Tutorial – 3

OE6020 – Meshfree methods applied to hydrodynamics **Roll No : AM17S015**

Least Square finite difference (LSFD) method **Name : Yewale Nikhil Janardan**

Problem: Consider the problem of a 2D Poisson equation in a square domain ($0 \le x \le 1, 0 \le y \le 1$) .The governing equation and boundary condition are defined by

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -2\pi^2 sin(\pi x).\sin(\pi y)$$

Boundary condition: $u = 1 + x$ on $\partial\Omega$.
The analytical solution for this problem is $u(x, y) = 1 + x + \sin(\pi x)\sin(\pi y)$

Solution: The functional value near a node 0 can be approximated by functional value and it's derivatives at node 0 by using the 2D Taylor expansion as follows,

$$\phi = \phi_0 + \Delta x\left(\frac{\partial\phi}{\partial x}\right)_0 + \Delta y\left(\frac{\partial\phi}{\partial y}\right)_0 + \frac{1}{2}(\Delta x)^2\left(\frac{\partial^2\phi}{\partial x^2}\right)_0 + \frac{1}{2}(\Delta y)^2\left(\frac{\partial^2\phi}{\partial y^2}\right)_0 + (\Delta x\Delta y)\left(\frac{\partial^2\phi}{\partial x\partial y}\right)_0$$

$$\frac{1}{6}(\Delta x)^3\left(\frac{\partial^3\phi}{\partial x^3}\right)_0 + \frac{1}{6}(\Delta y)^3\left(\frac{\partial^3\phi}{\partial y^3}\right)_0 + \frac{1}{2}(\Delta x)^2(\Delta y)\left(\frac{\partial^2\phi}{\partial x^2\partial y}\right)_0 + \frac{1}{2}(\Delta y)^2(\Delta x)\left(\frac{\partial^2\phi}{\partial y^2\partial x}\right)_0 + \cdots$$

The above functional approximation in the neighbourhood of node 0 can be expressed using optimal approximation of derivative vector (say ***b***) at node 0 as reported by *H.Ding et.al*[1]

$$\boldsymbol{b} = d\phi = (S^T S)^{-1} S^T \Delta\phi \qquad \textbf{\textit{(1)}}$$

where $S_j^T = [\Delta x_j, \Delta y_j, \frac{1}{2}(\Delta x_j)^2, \frac{1}{2}(\Delta y_j)^2, (\Delta x_j\Delta y_j), \frac{1}{6}(\Delta x_j)^3, \frac{1}{6}(\Delta y_j)^3, \frac{1}{2}(\Delta x_j)^2(\Delta y_j), \frac{1}{2}(\Delta y_j)^2(\Delta x_j)]$

$$d\phi_j = \left[\left(\frac{\partial\phi}{\partial x}\right)_0, \left(\frac{\partial\phi}{\partial y}\right)_0, \left(\frac{\partial^2\phi}{\partial x^2}\right)_0, \left(\frac{\partial^2\phi}{\partial y^2}\right)_0, \left(\frac{\partial^2\phi}{\partial x\partial y}\right)_0, \left(\frac{\partial^3\phi}{\partial x^3}\right)_0, \left(\frac{\partial^3\phi}{\partial y^3}\right)_0, \left(\frac{\partial^2\phi}{\partial x^2\partial y}\right)_0, \left(\frac{\partial^2\phi}{\partial y^2\partial x}\right)_0\right]$$

Introducing weighting function in least square approximation assumes square of the errors to be uniformly distributed across supporting domain. The usage of weighting function serves as a preconditioner in the estimation of derivative vector in such as way that,

$$d\phi = (S^T W S)^{-1} S^T W \Delta\phi \qquad \textbf{\textit{(2)}}$$

where *W* is an *n* x *n* diagonal matrix at *n* supporting points.

W = diag $(W_1, W_2, W_3, \dots, W_n)$. To avoid the singularity of $S^T S$ matrix, H.Ding[1] proposed scaling matrix to scale the local distance $(\Delta x_j, \Delta y_j)$ using radius of support domain d_0.

Scaling matrix $D = diag[\frac{1}{d_0}, \frac{1}{d_0}, \frac{1}{d_0^2}, \frac{1}{d_0^2}, \frac{1}{d_0^3}, \frac{1}{d_0^3}, \frac{1}{d_0^3}, \frac{1}{d_0^3}]$.

Thus equation (2), is rewritten using scaling as

$$d\phi = D(\bar{S}^T W \bar{S})^{-1} \bar{S}^T W \Delta\phi \qquad \textbf{\textit{(3)}}$$

where $\bar{S} = SD$

Results: Following figures show a close resemblance between analytical solution and LSFD based solution for given 2D Poisson equation. W1 weight function gives slightly superior results as compared to W0,W2,W3,W4 weight functions, hence are shown in figure below. Refer Appendix –A for weighting functions

21 x 21 Grid

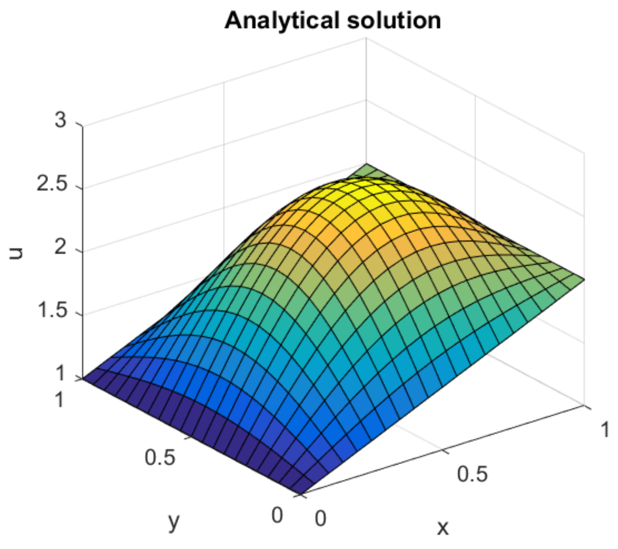

Figure 1.

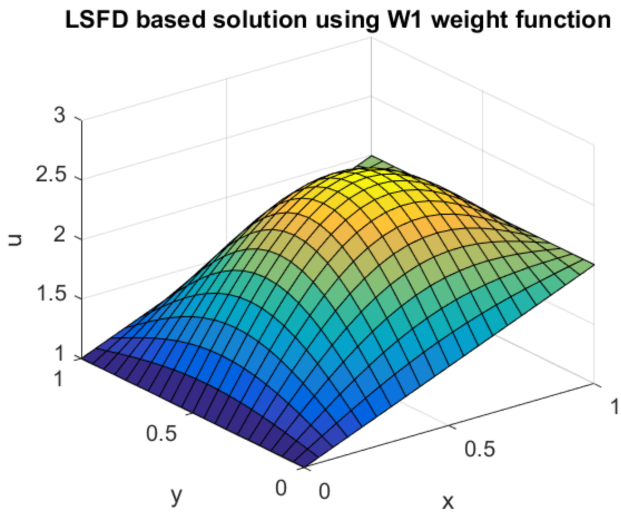

Figure 2

51 x 51 Grid

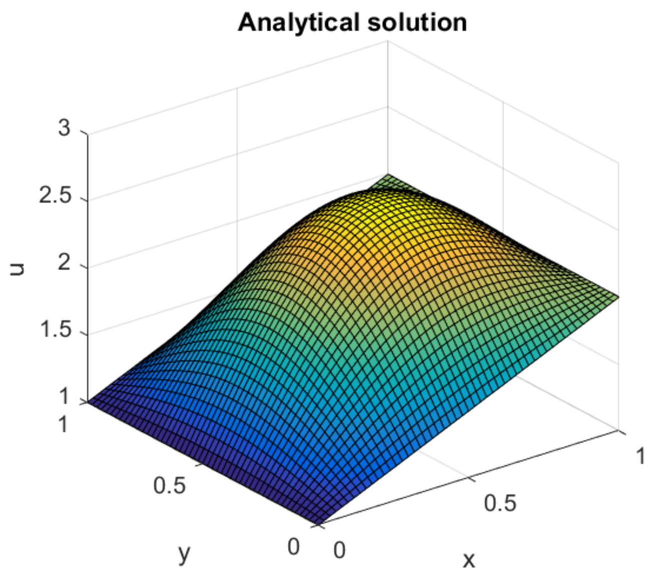

Figure 3

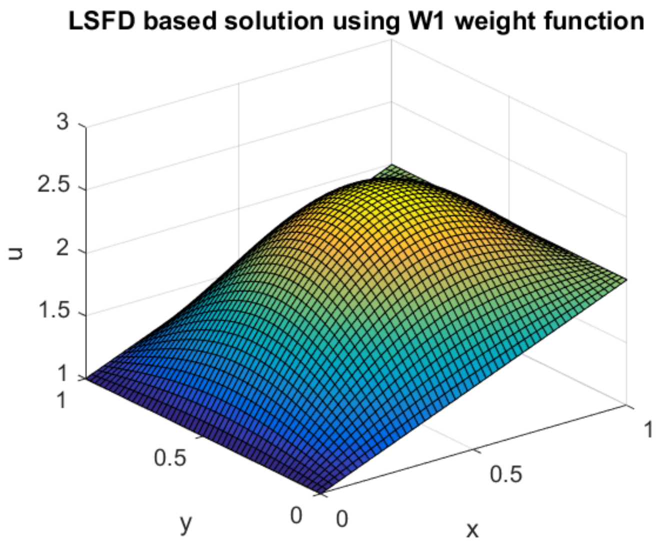

Figure 4

76 x 76 Grid

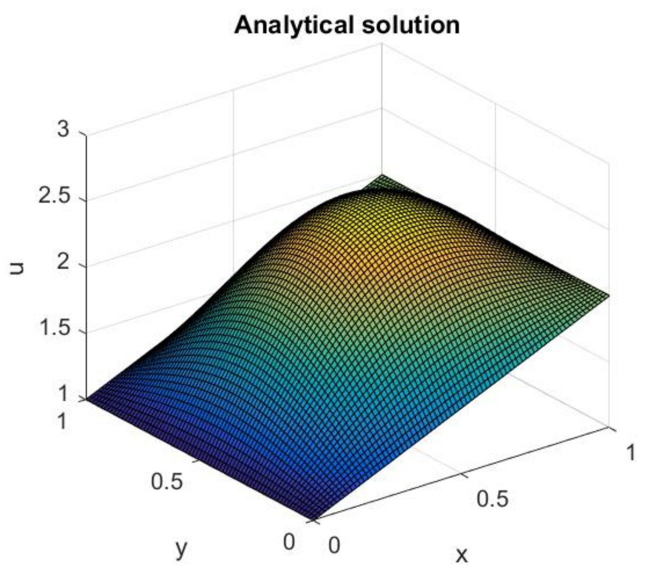

Figure 5

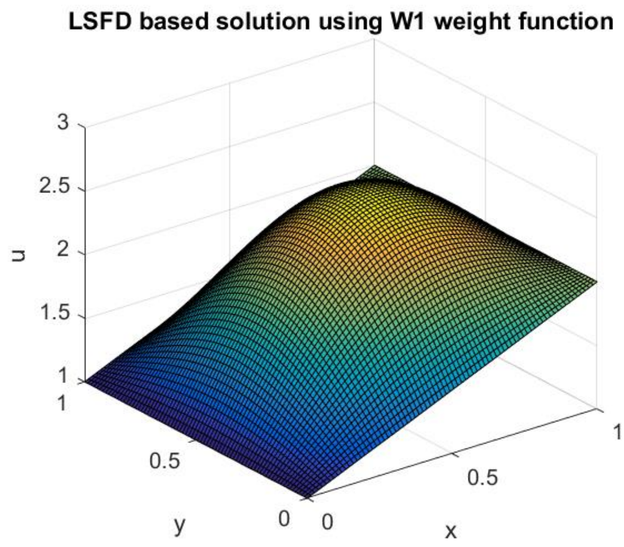

Figure 6

101 x 101 Grid

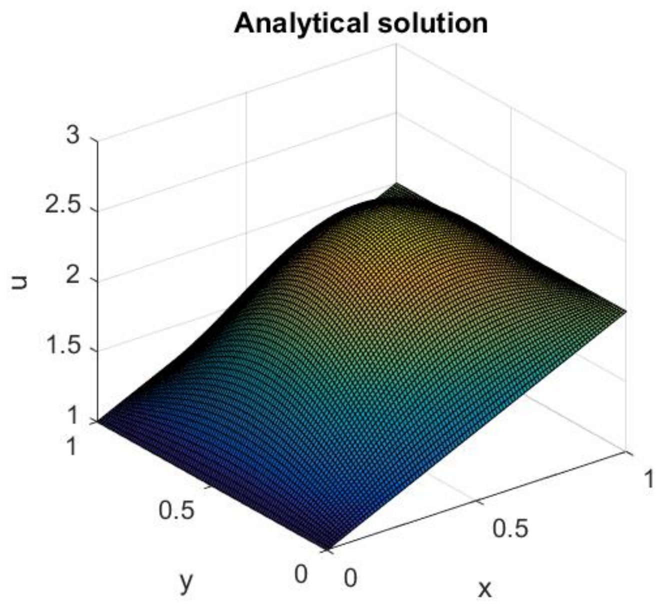

Figure 7

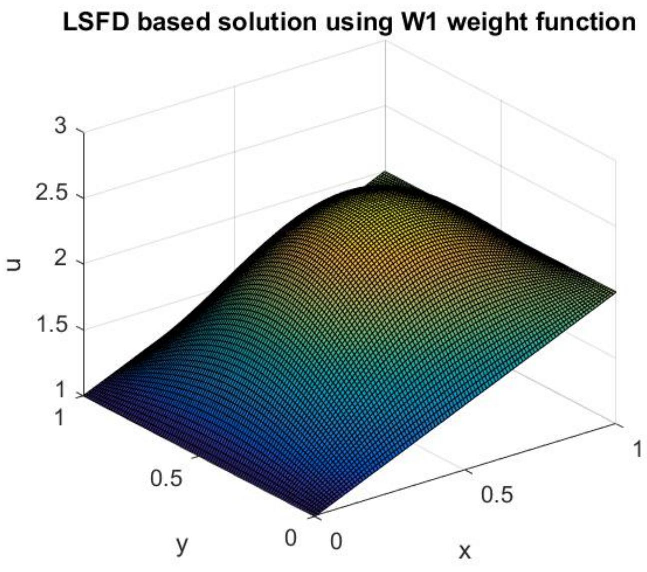

Figure 8

Table 1. Comparison of $log_{10}(err)$ for the solution of given Poisson equation with different weighting functions

Grid Spacing →	**0.0500**	**0.0200**	**0.0133**	**0.0100**
LSFD W0	-2.4056	-3.1930	-3.5520	-3.7415
W1	**-2.7060**	**-3.4945**	**-3.8507**	**-4.0400**
W2	-2.4266	-3.2132	-3.5720	-3.7623
W3	-2.6662	-3.4548	-3.8110	-4.0007
W4	-2.5599	-3.3440	-3.7019	-3.8975

As noted in Table 1 and Figure 9. For a given grid size, W1 weight function yields superior performance as compared to other weighting functions. Also, as seen from Figure 9, all the weight functions converge with same convergence rate (2.0104). This also means, weighting functions have no significant effect on convergence rate, but it does help in improving numerical accuracy slightly.

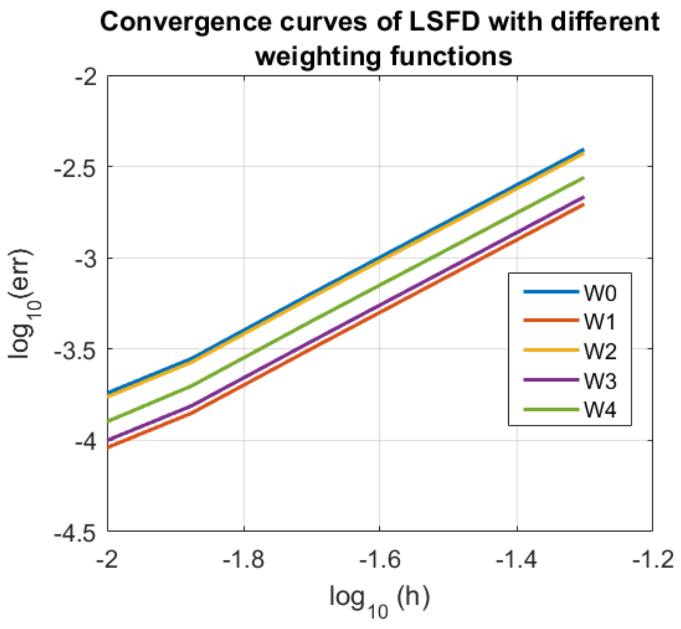

Figure 9

References:

1) Development of least square based two dimensional finite difference schemes and their application to simulate natural convection in a cavity. H. Ding, C.Shu , K.S.Yeo , D.Xu .Computer & Fluids 33(2004) 137-154. doi:10.1016/S0045-7930(03)00036-7

Appendix – A

Weight functions used in LSFD based solution for given Poisson equation are as follows:

i) $W_{0i} = 1 \quad (equivalent\ to\ no\ weighting)$

ii) $W_{1i} = \sqrt{\frac{4}{\pi}}(1 - r_i^2)^4$

iii) $W_{2i} = 1/r_i$

iv) $W_{3i} = 1 - 6r_i^2 + 8r_i^3 - 3r_i^4$

v) $W_{4i} = 1/r_i^4$

Appendix – B (i) MATLAB Code for LSFD based solution for given 2D-Poisson equation

```
Lx =1; Ly = 1;          % length of domain
Nx = 101;Ny = 101;      % grid points in x andy direction
[x,y] = meshgrid(linspace(0,Lx,Nx),linspace(0,Ly,Ny));
nx = length(x); ny = length(y);
g = (-2*pi*pi).*sin(pi*x).*sin(pi*y);   % function on RHS of given Poisson equation
f = 1 + x + sin(pi*x).*sin(pi*y);   % analytical solution

dx= Lx/(Nx-1);   dy = Ly/(Ny-1);
Flag = 2;    % flag for kernel function
                      % 1-W0  , 2-W1, 3-W2, 4-W3, 5-W5

u = zeros(nx,ny);    % field variable set to zeros
% Apply boundary conditions  .... u = 1+x on boundary
u(1,:) = 1+x(1,:);                u(end,:) = 1+x(end,:);
u(2:end-1,1) = 1+x(2:end-1,1);   u(2:end-1,end) = 1+x(2:end-1,end);

% matrix of boundary(1)/non-boundary point(0)
U = zeros(nx,ny);
% Apply boundary conditions
U(1,:) = 1;                U(end,:) = 1;
U(2:end-1,1) = 1;   U(2:end-1,end) = 1;
U = U(:);
u =  u (:);   u = [u,U];  % first column ..value...... second column boundary or not
% 1 for boundary point... 0 for interior point
r0 = zeros(nx*ny,1);
index = zeros(nx*ny,1);
for i = 1:nx*ny
    d0 = zeros(1,nx*ny);
    index(i) = i;
    for j = 1:nx*ny
        dx = x(i) - x(j);   dy = y(i)-y(j);
        d0(j) = sqrt(dx^2 + dy^2);
    end
    r0(i) = 0.03*max(d0);  % radius of support domain
end

k = 1;  % index iterating on 1:nn for A matrix n B vector
u = [u,index];   % third column has index of the point/node in grid
indexA = find(u(:,2)==0); % indexes for A values actually\
nn = length(indexA);
A = zeros(nn,nn);
B = zeros(nn,1);

for i = 1:nx*ny
    if(u(i,2) == 0)    % select interior points only
        count = 0;
        Wii = [];   Sii = [];  Uii = [];
        for j = 1:nx*ny
            dx = x(i) - x(j);   dy = y(i)-y(j);
            if(dx~=0 || dy~=0)                      % excluding the point itself from
neighbourhood
                rj = sqrt(dx^2 + dy^2)/r0(i);
                Wj = weight_func(rj,Flag);
                sj =
[dx;dy;0.5*(dx^2);0.5*(dy^2);dx*dy;(dx^3)/6;(dy^3)/6;0.5*(dx^2)*dy;0.5*dx*(dy^2)];
                if(Wj~=0)
                    Wii = [Wii,Wj];   % to construct weight matrices
                    Sii = [Sii,sj];
                    Uii = [Uii;u(j,1),u(j,2),u(j,3)];  % field values of node along with
boundary, index
                    count = count + 1;  % counts number of supporting points in the domain of
i'th point
                end
            end
        end

        if(count<9)
            error('Less number of points in support domain than 9');
        end

        Wi = diag(Wii);   % diagonal matrix of weights
        D = [r0(i),r0(i),(r0(i))^2,(r0(i))^2,(r0(i))^2,(r0(i))^3,(r0(i))^3,(r0(i))^3,(r0(i))^3];
        % Sii is of size 9 x n;  Sii' - n x 9...         Wi--> n x n

        D= inv(diag(D));   % scaling matrix inverted
        Sii = Sii'*D;    %( 400 x 9 )
        sii = Sii';      %(   9 x n)
        C = D*inv(sii*Wi*Sii)*(sii*Wi); % coefficient matrix of 9 x n (where n is number of
unknowns)

% compute A and B matrix for given poisson equation
        while (k <= nn)
            [a,b] = computeAB(C(3,:),C(4,:),g(i),Uii,indexA,nn);
            A(k,:) = a;
            A(k,k) = -1*(sum(C(3,:)) + sum(C(4,:)));
            B(k,1) = b;
            k = k+1;
            break;
        end
    end
end

sol = A\B;
u = reshape(u(:,1),Nx,Ny);
sol =reshape(sol,Nx-2,Ny-2);
u(2:end-1,2:end-1) = sol;
rel_err = norm(u(:)-f(:))/norm(f(:))
kk = log10(rel_err)
figure(1)
surf(x,y,f)
xlabel('x'); ylabel('y'); zlabel('u');
title('Analytical solution')
figure(2)
surf(x,y,u)
xlabel('x');ylabel('y'); zlabel('u');
title('LSFD based solution using W1 weight function')
```

Appendix – B (ii) weight function

```
function Wi = weight_func(ri,Flag)
switch(Flag)
    case 1
        if (ri>=0 && ri<=1)   %W0
            Wi = 1;  % no weighting
        else
            Wi = 0;
        end
    case 2                    %W1
        if(ri>=0 && ri<=1)  % best as per literature
            Wi = sqrt(4/pi)*((1-ri^2)^4);
        else
            Wi = 0;
        end
    case 3                   %W2
        if (ri>=0 && ri<=1)
            Wi = 1/ri;
        else
            Wi=0;
        end
    case 4                  %W3
        if(ri>=0 && ri<=1)  % same as quartic spline
            Wi = 1 - (6*(ri^2)) + (8*(ri^3)) - (3*(ri^4));
        else
            Wi = 0;
        end
    case 5                  %W4
        if(ri>=0 && ri<=1)
            Wi = 1/ri^4;
        else
            Wi = 0;
        end
end
end
```

Appendix – B (iii) compute A matrix and B vector in final Ax = B form

```
function [a,b] = computeAB(C3,C4,g,Uii,indexA,nn)
a = zeros(1,nn);

b = g - C3*Uii(:,1) - C4*Uii(:,1);  % vector with g , and contributions from known boundary
conditions

for i = 1:length(C3)    % OR C4 or Uii... doesn't matter

    if (Uii(i,2)==0) % considering only interior points
        % fill the entries of A matrix... in this function ..it fills each

        j = find(indexA==Uii(i,3));
        a(j) = C3(i) + C4(i);

    end

end

end
```