Tutorial -2

OE6020: Meshfree method applied to hydrodynamics **Roll No: AM17S015**

Moving Least Square Approximation **Yewale Nikhil Janardan**

Problem: Construct MLS based shape functions and its derivatives using different weight functions (cubic spline, quartic spline etc.). Check Kronecker delta property and partition of unity property for shape functions constructed.

Solution: We use rectangular support domain with weighting functions as specified in Appendix A for constructing MLS based shape functions and its derivatives. We use a square domain as shown in Figure 1 such that $x \in [-1,1]$, $y \in [-1,1]$ with 5 nodes each in x and y direction (i.e 25 nodes) respectively.

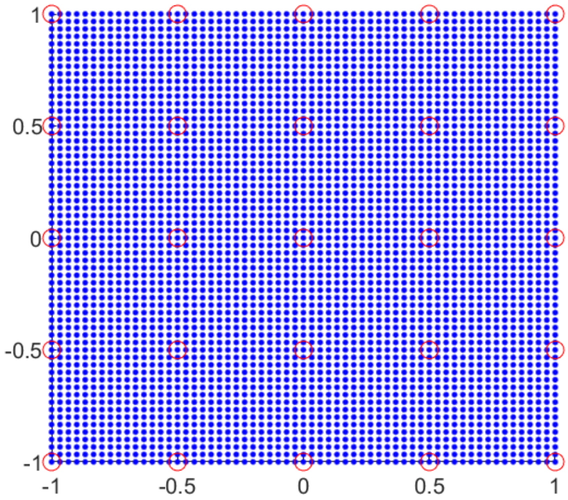

Figure 1

As seen from Figure 1, red circles denote nodal points (5 x 5) and the blue dots denote intermediate evaluation/interpolation points (61 x 61 for Figure 1 to Figure 9). Figure 2 & 3 show weight function variation for cubic and quartic spline respectively for node 13(0, 0). Fig 4 & 5 denote shape function variation for node 13 i.e. x, y = (0,0). Similarly, Fig 6 & 7 show $\partial\phi/\partial x$ and Fig. 8 & 9 show $\partial\phi/\partial y$ variation for node 13 x, y = (0,0).

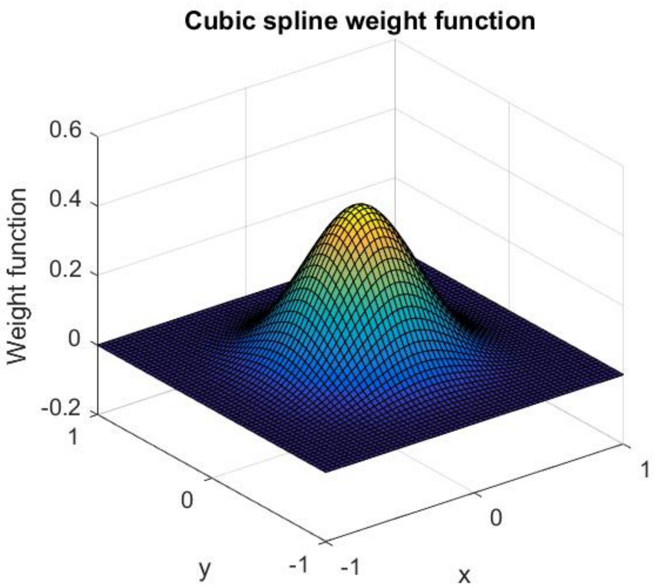

Figure 2

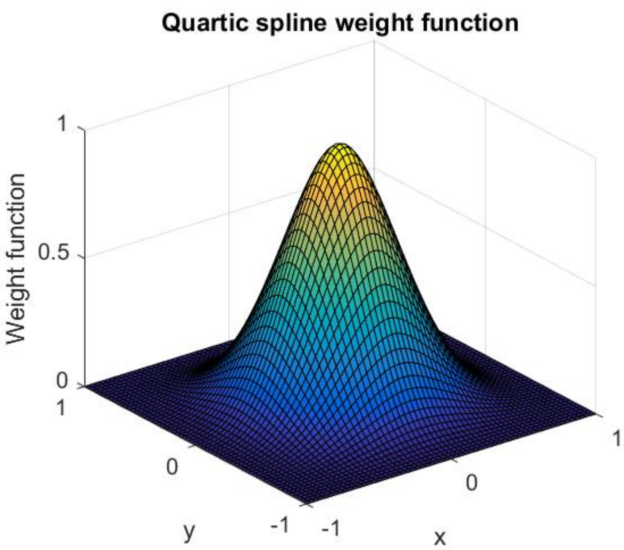

Figure 3

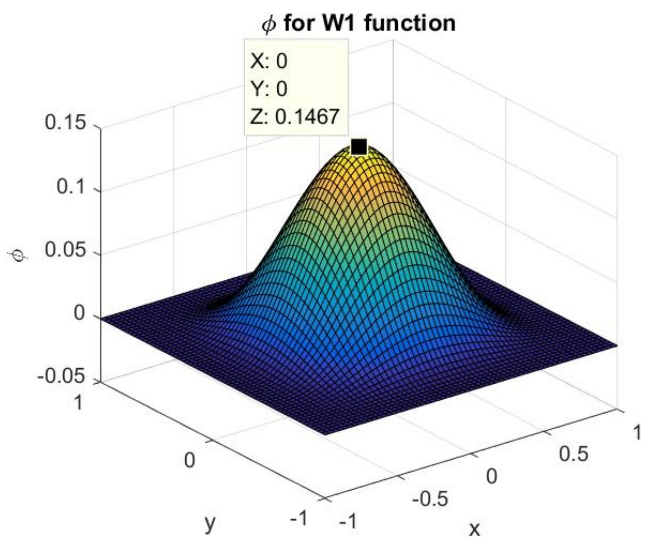

Figure 4

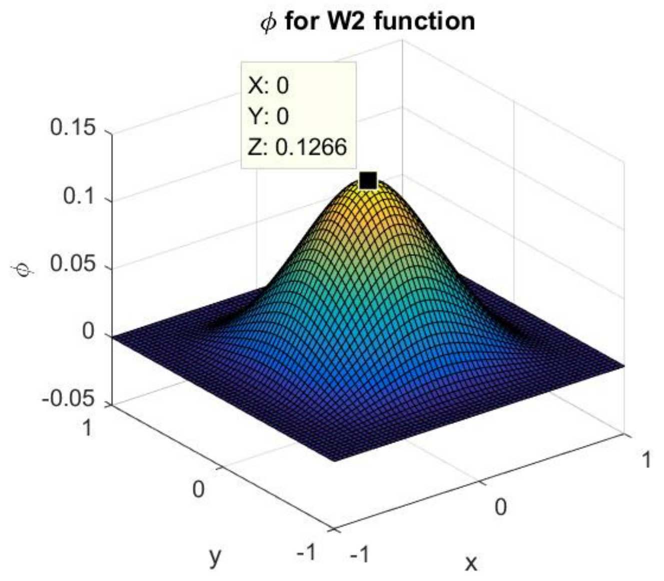

Figure 5

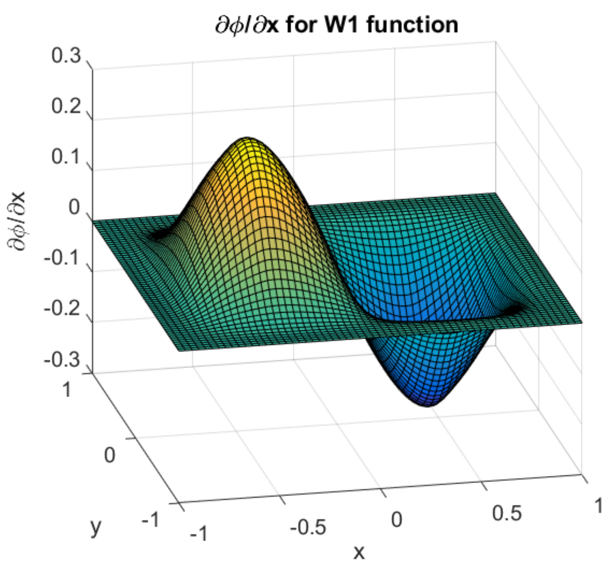

Figure 6

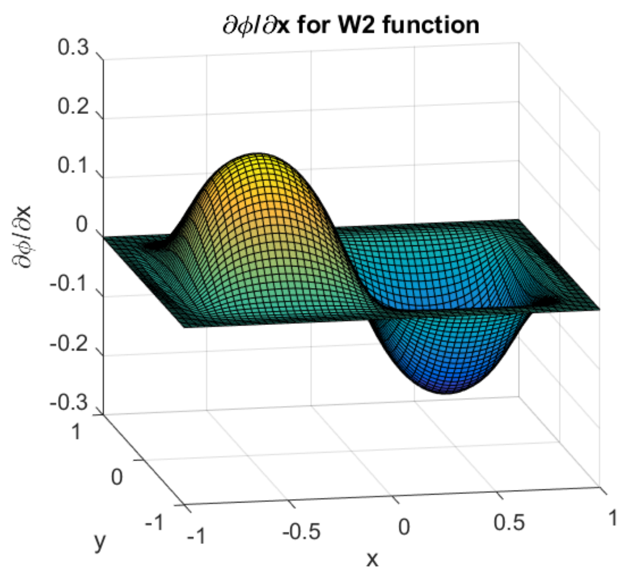

Figure 7

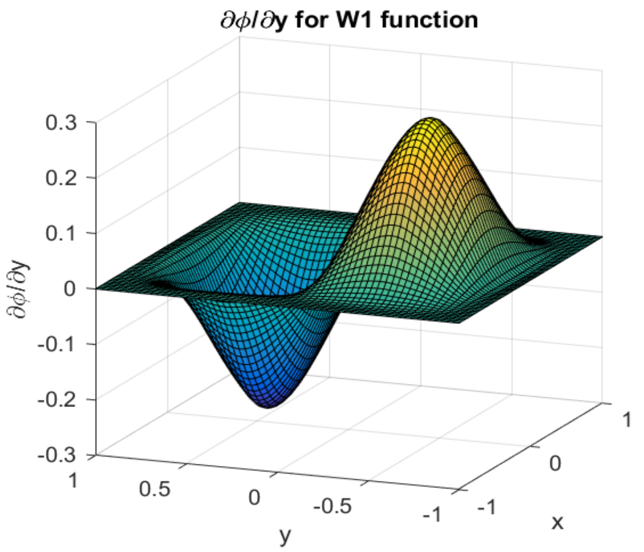

Figure 8

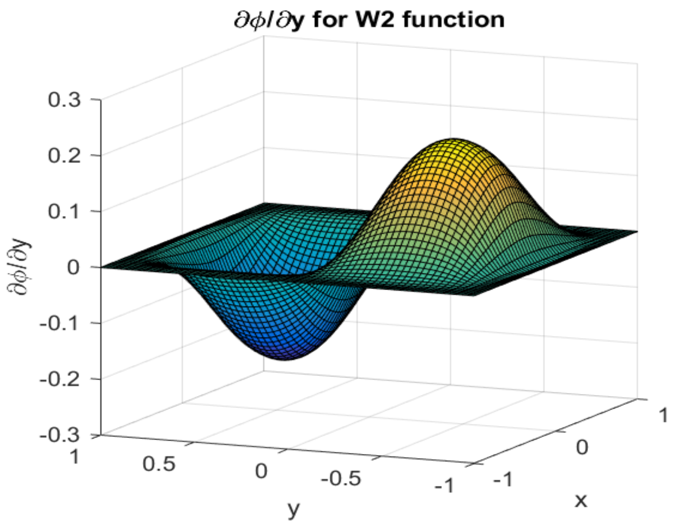

Figure 9

Lack of Kronecker delta property in shape functions: As seen in Figure (4) and Figure (5), *shape function obtained by MLS approximation is a smooth surface and does not pass through the nodal values, hence it does not satisfy Kronecker delta property*

Partition of Unity property in shape functions: See Appendix - B (ii)

Surface fitting using MLS based approach: Case (1) We fit following planar function/surface and observe the accuracy of MLS based approach in Figure 11(for 5 x 5 nodal points)

$f(x,y) = 1 + 2x^2 + 3y^2$ for same domain as shown in Figure (1)

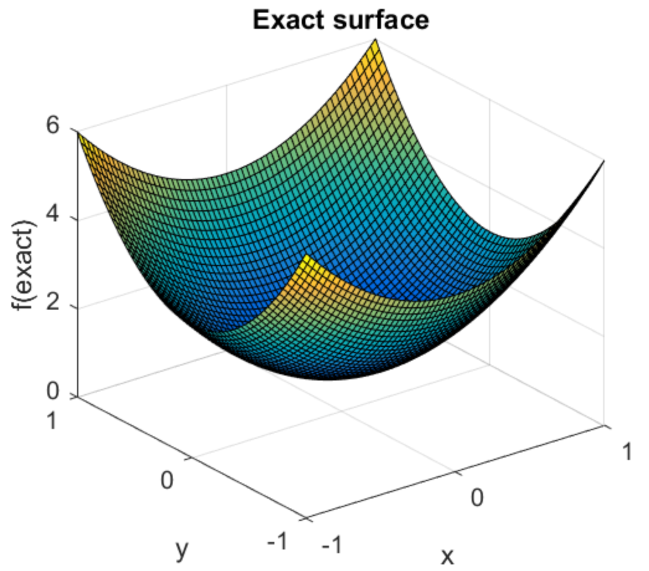

Figure 10

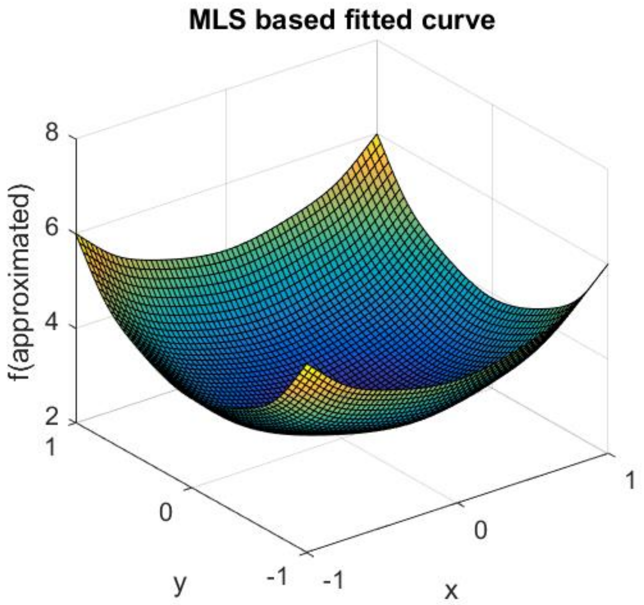

Figure 11

The average fitting errors of function values over the entire domain are defined as,

$$e_t = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{\tilde{f}_i - f_i}{f_i}\right| = 0.4644 \text{ (in case of Figure 10 \& 11)}$$

where $\tilde{f}_i$ is approximated values of function and f_i is the exact values of the function and N is number of nodes in the domain. Following logic of code (part of code) is adopted in MATLAB code as written in Appendix –B (i)

```
summ_errf1 = 0;   summ_errf2 = 0;
for i = 1:nx*ny                 % quantifying error
    summ_errf1 = summ_errf1 + (1/(nx*ny))*abs((f1_app(i) - f1(i))/f1(i));
    summ_errf2 = summ_errf2 + (1/(nx*ny))*abs((f2_app(i) - f2(i))/f2(i));
end
```

Figure 12 shows the steady convergence for the surface fitting using MLS approach for decrease in nodal spacing

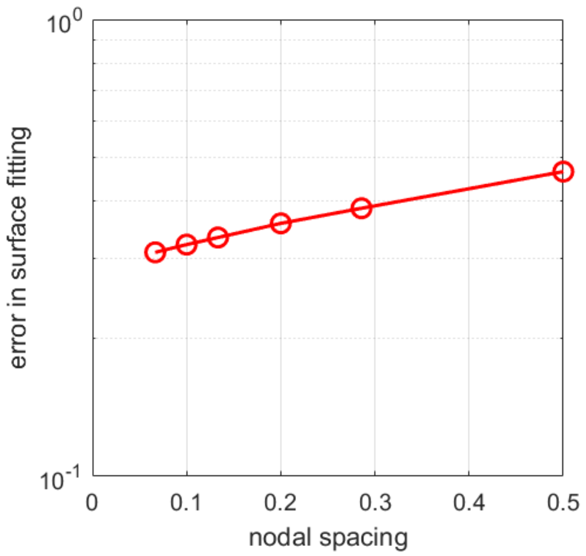

Figure 12

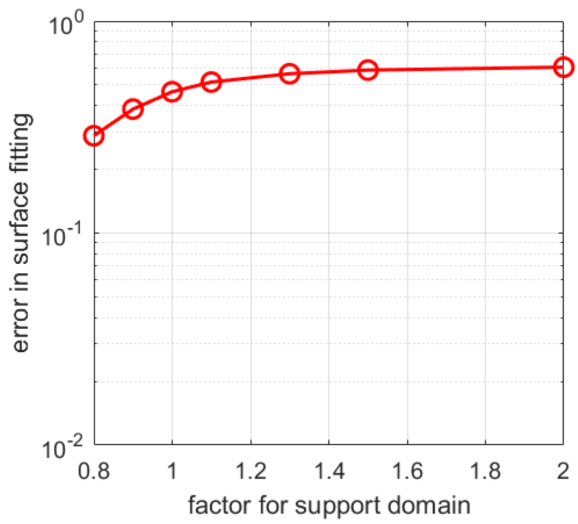

Figure 13

Case (2) We fit following planar function/surface and observe the accuracy of MLS based approach in Figure

$$u(x,z) = \frac{agk}{\omega}\frac{\cosh(k(d+z))}{\cosh(kd)}\sin(\omega t - kx)$$

at $t = 0$. Consider wave amplitude a = 0.2; g = 9.81; Wave frequency ω = 2; Wave number k = 0.5; Water depth d = 2 t=0.5 We plot for these parameters using MLS based approach in Figure 15 for same domain as in Figure (1)

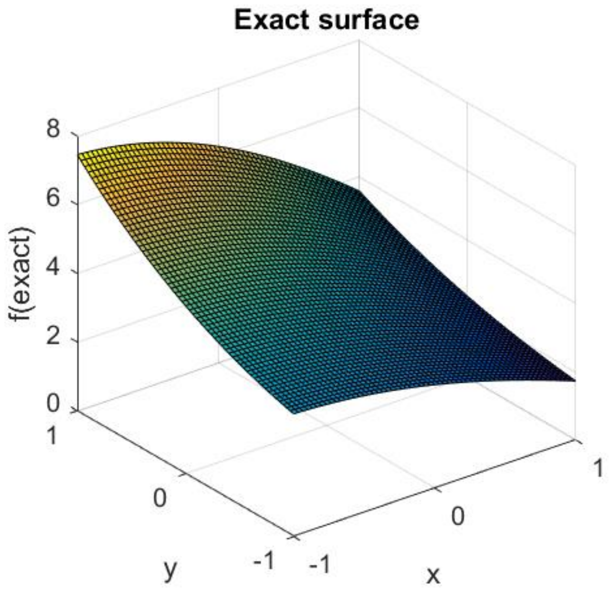

Figure 14

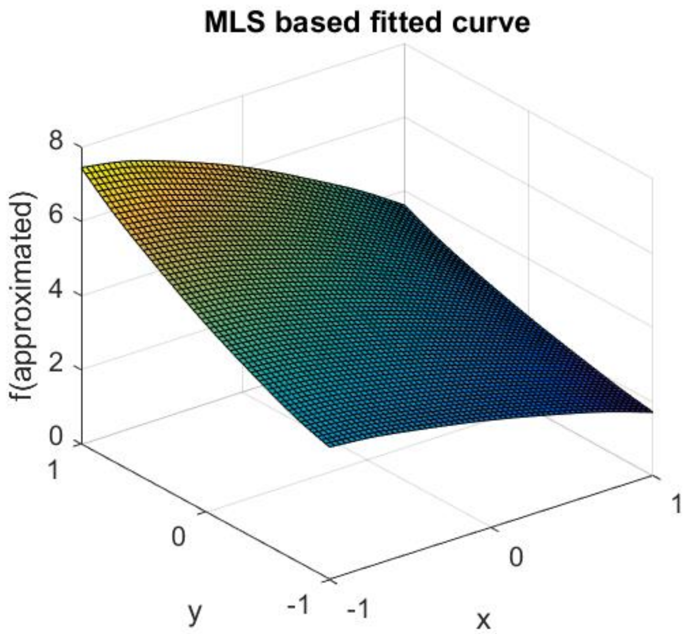

Figure 15

Figure 16 shows error in fitting given surface for different radius of support domain. MLS based approximation doesn't improve much the fitting results with increase in size support domain. Similar behaviour is observed for error in surface fitting for decrease in nodal spacing in Figure 17

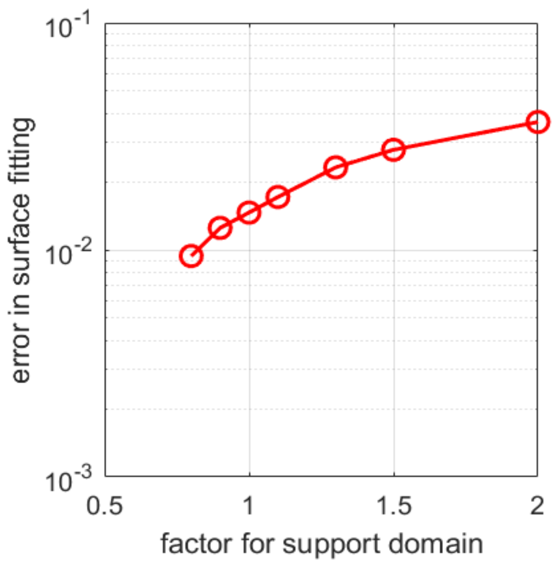

Figure 16

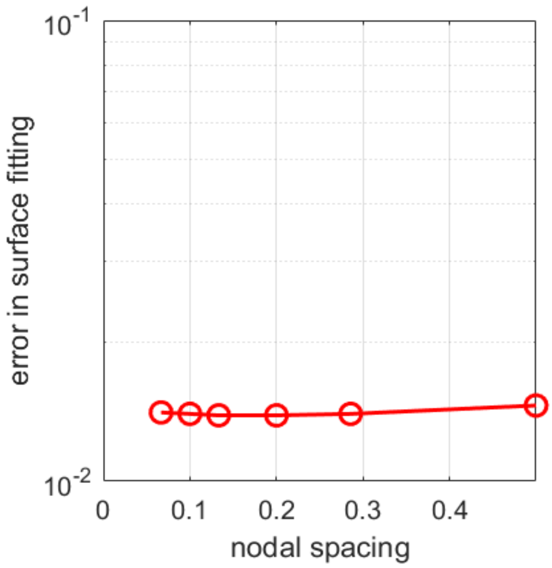

Figure 17

Appendix – A   Weight functions used in MATLAB code in Appendix B(i)

Cubic spline weight function (W1)

$$\widehat{W}(r_{ix}) = \begin{cases} \frac{2}{3} - 4r_{ix}^2 + 4r_{ix}^3 & r_{ix} \le 0.5 \\ \frac{4}{3} - 4r_{ix} + 4r_{ix}^2 - \frac{4}{3}r_{ix}^3 & 0.5 < r_{ix} \le 1 \\ 0 & r_{ix} > 1 \end{cases}$$

$$\widehat{W}(r_{iy}) = \begin{cases} \frac{2}{3} - 4r_{iy}^2 + 4r_{iy}^3 & r_{iy} \le 0.5 \\ \frac{4}{3} - 4r_{iy} + 4r_{iy}^2 - \frac{4}{3}r_{iy}^3 & 0.5 < r_{iy} \le 1 \\ 0 & r_{iy} > 1 \end{cases}$$

Quartic spline weight function (W2)

$$\widehat{W}(r_{ix}) = \begin{cases} 1 - 6r_{ix}^2 + 8r_{ix}^3 - 3r_{ix}^4 & 0 \le r_{ix} \le 1 \\ 0 & r_{ix} > 1 \end{cases}$$

$$\widehat{W}(r_{iy}) = \begin{cases} 1 - 6r_{iy}^2 + 8r_{iy}^3 - 3r_{iy}^4 & 0 \le r_{iy} \le 1 \\ 0 & r_{iy} > 1 \end{cases}$$

Appendix – B(i)   MATLAB code

```
close all
% 2D-MLS  with rectangular support domain (use 2D basis function only)
% By Nikhil Yewale
Nx = 5;Ny =5;                   % number of nodes in x and y direction
nnodes = Nx*Ny;                 % number of nodes (field nodes)
%dx =0.5; dy =0.5;              % nodal coordinates separated by dx and dy
[X,Y] = meshgrid(linspace(-1,1,Nx),linspace(-1,1,Ny));
Lx=2; Ly=2;                        % -1 to 1 is domain
%Lx = dx*(Nx-1) ; Ly = dy*(Ny-1);

nx = 61; ny = 61;    % number of coordinates in x and y direction for evaluation(resolution in
G.R.Liu book)
[x,y] = meshgrid(linspace(-1,1,nx),linspace(-1,1,ny));
npoints = nx*ny;   %number of coordinates for evaluation
dx1 = Lx/(nx-1);  dy1 = Ly/(ny-1);

Flag = 1;     % flag for weight function
              % 1 for cubic spline(W1)  , 2 for quartic spline(W2)
m = 3;          % number of terms for basis function

% rectangular support domain
dsx = zeros(nnodes,1);    dsy = zeros(nnodes,1);
% dsx and dsy according to code in introduction to meshfree programming Liu
% assures 25 nodes in support domain
factor = 1;                 % decides the size of support domain
for i =1:nnodes
    rx0 = abs(X(i)-1);   ry0 = abs(Y(i)-1);
    if(rx0 < abs(X(i)+1))
        rx0 = abs(X(i)+1);
    end
    if(ry0 < abs(Y(i)+1))
        ry0 = abs(Y(i)+1);
    end
    dsx(i,1) = factor*rx0;
    dsy(i,1) = factor*ry0;
end
```

```
scatter(x(:),y(:),'b.')  % denotes points
hold on
scatter(X(:),Y(:),'ro')   % denotes nodes
[PHI, DPHIDX, DPHIDY,WW] = MLS_Shape(m, nnodes, X,Y, npoints, x,y, dsx,dsy,Flag);
% maximum value of Shape function which we are plotting
max(PHI(:,13));   % index 13 denotes x = 0, y = 0 node
% summation of shape functions at nodal values
k = find(PHI(:,13)==max(PHI(:,13)));
%PHI(k,:);
summation_PHI = sum(PHI(k,:));  % actually valid for any k.. from 1 to nx*ny

% functions to be interpolated  F1 and F2
f1 = 1 + 2.*x.*x  + 3.*y.*y;   % FUNCTION VALUES AT ALL THE POINTS
f2 = (2*9.81*0.5/2).*(cosh(0.5*(2 + y))/cosh(0.5*2)).*(sin(-0.5*x + 2*0.5));
F1 = 1 + 2.*X.*X  + 3.*Y.*Y;    % FUNCTION VALUE ONLY AT NODES
F2 = (2*9.81*0.5/2).*(cosh(0.5*(2 + Y))/cosh(0.5*2)).*(sin(-0.5*X + 2*0.5));
f1_app = PHI*F1(:);            % approximated values at all the points
f2_app = PHI*F2(:);
f1_app = reshape(f1_app,nx,ny);
f2_app = reshape(f2_app,nx,ny);
summ_errf1 = 0;
summ_errf2 = 0;
for i = 1:nx*ny              % quantifying error
    summ_errf1 = summ_errf1 + (1/(nx*ny))*abs((f1_app(i) - f1(i))/f1(i));
    summ_errf2 = summ_errf2 + (1/(nx*ny))*abs((f2_app(i) - f2(i))/f2(i));
end
%--------------------Post-Processing
% Plot shape function at node 13.. i.e at x = 0 , y = 0
figure(2)
surf(x,y,reshape(PHI(:,13),nx,ny))
xlabel('x');      ylabel('y');      zlabel('\phi');
title('\phi for W1 function')
figure(3)
surf(x,y,reshape(DPHIDX(:,13),nx,ny))
xlabel('x');      ylabel('y');      zlabel('\partial\phi/\partialx')
title('\partial\phi/\partialx for W1 function')
figure(4)
surf(x,y,reshape(DPHIDY(:,13),nx,ny))
xlabel('x');      ylabel('y');      zlabel('\partial\phi/\partialy')
title('\partial\phi/\partialy for W1 function')
figure(5)
surf(x,y,reshape(WW(:,13),nx,ny))
xlabel('x');      ylabel('y');      zlabel('Weight function')
title('Cubic spline weight function')
figure(6)
surf(x,y,f1)
xlabel('x');      ylabel('y');      zlabel('f(exact)')
title('Exact surface')
figure(7)
surf(x,y,f1_app)
xlabel('x');      ylabel('y');      zlabel('f(approximated)')
title('MLS based fitted curve')
figure(8)
surf(x,y,f2)
xlabel('x');      ylabel('y');      zlabel('f(exact)')
title('Exact surface')
figure(9)
surf(x,y,f2_app)
xlabel('x');      ylabel('y');      zlabel('f(approximated)')
title('MLS based fitted curve')
```

Function file for shape function

```
function [PHI, DPHIDX, DPHIDY,WW] = MLS_Shape(m, nnodes, X,Y, npoints, x,y, dsx,dsy,Flag)

% output variables initialized to zeros
W = zeros(1,nnodes);            dWx = zeros(1,nnodes);
dWy = zeros(1,nnodes);          dWxx = zeros(1,nnodes);
dWyy = zeros(1,nnodes);         dWxy = zeros(1,nnodes);

XI = zeros(1,nnodes);            YI = zeros(1,nnodes);
```

```
WW = zeros(npoints,nnodes);
PHI = zeros(npoints,nnodes);     DPHIDX = zeros(npoints,nnodes);
DPHIDY = zeros(npoints,nnodes);

% LOOP OVER ALL EVALUATION POINTS TO CALCULATE VALUE OF SHAPE FUNCTION \partial\phi(X)
for j = 1:npoints
    % DETERMINE WEIGHT FUNCTIONS AND THEIR DERIVATIVES AT EVERY NODE of
    % evaluation points
    for i = 1:nnodes
        [W(i),dWx(i),dWy(i),dWxx(i),dWyy(i),dWxy(i)] =
weight_function(X(i),Y(i),x(j),y(j),dsx(i),dsy(i),Flag);
        XI(1,i) = X(i);
        YI(1,i) = Y(i);
    end
    WW(j,:) = W;
    % EVALUATE BASIS p, B MATRIX AND THEIR DERIVATIVES
    if (m == 1)
        p = [ones(1, nnodes)];
        pxy   = [1];
        dpdx  = [0];
        dpdy = [0];

        B    = p .* [W];    % B matrix... based on nodes in support domain
        DBdx   = p .* [dWx];
        DBdy  = p .* [dWy];
    elseif (m == 3)    % linear basis function
        p = [ones(1, nnodes);XI;YI]; %  polynomial of nodes in support domain
        pxy   = [1; x(j);y(j)];  % polynomial of considered evaluation point.. this is basis
function
        dpdx  = [0; 1;0];
        dpdy = [0; 0;1];

        B    = p .* [W; W;W];     % B matrix... based on nodes in support domain
        DBdx   = p .* [dWx; dWx;dWx];
        DBdy  = p .* [dWy; dWy;dWy];
    elseif (m == 6)           % quadratic basis function
        p = [ones(1, nnodes); XI;YI; XI.*XI;XI.*YI;YI.*YI];
        pxy   = [1; x(j); y(j);x(j)*x(j);x(j)*y(j);y(j)*y(j)];
        dpdx  = [0; 1; 0;2*x(j);y(j);0];
        dpdy = [0;0;1;0;x(j);2*y(j)];

        B    = p.* [W; W; W; W; W; W];  % B matrix... based on nodes in support domain
        DBdx   = p.* [dWx; dWx;dWx;dWx; dWx;dWx];
        DBdy  = p.* [dWy; dWy;dWy;dWy; dWy;dWy];
    else
        error('Re-enter the number of monomials');
    end

     % EVALUATE MATRICES A AND ITS DERIVATIVES
    A    = zeros(m, m);
    DAdx   = zeros(m, m);
    DAdy= zeros(m, m);
    for i = 1 : nnodes
        pp = p(:,i) * p(:,i)';    % [1 x y; x x*x x*y;y x*y y*y] matrix evaluated at nodes in
support domain
        A    = A + W(i)* pp;
        DAdx  = DAdx  + dWx(i)*pp;
        DAdy = DAdy + dWy(i)*pp;
    end

    AInv = inv(A);
    gam   = AInv*pxy;       % gamma---->    A*gamma = p(x)
    PHI(j,:) = gam'*B;     %  shape function

    dgamdx   = AInv*(dpdx-DAdx* gam);
    DPHIDX(j,:) = dgamdx'*B + gam' * DBdx; % first order derivatives of shape function with
                                               respect to x

    dgamdy = AInv*(dpdy -DAdy* gam);
    DPHIDY(j,:) = dgamdy'*B + gam' * DBdy; % first order derivatives of shape function to y
    % Similarly, second order derivatives in x and y direction can also be
    % calculated if required
end
end
```

Function file for weight function with rectangular support domain

```
function [W,dWx,dWy,dWxx,dWyy,dWxy] = weight_function(X,Y,x,y,dsx,dsy,Flag)

% X,Y -- node point              x,y -- evaluation point
rxi = abs(x - X)/dsx;     ryi = abs(y - Y)/dsy;

if abs(x-X) < 1e-20
    drdx = 0;
else
    drdx = (x-X)/(abs(x-X)*dsx);
end
if abs(y-Y) <  1e-20
    drdy = 0;
else
    drdy = (y-Y)/(abs(y-Y)*dsy);
end


switch(Flag)
    case 1   % cubic spline(W1)

        if (rxi <= 0.5)
            Wx = (2/3) - 4*rxi*rxi + 4*rxi*rxi*rxi;
            dWxdx = (-8*rxi + 12*rxi*rxi)*drdx;
            ddWxx = (-8 + 24*rxi)*(drdx^2);
        elseif (rxi > 0.5 && rxi <= 1)
            Wx = (4/3) - 4*rxi + 4*rxi*rxi - (4/3)*rxi*rxi*rxi;
            dWxdx = (-4 + 8*rxi - 4*rxi*rxi)*drdx;
            ddWxx = (8  - 8*rxi)*(drdx^2);
        elseif(rxi > 1)
            Wx = 0;
            dWxdx = 0;
            ddWxx = 0;
        end

        if (ryi <= 0.5)
            Wy = (2/3) - 4*ryi*ryi + 4*ryi*ryi*ryi;
            dWydy = (-8*ryi + 12*ryi*ryi)*drdy;
            ddWyy = (-8 + 24*ryi)*(drdy^2);
        elseif(ryi > 0.5 && ryi <= 1)
            Wy = (4/3) - 4*ryi + 4*ryi*ryi - (4/3)*ryi*ryi*ryi;
            dWydy = (-4 + 8*ryi - 4*ryi*ryi)*drdy;
            ddWyy = (8 - 8*ryi)*(drdy^2);
        elseif(ryi > 1)

            Wy = 0;
            dWydy = 0;
            ddWyy = 0;
        end

        W = Wx*Wy;
        dWx = dWxdx*Wy;
        dWy = dWydy*Wx;
        dWxx = ddWxx*Wy;
        dWyy = ddWyy*Wx;
        dWxy = dWxdx*dWydy;

    case 2   % quartic spline (W2)

        if  (rxi>=0 && rxi <= 1)
            Wx = 1 - 6*rxi*rxi + 8*rxi*rxi*rxi - 3*rxi*rxi*rxi*rxi;
            dWxdx = (-12*rxi + 24*rxi*rxi - 12*rxi*rxi*rxi)*drdx ;
            ddWxx = (-12 + 48*rxi - 36*rxi*rxi)*(drdx^2);
        elseif(rxi>1)
            Wx = 0;
            dWxdx = 0;
            ddWxx = 0;
        end

        if(ryi>=0 && ryi <= 1)
            Wy = 1 - 6*ryi*ryi + 8*ryi*ryi*ryi - 3*ryi*ryi*ryi*ryi;
            dWydy = (-12*ryi + 24*ryi*ryi - 12*ryi*ryi*ryi)*(drdy) ;
            ddWyy = (-12 + 48*ryi - 36*ryi*ryi)*(drdy^2);
```

```
elseif(ryi > 1)
            Wy = 0;
            dWydy = 0;
            ddWyy = 0;
        end
        W = Wx*Wy;
        dWx = dWxdx*Wy;
        dWy = dWydy*Wx;
        dWxx = ddWxx*Wy;
        dWyy = ddWyy*Wx;
        dWxy = dWxdx*dWydy;

end
end
```

Appendix – B(ii) Partition of Unity Property

Following table shows partition of unity property of MLS based shape function when W1 weight function is used. Similarly, the same can be shown using W2 weighting function also.

Node	Coordinate of node	ϕ (using W1)
1	[-1,-1]	0.009167043911272
2	[-1,-0.5]	0.020371208691716
3	[-1,0]	0.036668175645088
4	[-1,0.5]	0.020371208691716
5	[-1,1]	0.009167043911272
6	[-0.5,-1]	0.020371208691716
7	[-0.5,-0.5]	0.045269352648257
8	[-0.5,0]	0.081484834766863
9	[-0.5,0.5]	0.045269352648257
10	[-0.5,1]	0.020371208691716
11	[0,-1]	0.036668175645088
12	[0,-0.5]	0.081484834766863
13	[0,0]	0.146672702580353
14	[0,0.5]	0.081484834766863
15	[0,1]	0.036668175645088
16	[0.5,-1]	0.020371208691716
17	[0.5,-0.5]	0.045269352648257
18	[0.5,0]	0.081484834766863
19	[0.5,0.5]	0.045269352648257
20	[0.5,1]	0.020371208691716
21	[1,-1]	0.009167043911272
22	[1,-0.5]	0.020371208691716
23	[1,0]	0.036668175645088
24	[1,0.5]	0.020371208691716
25	[1,1]	0.009167043911272

$$\sum \boldsymbol{\phi} = \mathbf{1.0000}$$