Kragujevac Journal of Mathematics
Volume 46(6) (2022), Pages 929–942.

NUMERICAL SOLUTION OF SHRÖDINGER EQUATIONS BASED ON THE MESHLESS METHODS

MOJTABA RAHIMI[1], SEYED MEHDI KARBASSI[1],
AND MOHAMMAD REZA HOOSHMANDASL[2*]

Abstract. In this work, two-dimensional time-dependent quantum equation problems are studied. We introduce a numerical algorithm for solving the two-dimensional nonlinear complex quantum system with MLS and FDM methods. An efficient and accurate computational algorithm based on both, the moving least squares (MLS) and the finite difference (FDM) methods is proposed for solving it. The results demonstrate that the proposed algorithm is a robust algorithm with good accuracy. This is developed on MLS and FDM methods using numerical simulation for solving these kind of problems.

1. Introduction

Recently, some researchers have considered several types of quantum equations such as two-dimensional time-dependent Shrödinger equations mainly used for modeling several physical phenomena. These types of equations appear in many science and engineering problems. Very interesting problems in quantum physics consist of multi-particle systems that can be modeled by the multi-particle Shrödinger equations. These equations are important in many different fields of science such as wave propagation, relativistic quantum mechanics, quantum field theory and mathematical physics [2,8,10,12,16,18,19].

There are various achievements on the numerical solution of partial differential equations (PDEs). Many numerical algorithms have been developed for the solution of partial differential equations such as meshless methods, finite difference methods, differential quadrature methods, radial basis functions and collocation base methods.

Key words and phrases. Time-dependent quantum equations, moving least squares (MLS) method, finite difference scheme (FDM).

2010 *Mathematics Subject Classification.* Primary: 41A45. Secondary: 41A25, 65M06.

Received: December 12, 2019.

Accepted: May 22, 2020.

However, the meshless methods are generally used as mathematical tools for solving system of differential equations. These numerical approximation methods are suitable for solving ordinary and partial differential equations. The simple structure of these methods are implemented in algorithms to solve these types of differential equations [1, 3–5, 7–9, 13, 14, 17].

The main object of this paper is to present an efficient numerical algorithm based on the MLS method to solve the following 2D time-dependent Shrödinger equation of the form:

$$-iu_t(x,y,t) = (u_{xx}(x,y,t) + u_{yy}(x,y,t))u(x,y,t) + V_e(x,y,t)u(x,y,t), \tag{1.1}$$

where $(x,y,t) \in [a,b] \times [c,d] \times [0,T]$. The initial condition for the above equation is

$$u(x,y,0) = g_1(x,y), \tag{1.2}$$

while the boundary conditions are:

$$\begin{aligned} u(a,y,t) = h_1(y,t), u(b,y,t) = h_2(y,t),\\ u(x,c,t) = h_3(x,t), u(x,d,t) = h_2(x,t). \end{aligned} \tag{1.3}$$

2. The MLS Approximation

Consider the 2-D unknown function $u(x,y)$ and randomly located nodes (x_i, y_j), $i = 1, 2, \ldots, N$, $j = 1, 2, \ldots, M$. Then we define

$$z_k = (x_i, y_j),$$

where

$$i = \begin{cases} \frac{k}{M}, & M|k, \\ k - \left[\frac{k}{M}\right] M, & \text{otherwise}, \end{cases}$$

and

$$j = \begin{cases} M, & M|k, \\ \left[\frac{k}{M}\right] + 1, & \text{otherwise}. \end{cases}$$

In other words we have $k = (i-1)m + j$. To approximate $u(x,y)$ by MLS method, we can write

$$u(x,y) = \sum_{j=1}^{m} p_j(x,y)a_j(x,y) \triangleq \mathbf{p}^T(x,y)\mathbf{a}(x,y), \quad \text{for all } (x,y) \in [a,b] \times [c,d], \tag{2.1}$$

where $a_j(x,y)$ are the unknown coefficients and $\mathbf{p}^T(x,y) = [\ p_1(x,y) \ \cdots \ p_m(x,y)\ ]$ here $p_j(x,y)$ are the basis polynomial functions. For example, the linear basis is $\mathbf{p}^T(x,y) = [\ 1 \ \ x \ \ y\ ]$ and the quadratic basis is $\mathbf{p}^T(x,y) = [\ 1 \ \ x \ \ y \ \ x^2 \ \ y^2 \ \ xy\ ]$. The unknown coefficients $a_j(x,y)$ can be determined by MLS method. In this method, the main concept is minimizing the weighted error of the exact values and approximations of the function. The weighted error function is defined as

$$\mathcal{S}(x,y)=\sum_{k=1}^{N^2} w_k(x,y)\left(u^h(x_i,y_j)-u_k\right)^2=\sum_{k=1}^{N^2} w_k(x,y)\Big(\mathbf{p}^T(x_i,y_j)\mathbf{a}(x,y)-u_k\Big)^2. \tag{2.2}$$

In weighted error function, (x_i,y_j), $i=1,2,\dots,N$, $j=1,2,\dots,N$, are the nodes and $w_k(x_i,y_i)$, $k=1,2,\dots,N^2$, are weighting functions associated with the nodes (x_i,y_j), $i=1,2,\dots,N$, $j=1,2,\dots,N$. Here, the Gaussian weight function is preferred rather than other popular weighted functions. In k^{th} node, we use the Gaussian weight function as [11]:

$$w_k(x,y)=\begin{cases}\dfrac{\exp\left[-(\frac{d_k}{\alpha})^2\right]-\exp[-(\frac{h_k}{\alpha})^2]}{1-\exp\left[-(\frac{h_k}{\alpha})^2\right]}, & 0\le d_k<h_k,\\ 0, & d_k\ge h_k,\end{cases}$$

where $d_k=\|(x,y)-(x_i,y_j)\|$, α is a constant used for controlling the shape of the weight function and h_k denotes support domain size of the node (x_i,y_j).

In other words, an equivalent form of the Gaussian weight function can be used:

$$w_k(x,y)=\begin{cases}\dfrac{\exp\left(-s_k^2r_k^2\right)-\exp(-s_k^2)}{1-\exp(-s_k^2)}, & 0\le r_k<1,\\ 0, & r_k\ge 1,\end{cases} \tag{2.3}$$

where

$$r_k=\frac{\|(x,y)-(x_i,y_j)\|}{h_k},\qquad s_k=\frac{h_k}{\alpha}.$$

To minimize $\mathcal{S}$ in (2.2), with respect to $\mathbf{a}(x,y)$ we require that

$$\frac{\partial\mathcal{S}}{\partial\mathbf{a}}=0,$$

which yields the following equations:

$$\begin{aligned}\sum_{k=1}^{N^2} w_k(x,y)2p_1(x_i,y_j)[\mathbf{p}^T(x_i,y_j)\mathbf{a}(x,y)-u_k]&=0,\\ \sum_{k=1}^{N^2} w_k(x,y)2p_2(x_i,y_j)[\mathbf{p}^T(x_i,y_j)\mathbf{a}(x,y)-u_k]&=0,\\ &\vdots\\ \sum_{k=1}^{N^2} w_k(x,y)2p_m(x_i,y_j)[\mathbf{p}^T(x_i,y_j)\mathbf{a}(x,y)-u_k]&=0.\end{aligned}$$

The above equations can be formulated as

$$\begin{aligned}(2.4)\qquad &\sum_{k=1}^{N^2} w_k(x,y)\begin{pmatrix} p_1(x_i,y_j)\\ \vdots \\ p_m(x_i,y_j)\end{pmatrix}[p_1(x_i,y_j),\dots,p_m(x_i,y_j)]\begin{pmatrix} a_1(x,y)\\ \vdots \\ a_m(x,y)\end{pmatrix}\\ &=\sum_{k=1}^{N^2} w_k(x,y)\begin{pmatrix} p_1(x_i,y_j)\\ \vdots \\ p_m(x_i,y_j)\end{pmatrix}u_k.\end{aligned}$$

The matrices $\mathbf{C}(x,y)$, $\mathbf{D}(x,y)$ and column vector $\mathbf{u}$ are defined as follows:

$$\begin{aligned}(2.5)\qquad &\mathbf{C}(x,y)=\sum_{k=1}^{N^2} w_k(x,y)\begin{pmatrix} p_1(x_i,y_j)\\ \vdots \\ p_m(x_i,y_j)\end{pmatrix}[p_1(x_i,y_j),\dots,p_m(x_i,y_j)],\\ &\mathbf{D}(x,y)=\sum_{k=1}^{N^2} w_k(x,y)\begin{pmatrix} p_1(x_i,y_j)\\ \vdots \\ p_1(x_i,y_j)\end{pmatrix},\\ &\mathbf{u}=[u_1,\dots,u_{N^2}]^T.\end{aligned}$$

Then (2.4) may be re-written in the following compact form:

$$\mathbf{C}(x,y)\mathbf{a}(x,y)=\mathbf{D}(x,y)\mathbf{u},$$

therefore, we have

$$\mathbf{a}(x,y)=\mathbf{C}^{-1}(x,y)\mathbf{D}(x,y)\mathbf{u}.$$

After computing $\mathbf{a}(x,y)$ in the above equation and substituting it into (2.1), the MLS can be approximated as follows:

$$u(x,y)=\mathbf{p}^T(x,y)\mathbf{C}^{-1}(x,y)\mathbf{D}(x,y)\ =\boldsymbol{\Phi}^T(x,y)\mathbf{u}=\sum_{k=1}^{N^2}\phi_k(x,y)u_k,$$

where

$$(2.6)\qquad \boldsymbol{\Phi}^T(x,y)=[\ \phi_1(x,y)\quad\cdots\quad\phi_{N^2}(x)\ ]=\mathbf{p}^T(x,y)\mathbf{C}^{-1}(x,y)\mathbf{D}(x,y)$$

and

$$\phi_k(x,y)=\mathbf{p}^T(x,y)[\mathbf{C}^{-1}(x,y)w_k(x,y)\mathbf{p}(x_i,y_j).$$

The function $\phi_k(x,y)$ is commonly known as the shape function of the nodal point (x_i,y_j) in the MLS approximation. In this section, we need to compute derivatives of $\phi_k(x,y)$ and $\mathbf{C}^{-1}(x,y)$. If $\mathbf{F}$ is a nonsingular matrix then, we have $\mathbf{F}^{-1}\mathbf{F}=\mathbf{I}$. Thus, its differentiation with respect to x gives $\mathbf{F}_x^{-1}\mathbf{F}+\mathbf{F}^{-1}\mathbf{F}_x=0$ and $\mathbf{F}_x^{-1}=-\mathbf{F}^{-1}\mathbf{F}_x\mathbf{F}^{-1}$. We can write

$$\mathbf{F}_{xx}^{-1}=-\left(\mathbf{F}_x^{-1}\mathbf{F}_x\mathbf{F}^{-1}+\mathbf{F}^{-1}\mathbf{F}_{xx}\mathbf{F}^{-1}+\mathbf{F}^{-1}\mathbf{F}_x\mathbf{F}_x^{-1}\right).$$

The above equations can be written in a simpler form:

$$\mathbf{F}_{xx}^{-1} = 2\left(\mathbf{F}^{-1}\mathbf{F}_x\mathbf{F}^{-1}\mathbf{F}_x\mathbf{F}^{-1}\right) - \mathbf{F}^{-1}\mathbf{F}_{xx}\mathbf{F}^{-1}.$$

Now, the first derivative of $\phi_j(x,y)$ with respect to x is obtained as

$$\begin{aligned}\mathbf{\Phi}_x^T &= \mathbf{p}_x^T\mathbf{C}^{-1}\mathbf{D} + \mathbf{p}^T\mathbf{C}_x^{-1}\mathbf{D} + \mathbf{p}^T\mathbf{C}^{-1}\mathbf{D}_x\\ &= \mathbf{p}_x^T\mathbf{C}^{-1}\mathbf{D} - \mathbf{p}^T\mathbf{C}^{-1}\mathbf{C}_x\mathbf{C}^{-1}\mathbf{D} + \mathbf{p}^T\mathbf{C}^{-1}\mathbf{D}_x,\end{aligned}$$

also the second derivative of $\phi_j(x,y)$ with respect to x is obtained as

$$\begin{aligned}\mathbf{\Phi}_{xx}^T =& \mathbf{p}_{xx}^T\mathbf{C}^{-1}\mathbf{D} - \mathbf{p}_x^T\mathbf{C}^{-1}\mathbf{C}_x\mathbf{C}^{-1}\mathbf{D} + \mathbf{p}_x^T\mathbf{C}^{-1}\mathbf{D}_x - \mathbf{p}_x^T\mathbf{C}^{-1}\mathbf{C}_x\mathbf{C}^{-1}\mathbf{D}\\ &+ \mathbf{p}^T\mathbf{C}^{-1}\mathbf{C}_x\mathbf{C}^{-1}\mathbf{C}_x\mathbf{C}^{-1}\mathbf{D} - \mathbf{p}^T\mathbf{C}^{-1}\mathbf{C}_{xx}\mathbf{C}^{-1}\mathbf{D} + \mathbf{p}^T\mathbf{C}^{-1}\mathbf{C}_x\mathbf{C}^{-1}\mathbf{C}_x\mathbf{C}^{-1}\mathbf{D}\\ &- \mathbf{p}^T\mathbf{C}^{-1}\mathbf{C}_x\mathbf{C}^{-1}\mathbf{D}_x + \mathbf{p}_x^T\mathbf{C}^{-1}\mathbf{D}_x + \mathbf{p}^T\mathbf{C}^{-1}\mathbf{D}_{xx} - \mathbf{p}^T\mathbf{C}^{-1}\mathbf{C}_x\mathbf{C}^{-1}\mathbf{D}_x.\end{aligned}$$

We can write these equations in a simpler form:

$$\begin{aligned}\mathbf{\Phi}_{xx}^T =& \mathbf{p}_{xx}^T\mathbf{C}^{-1}\mathbf{D} + 2(\mathbf{p}^T\mathbf{C}^{-1}\mathbf{C}_x\mathbf{C}^{-1}\mathbf{C}_x\mathbf{C}^{-1}\mathbf{D} - \mathbf{p}^T\mathbf{C}^{-1}\mathbf{C}_x\mathbf{C}^{-1}\mathbf{D}_x)\\ &- \mathbf{p}^T\mathbf{C}^{-1}\mathbf{C}_{xx}\mathbf{C}^{-1}\mathbf{D} + \mathbf{p}^T\mathbf{C}^{-1}\mathbf{D}_{xx} - 2(\mathbf{p}_x^T\mathbf{C}^{-1}\mathbf{C}_x\mathbf{C}^{-1}\mathbf{D} - \mathbf{p}_x^T\mathbf{C}^{-1}\mathbf{D}_x).\end{aligned}\tag{2.7}$$

In a similar method, we can obtain the first and second derivatives of $\phi_j(x,y)$ with respect to y as follows:

$$\mathbf{\Phi}_y^T = \mathbf{p}_y^T\mathbf{C}^{-1}\mathbf{D} - \mathbf{p}^T\mathbf{C}^{-1}\mathbf{C}_y\mathbf{C}^{-1}\mathbf{D} + \mathbf{p}^T\mathbf{C}^{-1}\mathbf{D}_y$$

and

$$\begin{aligned}\mathbf{\Phi}_{yy}^T =& \mathbf{p}_{yy}^T\mathbf{C}^{-1}\mathbf{D} - \mathbf{p}^T\mathbf{C}^{-1}\mathbf{C}_{yy}\mathbf{C}^{-1}\mathbf{D} + \mathbf{p}^T\mathbf{C}^{-1}\mathbf{D}_{yy} + 2(\mathbf{p}^T\mathbf{C}^{-1}\mathbf{C}_y\mathbf{C}^{-1}\mathbf{C}_y\mathbf{C}^{-1}\mathbf{D})\\ &- 2(\mathbf{p}^T\mathbf{C}^{-1}\mathbf{C}_y\mathbf{C}^{-1}\mathbf{D}_y + \mathbf{p}_y^T\mathbf{C}^{-1}\mathbf{C}_y\mathbf{C}^{-1}\mathbf{D} - \mathbf{p}_y^T\mathbf{C}^{-1}\mathbf{D}_y).\end{aligned}\tag{2.8}$$

3. Discretization of Shrödinger Equation

Consider the Shrödinger equation (1.1) with initial condition (1.2) and boundary condition (1.3). This can be discretized by the following $\theta-$weighted plan [15]:

$$\begin{aligned}-i\frac{u(x,y,t+dt)-u(x,y,t)}{dt} =& \theta(\partial_{xx}+\partial_{yy}+V_e(x,y,t+dt))u(x,y,t+dt)\\ &+(1-\theta)(\partial_{xx}+\partial_{yy}+V_e(x,y,t))u(x,y,t).\end{aligned}$$

Then

$$\begin{aligned}&(-i-dt\theta(\partial_{xx}+\partial_{yy}+V_e(x,y,t+dt)))u(x,y,t+dt)\\ =&(-i+(1-\theta)dt(\partial_{xx}+\partial_{yy}+V_e(x,y,t)))u(x,y,t).\end{aligned}\tag{3.1}$$

Consider N^2 points (x_i,y_j), where $x_i = a+(i-1)h \in [a,b]$, $y_j = c+(j-1)k \in [c,d]$ and $i,j = 1,\dots,N$, such that $x_1 = a$, $x_N = b$, $y_1 = c$, $y_N = d$, $h = \frac{b-a}{N-1}$ and $k = \frac{d-c}{N-1}$. In addition, in the time interval $[0,T]$, the grid points are $t^n = ndt$, $n = 0,1,2,\dots,M$, where $M = \frac{T}{dt}$. We apply the finite difference method to discretize the variable-order time fractional derivative by replacement t^{n+1} into (3.1). Then we have:

$$(-i - \theta dt(\partial_{xx} + \partial_{yy} + V_e^{n+1}))u^{n+1} = (-i + (1-\theta)dt(\partial_{xx} + \partial_{yy} + V_e^{n}))u^{n}. \tag{3.2}$$

Now, by using the MLS shape functions we can approximate $u^n(x)$ as follows:

$$u^n(x,y) = \sum_{k=1}^{N^2} \mu_k^n \varphi_k(x,y), \tag{3.3}$$

where $\varphi_k(x,y)$, $k = 1,2,\dots,N^2$, are the shape functions for the MLS approximation and μ_k^n, $k = 1,2,\dots,N^2$, are unknown coefficients, to be determined. Thus, to determine the values of coefficients μ_k^n, $k = 1,2,\dots,N^2$, we use N^2 collocation points of $u^n(x,y)$ as:

$$u^n(x_i,y_j) = \sum_{k=1}^{N^2} \mu_k^n \varphi_k(x_i,y_j), \tag{3.4}$$

where $i,j = 1,2,\dots,N$. Rewriting (3.4) in a compact form, we have:

$$[\mathbf{u}]^n = \mathbf{L}[\mu]^n,$$

where $[\mathbf{u}]^n = [\begin{array}{cccc} u_1^n & u_2^n & \cdots & u_{N^2}^n \end{array}]^T$, $[\mu]^n = [\begin{array}{cccc} \mu_1^n & \mu_2^n & \cdots & \mu_{N^2}^n \end{array}]^T$ and $\mathbf{L}$ is an $N^2 \times N^2$ matrix given by:

$$\mathbf{L} = [l_{ij}] = \begin{pmatrix} \varphi_{1,(1,1)} & \cdots & \varphi_{N^2,(1,1)} \\ \vdots & \ddots & \vdots \\ \varphi_{1,(N,N)} & \cdots & \varphi_{N^2,(N,N)} \end{pmatrix}, \tag{3.5}$$

where $\varphi_{k,(i,j)} = \varphi_k(x_i,y_j)$.

Assuming q internal points and $N-q$ boundary points, then matrix $\mathbf{L}$ can be separated into $\mathbf{L} = \mathbf{L1} + \mathbf{L2}$ in which the entries of $\mathbf{L1}$ and $\mathbf{L2}$ are:

$$\begin{aligned} \mathbf{L1} = [l_{ij}^{(1)}] &= \begin{cases} L_{ij}, & 1 \le i \le q,\ 1 \le j \le N, \\ 0, & \text{elsewhere}, \end{cases} \\ \mathbf{L2} = [l_{ij}^{(2)}] &= \begin{cases} L_{ij}, & q+1 \le i \le N,\ 1 \le j \le N, \\ 0, & \text{elsewhere}. \end{cases} \end{aligned} \tag{3.6}$$

Using (3.3), we can write u_{xx} and u_{yy} as follows:

$$u_{xx}^n(x,y) = \sum_{k=1}^{N^2} \mu_k^n \frac{\partial^2 \varphi_k(x,y)}{\partial x^2} = \sum_{k=1}^{N^2} \mu_k^n \psi_k(x,y), \tag{3.7}$$

$$u_{yy}^n(x,y) = \sum_{k=1}^{N^2} \mu_k^n \frac{\partial^2 \varphi_k(x,y)}{\partial y^2} = \sum_{k=1}^{N^2} \mu_k^n \eta_k(x,y), \tag{3.8}$$

where $\psi_k(x,y)$ and $\eta_k(x,y)$ for $k=1,2,\dots,N^2$, are obtained by (2.7) and (2.8). By taking the collocation points in (3.7) and (3.8), we obtain:

$$u_{xx}^n(x_i,y_j)=\sum_{k=1}^{N^2}\mu_k^n\psi_k(x_i,y_j),\quad i,j=2,3,\dots,N-1,$$
$$u_{yy}^n(x_i,y_j)=\sum_{k=1}^{N^2}\mu_k^n\eta_k(x_i,y_j),\quad i,j=2,3,\dots,N-1,$$

which can be rephrased as:

$$[\mathbf{u}_{xx}]^n=\mathbf{K}[\mu]^n, \tag{3.9}$$
$$[\mathbf{u}_{yy}]^n=\mathbf{H}[\mu]^n, \tag{3.10}$$

where

$$\mathbf{K}=[k_{ij}]=\begin{pmatrix}\psi_{11} & \cdots & \psi_{(N)1}\\ \vdots & \ddots & \vdots\\ \psi_{1(q)} & \cdots & \psi_{(N)(q)}\\ 0 & \cdots & 0\end{pmatrix}, \tag{3.11}$$

$$\mathbf{H}=[h_{ij}]=\begin{pmatrix}\eta_{11} & \cdots & \eta_{(N)1}\\ \vdots & \ddots & \vdots\\ \eta_{1(q)} & \cdots & \eta_{(N)(q)}\\ 0 & \cdots & 0\end{pmatrix}, \tag{3.12}$$

in which $\eta_{ji}=\eta_j(x_i,y_i)$ and $\psi_{ji}=\psi_j(x_i,y_i)$.

Now, by replacing (3.9) and (3.10) into (3.2) together with (1.2), (1.3) and using the collocation points, the following matrix form is obtained:

$$\mathbf{M}[\mu]^{n+1}=\mathbf{N}[\mu]^n+[\mathbf{G}]^{n+1}, \tag{3.13}$$

where

$$\mathbf{M}=-i\mathbf{L1}+\theta dt([\mathbf{K}+\mathbf{H}]-\theta dt\mathbf{V}_e^n*\mathbf{L1}+\mathbf{L2}, \tag{3.14}$$
$$\mathbf{N}=(1-\theta)dt[\mathbf{K}+\mathbf{H}]+(1-\theta)dt\mathbf{V_e^n}*\mathbf{L1}+\mathbf{L2},$$
$$[\mathbf{G}]^{n+1}=[\ 0\ \ 0\ \ \cdots\ \ g_{q+1}^{n+1}\cdots\cdots g_N^{n+1}\ ]^T. \tag{3.15}$$

The symbol $*$ means that each component of the left vector is multiplied to all the components of corresponding row of right matrix. For solving the system, $[\mu]^{n+1}$ can be computed by recursive relation in (3.13) for $n=1,2,\dots,M$. First we must compute $[\mu]^0$. Also, it should be noted that according to (3.13) for $n=0$, we have:

$$\mathbf{M}[\mu]^1=\mathbf{N}[\mu]^0+[\mathbf{G}]^1.$$

We can write an algorithm for this approach.

TABLE 1. Absolute errors obtained in Example 5.1 for $N = 8$, $m = 10$ and $dt = 0.01$.

t	Real part(error)	Imaginary part(error)	MAX(error)
0.1	7.748E-7	4.299E-8	7.775E-7
0.3	7.451E-7	2.223E-7	7.776E-7
0.5	6.861E-7	3.659E-7	7.775E-7
0.7	5.998E-7	4.949E-7	7.780E-7
1	4.266E-7	4.949E-7	7.774E-7

4. Proposed Algorithm

The object of this algorithm is designed to solve the Shrödinger equation.
Input: N, m, T(final time), dt(step length) and the functions $V_e(x,y,t)$, $h_1(y,t)$, $h_2(y,t)$, $h_3(x,t)$, $h_4(x,t)$, $g(x,y)$.
Step 1: Define X, Y vectors of grid points in (x,y) coordinates and $P(x,y)$ vectors of the basis function.
Step 2: Define $w(x,y)$ by (2.3).
Step 3: Compute matrices C and D by (2.5).
Step 4: Compute Φ by (2.6), and Φ_{xx} and Φ_{yy} by (2.7) and (2.8).
Step 5: Compute matrices L, L_1, L_2, K and H by (3.5), (3.6), (3.11) and (3.12).
Step 6: Discrete the Shrödinger equation to (3.2).
Step 7: Compute matrices M, N and vector G by (3.14).
Step 8: Put $\mathbf{M}[\mu]^{n+1} = \mathbf{N}[\mu]^n + [\mathbf{G}]^{n+1}$.
Step 9: Compute $[\mu]^0 = L^{-1}g_1(X,Y)$.
Step 10: Substitute $[\mu]^0$ in the above matrix equation to obtain other $[\mu]^i$, $i = 0,\dots,n$.
Output: The approximate solution $u^n(x,y) = \sum_{j=1}^{N} \mu_j^n \varphi_j(x,y)$.

5. Numerical Examples

In this section, some numerical examples are presented with their exact solutions, to demonstrate the performance and validity of the proposed method.

Example 5.1. Consider the quantum equation as following

$$-iu_t(x,y,t) = (u_{xx}(x,y,t) + u_{yy}(x,y,t))u(x,y,t) + V_e(x,y,t)u(x,y,t),$$

in the region $(x,y) \in [0,1]$ with $V_e(x,y,t) = 1 - \frac{60}{x^2} - \frac{60}{y^2}$.

The initial condition is $u(x,y,0) = x^5y^5$ and the boundary conditions are

$$u(0,y,t) = 0, \quad u(1,y,t) = y^2e^{it},$$
$$u(x,0,t) = 0, \quad u(x,1,t) = x^2e^{it}.$$

The exact solution of this equation is $u(x,y,t) = x^5y^5e^{it}$.

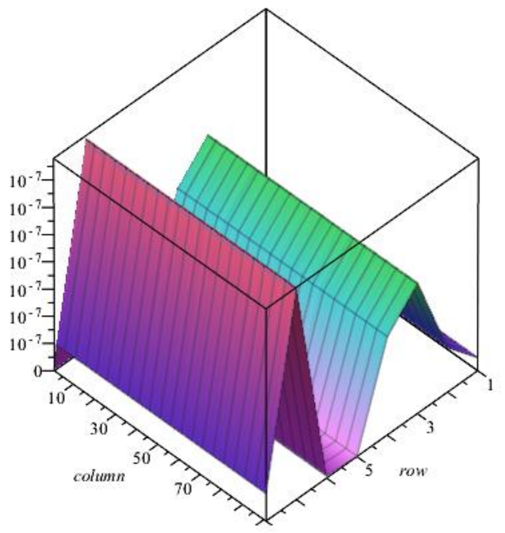

FIGURE 1. The plot of absolute error for Example 5.1.

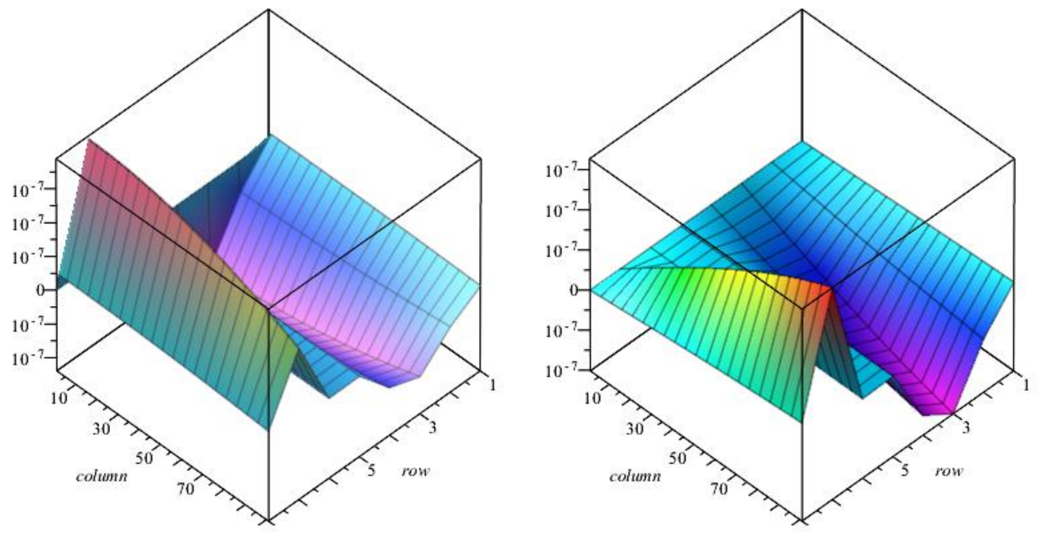

FIGURE 2. The plots of the approximate results (real part in left side) and (imaginary part in right side) for Example 5.1.

We solved the above problem by proposed algorithm for $N = 8$, $m = 10$ and $dt = 0.01$. The plot of the absolute errors is shown in Figure 1 and plots of real and imaginary parts of absolute errors are shown in Figure 2. Approximate results of absolute errors and their real and imaginary parts for different t are presented in Table 1.

From Figures 1–2 and Table 1 it can be observed that the proposed algorithm is very efficient and accurate.

TABLE 2. Absolute errors obtained in Example 5.2 for $N = 8$, $m = 12$ and $dt = 0.01$.

t	Real part	Imaginary part	MAX(error)
0.1	3.212E-4	2.733E-7	3.221E-4
0.3	3.214E-4	8.692E-7	3.214E-4
0.5	2.828E-4	1.430E-6	2.828E-4
0.7	2.819E-4	1.935E-6	2.811E-4
1	2.817E-4	2.254E-6	2.817E-4

Example 5.2. Let us consider the quantum equation as following

$$-iu_t(x, y, t) = (u_{xx}(x, y, t) + u_{yy}(x, y, t))u(x, y, t) + V_e(x, y, t)u(x, y, t),$$

in the region $(x, y) \in [0, 1]$ with $V_e(x, y, t) = 1 + x^2 + y^2$.

The initial condition $u(x, y, 0) = \sin(xy)$, and boundary conditions are

$$u(0, y, t) = 0, \quad u(1, y, t) = \sin(y)e^{it},$$
$$u(x, 0, t) = 0, \quad u(x, 1, t) = \sin(x)e^{it}.$$

The exact solution of the above equation is $u(x, y, t) = \sin(xy)e^{it}$.

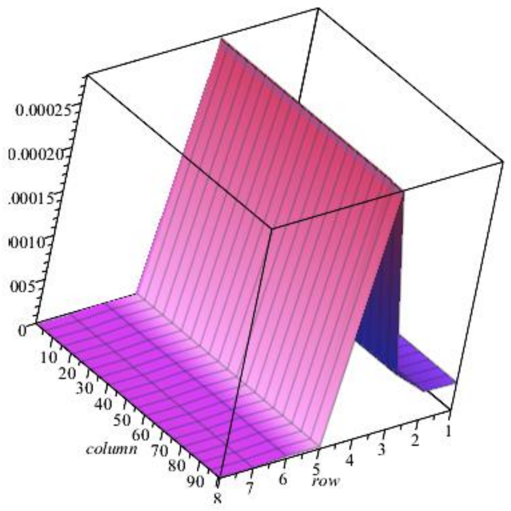

FIGURE 3. The plot of absolute error for Example 5.2.

We solved the above problem by proposed algorithm for $N = 8$, $m = 12$ and $dt = 0.01$. The plot of the absolute errors is shown in Figure 3 and plots of real and imaginary parts of absolute errors are shown in Figure 4. Approximate results of absolute errors and their real and imaginary parts for different t are presented in Table 2. From Figures 3–4 and Table 2 it can be observed that the proposed algorithm is very efficient and accurate.

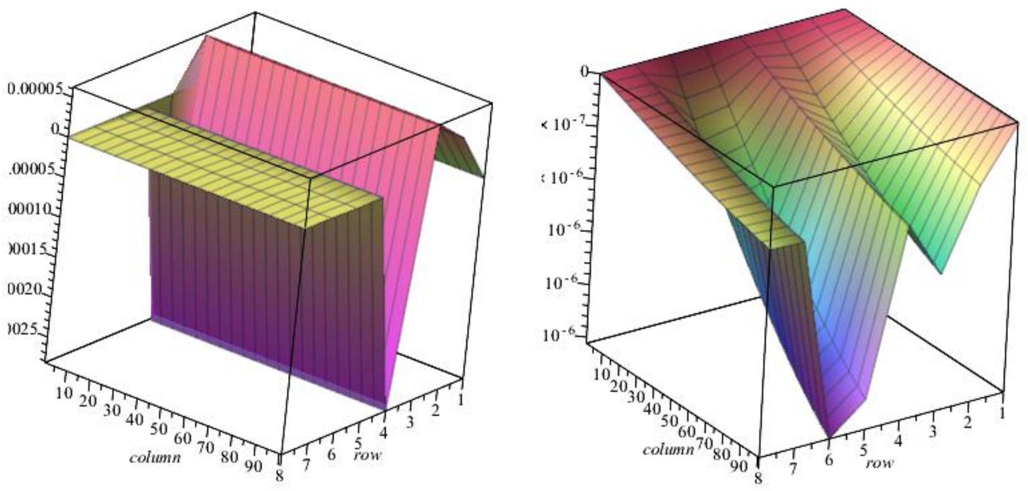

FIGURE 4. The plots of the approximate results (real part in left side) and (imaginary part in right side) for Example 5.2.

TABLE 3. Absolute errors obtained in Example 5.3 for $N = 8$, $m = 10$ and $dt = 0.01$.

t	Real part	Imaginary part	MAX(error)
0.1	8.895E-6	8.027E-6	8.932E-6
0.3	8.553E-6	2.255E-6	8.932E-6
0.5	7.881E-6	4.203E-6	8.932E-6
0.7	6.888E-6	6.585E-6	8.932E-6
1	4.901E-6	7.467E-6	8.832E-6

Example 5.3. Let us consider the quantum equation as following

$$-iu_t(x,y,t) = (u_{xx}(x,y,t) + u_{yy}(x,y,t))u(x,y,t) + V_e(x,y,t)u(x,y,t),$$

in the region $(x,y) \in [0,1]$ with $V_e(x,y,t) = 1 - \frac{2}{x^2} - \frac{2}{y^2}$.

The initial condition is $u(x,y,0) = x^2y^2$, and boundary conditions are

$$u(0,y,t) = 0, \quad u(1,y,t) = y^2e^{it},$$
$$u(x,0,t) = 0, \quad u(x,1,t) = x^2e^{it}.$$

The exact solution of the this example is $u(x,y,t) = x^2y^2e^{it}$.

We solved the above problem by proposed algorithm for $N = 8$, $m = 10$ and $dt = 0.01$. The plot of the absolute errors is shown in Figure 5 and plots of real and imaginary parts of absolute errors are shown in Figure 6. Approximation results of absolute errors and their real and imaginary parts for different t are presented in Table 3.

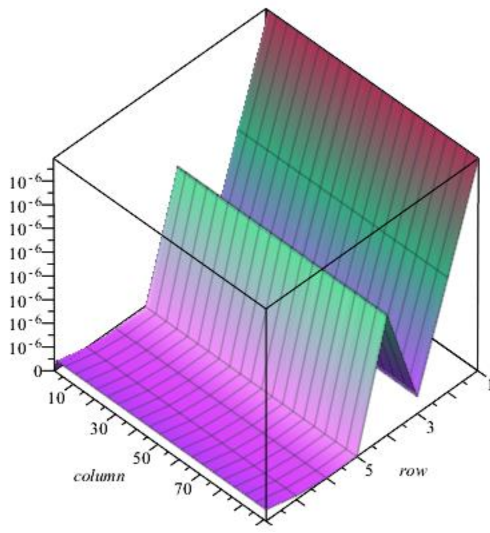

FIGURE 5. The plot of absolute error for Example 5.3.

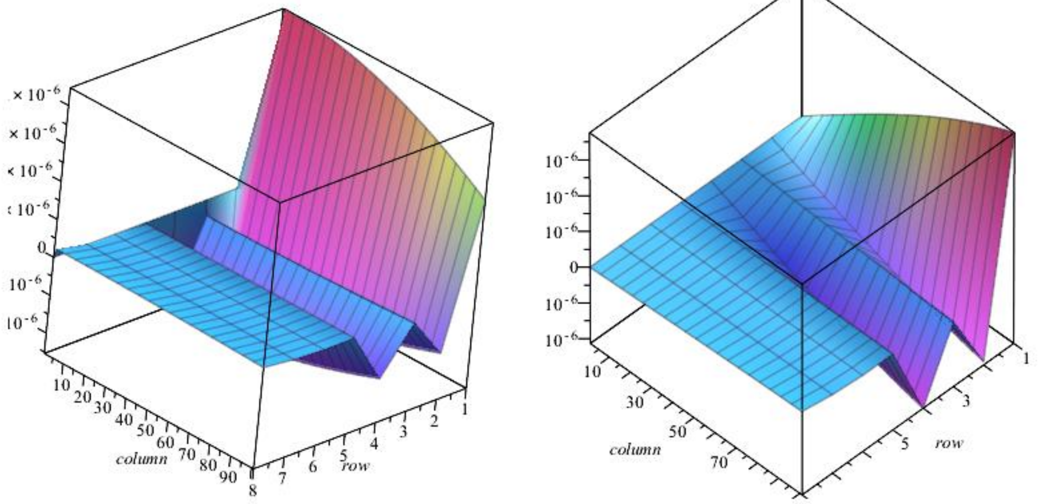

FIGURE 6. The plots of the approximate results (real part in left side) and (imaginary part in right side) for Example 5.3.

Dehghan and Shokri have solved this problem for $dx = dy = 0.1$ and $dt = 0.001$ [6]. By comparing the results of the two methods, it is observed that the proposed algorithm in this work is much better than [6]. In this example, the number of nodes (N) mentioned in [6] was 100 and in the proposed method is $N = 10$. From Figures 5-6 and Table 3 it can be observed that the proposed algorithm is very efficient and accurate.

6. Conclusion

In this paper, based on the moving least squares method (MLS) and the finite difference method (FDM) an algorithm was proposed for solving Shrödinger equation. For this purpose, first we discretized the Shrödinger equation by FDM and then applied the MLS method to obtain a numerical algorithm for solving the partial differential equation thus obtained. To verify the results, three numerical examples were presented. The merit of our approach is the applicability and accuracy the algorithm provides.

References

[1] J. E. Akin, *Finite Elements for Analysis and Design: Computational Mathematics and Applications Series*, Academic Press, San Diego, 2014.

[2] A. Arnold, *Numerically absorbing boundary conditions for quantum evolution equations*, VLSI Design **6** (1998), 313–319.

[3] J. C. Cortés, L. Jódar and L. Villafuerte, *Mean square numerical solution of random differential equations: facts and possibilities*, Comput. Math. Appl. **53** (2007), 1098–1106.

[4] J. C. Cortés, L. Jódar and L. Villafuerte, *Numerical solution of random differential equations: a mean square approach*, Math. Comput. Model. **45** (2007), 757–765.

[5] M. Dehghan and D. Mirzaei, *Meshless local Petrov-Galerkin (MLPG) method for the unsteady magnetohydrodynamic (MHD) flow through pipe with arbitrary wall conductivity*, Appl. Numer. Math. **59** (2009), 1043–1058.

[6] M. Dehghan and A. Shokri, *A numerical method for two-dimensional Schrödinger equation using collocation and radial basis functions*, Comput. Math. Appl. **54** (2007), 136–146.

[7] M. Ehler, *Shrinkage rules for variational minimization problems and applications to analytical ultracentrifugation*, J. Inverse Ill-Posed Probl. **19** (2011), 593–614.

[8] J. C. Kalita, P. Chhabra and S. Kumar, *A semi-discrete higher order compact scheme for the unsteady two-dimensional Schrödinger equation*, J. Comput. Appl. Math. **197** (2006), 141–149.

[9] M. Khodabin, K. Maleknejad, M. Rostami and M. Nouri, *Numerical solution of stochastic differential equations by second order Runge–Kutta methods*, Math. Comput. Model. **53** (2011), 1910–1920.

[10] L. Kong, J. Hong, F. Fu and J. Chen, *Symplectic structure-preserving integrators for the two-dimensional Gross–Pitaevskii equation for BEC*, J. Comput. Appl. Math. **235** (2011), 4937–4948.

[11] G. Li, X. Jin and N. Alum, *Meshless methods for numerical solution of partial differential equations*, in: *Handbook of Materials Modeling*, Springer, Dordrecht, 2005, 2447–2474.

[12] H. Li, C. Jiang and Z. Lv, *A Galerkin energy-preserving method for two dimensional nonlinear Schrödinger equation*, Appl. Math. Comput. **324** (2018), 16–27.

[13] H. Li, Y. Wang and Q. Sheng, *An energy-preserving Crank-Nicolson Galerkin method for Hamiltonian partial differential equations*, Numer. Methods for Partial Differential Equations **32** (2016), 1485–1504.

[14] H. Lin and S. Atluri, *The meshless local Petrov-Galerkin (MLPG) method for solving incompressible Navier-Stokes equations*, CMES-Computer Modeling in Engineering and Sciences **2** (2001), 117–142.

[15] A. Mardani, M. Hooshmandasl, M. Hosseini and M. Heydari, *Moving least squares (MLS) method for the nonlinear hyperbolic telegraph equation with variable coefficients*, Int. J. Comput. Methods **14** (2017), Paper ID 1750026.

[16] M. N. Oğuztöreli, *Time-Lag Control Systems*, Academic Press, London, 1966.

[17] M. Shibahara and S. Atluri, *The meshless local Petrov-Galerkin method for the analysis of heat conduction due to a moving heat source, in welding*, International Journal of Thermal Sciences **50** (2011), 984–992.

[18] W. Thirring, *Classical Mathematical Physics: Dynamical Systems and Field Theories*, Springer Science & Business Media, New York, 2013.

[19] T. Wang, B. Guo and Q. Xu, *Fourth-order compact and energy conservative difference schemes for the nonlinear Schrödinger equation in two dimensions*, J. Comput. Phys. **243** (2013), 382–399.

[1]Department of Mathematical Sciences,
Yazd University,
Yazd, Iran
Email address: `rahimimojtaba1358@gmail.com`
Email address: `smkarbassi@yazd.ac.ir`

[2]Department of Computer Science,
Yazd University,
Yazd, Iran
Email address: `hooshmandasl@yazd.ac.ir`

*Corresponding Author