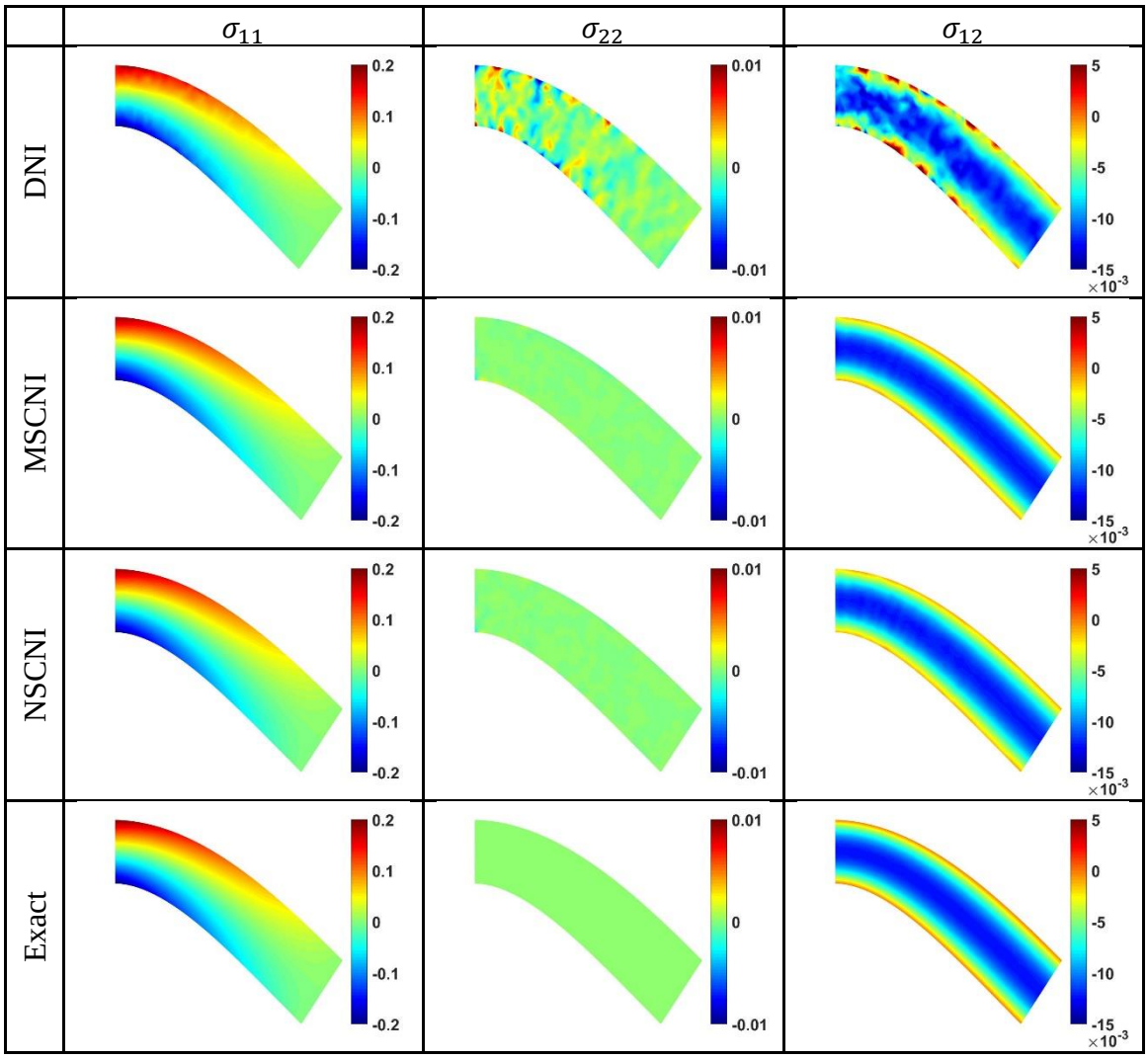

Figure 19. Stress fields of the cantilever beam problem under non-uniform discretization.

References

[1] J.-S. Chen, C. Pan, C.-T. Wu and W. K. Liu, "Reproducing kernel particle methods for large deformation analysis of non-linear structures," *Computer Methods in Applied Mechanics and Engineering,* vol. 139, no. 1-4, pp. 195-227, 1996.

[2] J.-S. Chen, M. Hillman and S.-W. Chi, "Meshfree methods: progress made after 20 years," *Journal of Engineering Mechanics,* vol. 143, no. 4, p. 04017001, 2017.

[3] W. K. Liu, S. Jun and Y. F. Zhang, "Reproducing kernel particle methods," *International Journal for Numerical Methods in Fluids,* vol. 20, no. 8-9, pp. 1081-1106, 1995.

[4] W. K. Liu, S. Hao, T. Belytschko, S. Li and C. T. Chang, "Multiple scale meshfree methods for damage fracture and localization," *Computational materials science,* vol. 16, no. 1-4, pp. 197-205, 1999.

[5] S. Li, W. Hao and W. K. Liu, "Mesh-free simulations of shear banding in large deformation," *International Journal of solids and structures,* vol. 37, no. 48-50, pp. 7185-7206, 2000.

[6] W. K. Liu, S. Jun, D. T. Sihling, Y. Chen and W. Hao, "Multiresolution reproducing kernel particle method for computational fluid dynamics," *International Journal for Numerical Methods in Fluids,* vol. 24, no. 12, pp. 1391--1415, 1997.

[7] W. K. Liu and Y. Chen, "Wavelet and multiple scale reproducing kernel methods," *International Journal for Numerical Methods in Fluids,* vol. 21, no. 10, pp. 901-931, 1995.

[8] W. K. Liu, Y. Chen, R. A. Uras and C. T. Chang, "Generalized multiple scale reproducing kernel particle methods," *Computer Methods in Applied Mechanics and Engineering,* vol. 139, no. 1-4, pp. 91-157, 1996.

[9] M. O. Ruter and J.-S. Chen, "An enhanced-strain error estimator for Galerkin meshfree methods based on stabilized conforming nodal integration," *Computers & Mathematics with Applications,* vol. 74, no. 9, pp. 2144-2171, 2017.

[10] Y. You, J.-S. Chen and H. Lu, "Filters, reproducing kernel, and adaptive meshfree method," *Computational Mechanics,* vol. 31, no. 3-4, pp. 316-326, 2003.

[11] J.-S. Chen, M. Hillman and M. Ruter, "An arbitrary order variationally consistent integration for Galerkin meshfree methods," *International Journal for Numerical Methods in Engineering,* vol. 95, no. 5, pp. 387-418, 2013.

[12] M. Hillman and J.-S. Chen, "An accelerated, convergent, and stable nodal

integration in Galerkin meshfree methods for linear and nonlinear mechanics," *International Journal for Numerical Methods in Engineering,* vol. 107, no. 7, pp. 603-630, 2016.

[13] J.-S. Chen and D. Wang, "A constrained reproducing kernel particle formulation for shear deformable shell in Cartesian coordinates," *International Journal for Numerical Methods in Engineering,* vol. 68, no. 2, pp. 151-172, 2006.

[14] N. H. Kim, K. K. Choi, J.-S. Chen and M. E. Botkin, "Meshfree analysis and design sensitivity analysis for shell structures," *International Journal for Numerical Methods in Engineering,* vol. 53, no. 9, pp. 2087-2116, 2002.

[15] J.-S. Chen, C. Pan, C. Roque and H.-P. Wang, "A Lagrangian reproducing kernel particle method for metal forming analysis," *Computational Mechanics,* vol. 22, no. 3, pp. 289-307, 1998.

[16] J.-S. Chen, H.-P. Wang, S. Yoon and Y. You, "Some recent improvements in meshfree methods for incompressible finite elasticity boundary value problems with contact," *Computational Mechanics,* vol. 25, no. 2-3, pp. 137-156, 2000.

[17] H.-P. Wang, C.-T. Wu and J.-S. Chen, "A reproducing kernel smooth contact formulation for metal forming simulations," *Computational Mechanics,* vol. 54, no. 1, pp. 151-169, 2014.

[18] J.-S. Chen, R. R. Basava, Y. Zhang, R. Csapo, V. Malis, U. Sinha, J. Hodgson and S. Sinha, "Pixel-based meshfree modelling of skeletal muscles," *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization,* vol. 4, no. 2, pp. 73-85, 2016.

[19] H. Wei, J.-S. Chen and M. Hillman, "A stabilized nodally integrated meshfree formulation for fully coupled hydro-mechanical analysis of fluid-saturated porous media," *Computers & Fluids,* vol. 141, pp. 105-115, 2016.

[20] H. Wei, J.-S. Chen, F. Beckwith and J. Baek, "A naturally stabilized semi-Lagrangian meshfree formulation for multiphase porous media with application to landslide modeling," *Journal of Engineering Mechanics,* vol. under review, 2018.

[21] J.-S. Chen, C.-T. Wu and T. Belytschko, "Regularization of material instabilities by meshfree approximations with intrinsic length scales," *International Journal for Numerical Methods in Engineering,* vol. 47, no. 7, pp. 1303-1322, 2000.

[22] J.-S. Chen, X. Zhang and T. Belytschko, "An implicit gradient model by a reproducing kernel strain regularization in strain localization problems," *Computer Methods in Applied Mechanics and Engineering,* vol. 193, no. 27-29, pp. 2827-2844, 2004.

[23] H. Wei and J.-S. Chen, "A damage particle method for smeared modeling of brittle fracture," *International Journal for Multiscale Computational Engineering,* vol. 16, no. 4, 2018.

[24] M. J. Roth, J.-S. Chen, T. R. Slawson and K. T. Danielson, "Stable and flux-conserved meshfree formulation to model shocks," *Computational Mechanics,* vol. 57, no. 5, pp. 773-792, 2016.

[25] M. J. Roth, J.-S. Chen, K. T. Danielson and T. R. Slawson, "Hydrodynamic meshfree method for high-rate solid dynamics using a Rankine--Hugoniot enhancement in a Riemann-SCNI framework," *International Journal for Numerical Methods in Engineering,* vol. 108, no. 12, pp. 1525-1549, 2016.

[26] P.-C. Guan, S.-W. Chi, J.-S. Chen, T. Slawson and M. J. Roth, "Semi-Lagrangian reproducing kernel particle method for fragment-impact problems," *International Journal of Impact Engineering,* vol. 38, no. 12, pp. 1033-1047, 2011.

[27] J. A. Sherburn, M. J. Roth, J. Chen and M. Hillman, "Meshfree modeling of concrete slab perforation using a reproducing kernel particle impact and penetration formulation," *International Journal of Impact Engineering,* vol. 86, pp. 96-110, 2015.

[28] S.-W. Chi, C.-H. Lee, J.-S. Chen and P.-C. Guan, "A level set enhanced natural kernel contact algorithm for impact and penetration modeling," *International Journal for Numerical Methods in Engineering,* vol. 102, no. 34, pp. 839-866, 2015.

[29] J.-S. Chen, W. K. Liu, M. Hillman, S.-W. Chi, Y. Lian and M. Bessa, "Reproducing Kernel Particle Method for Solving Partial Differential Equations," *Encyclopedia of Computational Mechanics, Second Edition,* pp. 1-44, 2017.

[30] MathWorks, *MATLAB,* Natick, Massachusetts: The MathWorks Inc., 1984.

[31] J. Nitsche, "Uber ein Variationsprinzip zur Losung von Dirichlet-Problemen bei Verwendung von Teilraumen, die keinen Randbedingungen unterworfen sind.," *Abh. Math. Sem. Univ. Hamburg,* vol. 36, pp. 9-15, 1970-1971.

[32] S. Fernandez-Mendez and A. Huerta, "Imposing essential boundary conditions in mesh-free methods," *Computer Methods in Applied Mechanics and Engineering,* vol. 193, no. 12-14, pp. 1257-1275, 2004.

[33] E. Hardee, K.-H. Chang, S. Yoon, M. Kaneko, I. Grindeanu and J.-S. Chen, "A Structural Nonlinear Analysis Workspace (SNAW) based on meshless methods," *Advances in Engineering Software,* vol. 30, no. 30, pp. 153-175, 1999.

[34] Y.-M. Hsieh and M.-S. Pan, "ESFM: An essential software framework for meshfree methods," *Advances in Engineering Software,* vol. 76, pp. 133-147, 2014.

[35] J.-S. Chen, C.-T. Wu, S. Yoon and Y. You, "A stabilized conforming nodal integration for Galerkin mesh-free methods," *International Journal for Numerical Methods in Engineering,* vol. 50, no. 2, pp. 435-466, 2001.

[36] J.-S. Chen, W. Hu, M. Puso, Y. Wu and X. Zhang, "Strain smoothing for stabilization and regularization of Galerkin meshfree methods," in *Meshfree Methods for Partial Differential Equations III*, Springer, 2007, pp. 57-75.

[37] S. Li and W. K. Liu, Meshfree particle methods, Berlin: Springer Science & Business Media, 2007.

[38] J.-S. Chen and H.-P. Wang, "New boundary condition treatments in meshfree computation of contact problems," *Computer Methods in Applied Mechanics and Engineering,* vol. 187, no. 3-4, pp. 441-468, 2000.

[39] J. Dolbow and T. Belytschko, "Numerical integration of the Galerkin weak form in meshfree methods," *Computational Mechanics,* vol. 23, no. 3, pp. 219-230, 1999.

[40] Y. Lu, T. Belytschko and L. Gu, "A new implementation of the element free Galerkin method," *Computer Methods in Applied Mechanics and Engineering,* vol. 113, no. 3-4, pp. 397-414, 1994.

[41] M. A. Puso, E. Zywicz and J. Chen, "A new stabilized nodal integration approach," in *Meshfree Methods for Partial Differential Equations III*, Berlin, Springer, 2007, pp. 207-217.

[42] T.-H. Huang, H. Wei, J.-S. Chen and M. Hillman, "RKPM2D: Open-Source Implementation of Nodally Integrated Reproducing Kernel Particle Method for Solving Partial Differential Equations.," *Advances in Engineering Software, Submitted,* 2019.

[43] A. Shestakov, D. Kershaw and G. Zimmerman, "Test problems in radiative transfer calculations," *Nuclear science and engineering,* vol. 105, no. 1, pp. 88-104, 1990.

Appendix

The elasticity problem is chosen to demonstrate the performance of RKPM2D, whereas only slight modifications in certain subroutines are needed to convert the code to solve a different PDE. An example of converting the code to solve a Poisson problem is illustrated here. The corresponding input files and modified subroutines are also included in folder "04_PoissonProblem_src".

A Poisson problem is given as follows:

$$\begin{aligned} \left(D_{ij}u_{,j}\right)_{,i} + b = 0 \quad & \text{on } \Omega \\ D_{ij}u_{,j}n_i = t \quad & \text{on } \partial\Omega_t \\ u = g \quad & \text{on } \partial\Omega_g \end{aligned} \tag{36}$$

where u is a scalar field, D_{ij} is the diffusivity, b is the source term, t and g are the prescribed boundary flux and boundary values of u on $\partial\Omega_t$ and $\partial\Omega_g$, respectively. By introducing the RK approximation in Eq. (11), (36) can be recast into the following matrix equations as:

$$\sum_J K_{IJ}u_J - F_I = 0 \tag{37}$$

where

$$K_{IJ} = K_{IJ}^d + K_{IJ}^\beta - \left(K_{IJ}^g + K_{IJ}^{g^T}\right) \tag{38}$$

$$F_I = F_I^b + F_I^t + F_{IJ}^\beta - F_I^g \tag{39}$$

in which the matrices and vectors in nodal integration are expressed as

$$K_{IJ}^d = \int_\Omega \boldsymbol{B}_I^T(\boldsymbol{x})\boldsymbol{D}\boldsymbol{B}_J(\boldsymbol{x})d\Omega \approx \sum_{N=1}^{NP} \boldsymbol{B}_I^T(\boldsymbol{x}_N)\boldsymbol{D}\boldsymbol{B}_J(\boldsymbol{x}_N)A_N \tag{40}$$

$$F_I^b = \int_{\Omega} \Psi_I^T(\boldsymbol{x}) b(\boldsymbol{x}) d\Omega \approx \sum_{N=1}^{NP} \Psi_I^T(\boldsymbol{x}_N) b(\boldsymbol{x}_N) A_N \tag{41}$$

$$F_I^t = \int_{\partial\Omega_t} \Psi_I^T(\boldsymbol{x}) t(\boldsymbol{x}) d\Gamma \approx \sum_{N=1}^{NPt} \Psi_I^T(\boldsymbol{x}_N) t(\boldsymbol{x}_N) L_N \tag{42}$$

$$K_{IJ}^{\beta} = \beta \int_{\partial\Omega_g} \Psi_I^T(\boldsymbol{x}) S \Psi_J(\boldsymbol{x}) d\Gamma \approx \beta \sum_{N=1}^{NPg} \Psi_I^T(\boldsymbol{x}_N) S \Psi_J(\boldsymbol{x}_N) L_N \tag{43}$$

$$K_{IJ}^{g} = \int_{\partial\Omega_g} \boldsymbol{B}_I^T(\boldsymbol{x}) \boldsymbol{D}\boldsymbol{\eta} S \Psi_J(\boldsymbol{x}) d\Gamma \approx \sum_{N=1}^{NPg} \boldsymbol{B}_I^T(\boldsymbol{x}_N) \boldsymbol{D}\boldsymbol{\eta} S \Psi_J(\boldsymbol{x}_N) L_N \tag{44}$$

$$F_{IJ}^{\beta} = \beta \int_{\partial\Omega_g} \Psi_I^T(\boldsymbol{x}) S g d\Gamma \approx \beta \sum_{N=1}^{NPg} \Psi_I^T(\boldsymbol{x}_N) S g L_N \tag{45}$$

$$F_I^{g} = \int_{\partial\Omega_g} \boldsymbol{B}_I^T(\boldsymbol{x}) \boldsymbol{D}\boldsymbol{\eta} S g d\Gamma \approx \sum_{N=1}^{NPg} \boldsymbol{B}_I^T(\boldsymbol{x}_N) \boldsymbol{D}\boldsymbol{\eta} S g L_N \tag{46}$$

$$\boldsymbol{B}_I(\boldsymbol{x}_N) = \begin{bmatrix} \Psi_{I,1}(\boldsymbol{x}_N) \\ \Psi_{I,2}(\boldsymbol{x}_N) \end{bmatrix}, \ \boldsymbol{D} = \begin{bmatrix} d & 0 \\ 0 & d \end{bmatrix}, \ \boldsymbol{\eta} = \begin{bmatrix} n_1 \\ n_2 \end{bmatrix}, \ S = 1. \tag{47}$$

where $\boldsymbol{D}$ is the diffusive tensor, d is the diffusion coefficient and $\boldsymbol{\eta}$ is a collection of components of the surface unit normal on the boundary, and $S = 1$ is set for the convenience of keeping a unified coding structure in RKPM2D. For demonstration purpose, we set the exact solution of the scalar field is $u = 0.1 + 0.1x_1 + 0.2x_2$, as a linear patch test for Poisson equations and all boundary conditions terms can be obtained from the exact solution

The input file for this problem is generated in the function `getInput`. Compared to the input files of Listing 11 for linear elasticity, the following changes need to be made:

- Remove `Model.nu` and `Model.Condition`, as Poisson ratio and plane-stress/strain condition are not required for Poisson problem.

- Replace `Model.E` with `Model.d` (i.e., change the definition of Young's modulus E to be the diffusion coefficient d).
- Replace `Model.ElasticTensor` with `Model.DiffusiveTensor` (i.e., change the definition of elastic tensor $\boldsymbol{C}$ to be the diffusive tensor $\boldsymbol{D}$).
- Set `Model.DiffusiveTensor = diag([Model.d,Model.d])` to define the diffusive tensor $\boldsymbol{D} = \begin{bmatrix} d & 0 \\ 0 & d \end{bmatrix}$.
- Set `Model.DOFu = 1` to change the nodal degrees of freedom `DOFu` from 2 to 1.
- Set `u_exact = 0.1+0.1*x1+0.2*x2` to define the exact solution u^{exact} .

In addition, we need to modify the subroutine `getBoundaryConditions` to generate the exact boundary flux $t = \boldsymbol{\eta}^T \boldsymbol{D}\boldsymbol{\nabla} u^{exact}$, body sources $b = \boldsymbol{\nabla} \cdot (\boldsymbol{D}\boldsymbol{\nabla} u^{exact})$, essential boundary conditions $g = u^{exact}$, and switch matrix $S = 1$ based on a given expression of the exact solution u_{exact} in a symbolic form, as shown in Listing 13.

```
function [function_S,function_g,function_traction,function_b] =
getBoundaryConditions(Model)
syms x1 x2 n1 n2
% function handle for essential boundary condition g
u = Model.ExactSolution.u_exact;
D = Model.DiffusiveTensor;
function_g = matlabFunction(u);
% function handle for diff(u)
dudx1 = diff(u,x1);
dudx2 = diff(u,x2);
flux = D*[dudx1; dudx2;];
% function handle for surface flux (traction)
eta = [n1; n2;];
surf_flux = eta'*flux;
function_traction = matlabFunction(surf_flux,'Vars',[x1 x2 n1 n2]);
% function handle for source b
b = [diff(flux(1),x1)+ diff(flux(2),x2)];
function_b = matlabFunction(b,'Vars',[x1 x2]);
% function handle for switch S
function_S = matlabFunction(sym(1),'Vars',[x1 x2]);
end
```

Listing 13. Command lines of function to generate exact heat flux t, heat sources b, imposed temperature g, and switch matrix S for Poisson problem.

Due to the change of dimensionality in $\boldsymbol{B}$ and $\boldsymbol{\Psi}$ matrix, modifications are made to `MatrixAssmebly` as follows

- Set `d = Model.d` to define the diffusivity coefficient.

- Set `D = Model.DiffusiveTensor` to define the diffusivity from input files.
- Replace `E` with `d` (i.e., replace the Young's modulus `E` with diffusion coefficient `d)`.
- Replace `C` with `D` (i.e., replace elastic tensor `C` with diffusive tensor `D)`.
- Set `B = sparse(2,nP*DOFu)`.
- Set `PSI = sparse(1,nP*DOFu)`.
- Modify the allocation of the `B` and `PSI` from shape function `SHP` and derivative `SHPDX1`, `SHPDX2` as:
  - `PSI = SHP(idx_nQuad,:);`
  - `B(1,:)= SHPDX1(idx_nQuad,:);`
  - `B(2,:) = SHPDX2(idx_nQuad,:);`
- Set `ETA = [n1; n2;]` to define the surface normal $\boldsymbol{\eta}$.

With the abovementioned modifications, RKPM2D is converted to a code for solving a Poisson problem. By comparing the original code for the elasticity problem with the modified code for Poisson problem, one can see that very few code modifications are required. This capability of easy code extension is a unique feature of RKPM2D.