APPM7440: Radial Basis Functions - Finite Differences — April 30, 2015

Lab Report 4

Homework 4 — *Author: Prasanth Prahladan(100817764)*

1 Time-Independent PDE

Attempt to reproduce Figure 4.4(a) in text book, with a node-distribution similar to that in Figure 4.3(c).

1.1 Derivation of Laplacian

$$\begin{aligned}
\Phi(r) &= \Phi(\sqrt{(}x^2+y^2)) \\
\Delta\Phi &= \frac{\partial^2}{\partial^2 x^2} + \frac{\partial^2}{\partial^2 y^2} \\
\frac{\partial^2\Phi}{\partial^2 x^2} &= (\frac{\partial r}{\partial x})^2(\frac{\partial^2\Phi}{\partial^2 r^2} - \frac{1}{r}\frac{\partial\Phi}{\partial r}) \\
\frac{\partial^2\Phi}{\partial^2 y^2} &= (\frac{\partial r}{\partial y})^2(\frac{\partial^2\Phi}{\partial^2 r^2} - \frac{1}{r}\frac{\partial\Phi}{\partial r})
\end{aligned}$$

We thus derive, the relationship for the Laplacian of the RBF

$$\Delta\Phi = (\frac{\partial^2}{\partial^2 x^2} + \frac{\partial^2}{\partial^2 y^2})\Phi \tag{1}$$

$$= (\frac{\partial^2}{\partial^2 r^2} - \frac{1}{r}\frac{\partial}{\partial r})\Phi \tag{2}$$

For the different families of RBF-functions we derive the following

1. Gaussian RBF

$$\Phi(r) = e^{-(\epsilon r)^2} \tag{3}$$

$$\Delta\Phi(r) = 4\epsilon^4 r^2 e^{-(\epsilon r)^2} \tag{4}$$

2. Multi-quadratic(MQ)

$$\Phi(r) = \sqrt{1+(\epsilon r)^2} \tag{5}$$

$$\Delta\Phi(r) = -(\epsilon r)^2\big(1+(\epsilon r)^2\big)^{-3/2} \tag{6}$$

3. Inverse Quadratic(IQ)

$$\Phi(r) = \frac{1}{1+(\epsilon r)^2} \tag{7}$$

$$\Delta\Phi(r) = 8\epsilon^4\frac{r^2}{\big(1+(\epsilon r)^2\big)^3} \tag{8}$$

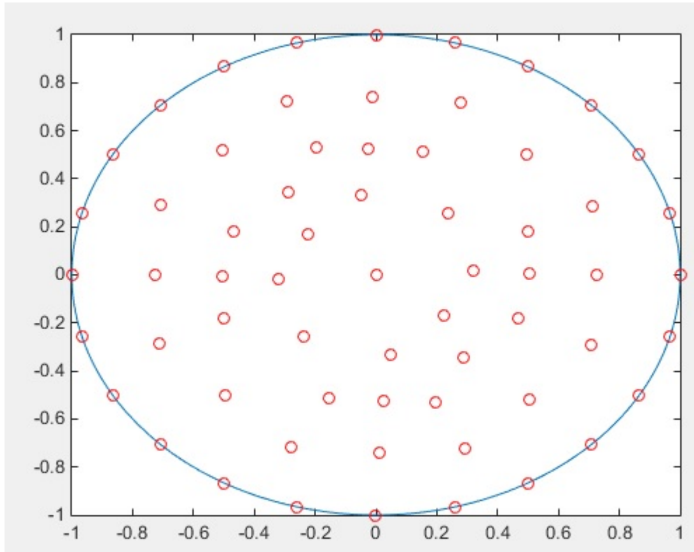
Figure 1: Node Distribution in Disc

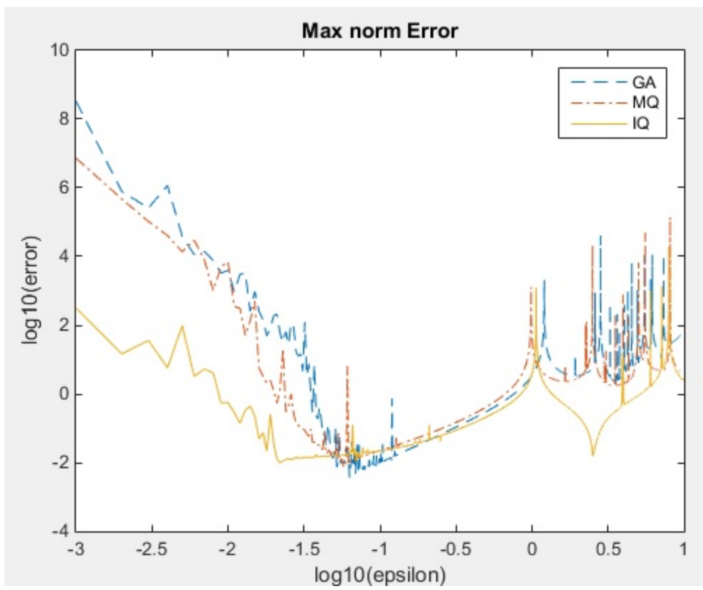

Figure 2: Max Norm Error variation with epsilon

1.2 Plots generated

1.3 Matlab Implementation

```
% Code to reproduce Fig 4.4(a) using Matlab PDE Toolbox - Mesh Generating
% function to create Mesh in Fig 4.3(c)

clear all
close all

bndryN = 16;
```

```
intrN = 48;
totalN = bndryN + intrN;

%% ================================================================================
%% ================================================================================
% Determine the Test Points
r = 0.3;
[p,e,t] = initmesh('circleg','Hmax',r); % create Circular Mesh with 64 points
tstX = p(1,:)';
tstY = p(2,:)';
%% ================================================================================
% Determine the Points of Grid
r = 0.35;
[p,e,t] = initmesh('circleg','Hmax',r); % create Circular Mesh with 64 points

% p : DESIRED CIRCULAR MESH for RBF INTERPOLATION.
gridX = p(1,:)';
gridY = p(2,:)';

% -------------------------------------------------------------
% plot Unit Circle
theta = 0:0.01:2*pi;
x = cos(theta);
y = sin(theta);
fig1 = figure(1)
plot(x,y)
hold on
% plot the mesh-points
scatter(gridX, gridY,'r')
hold off

%% ================================================================================
    %% ================================================================================
    %[g f] = evalPDEatPoints(p);
    % -------------------------------------------------
    % g = u(x,y) =  A ./ (A + (x - a).^2 + b* y.^2);
    % f = Laplacian[u(x,y)] = -2*A*(A + (x - a).^2 + b* y.^2).^(-3)
    % .*(A(b+1) + (x-a)^2(b-3) + y^2(b -3b^2))
    % -------------------------------------------------
    % Boundary Constraint
    g = @(X, Y) (100 ./(100 + (X - 0.2).^2 + 2* Y.^2) );
    % Interior Constraint
    f = @(X,Y) (-2*100*(100*(2+1) + (2-3)*(X-0.2).^2 + (2 - 3*2^2)*Y.^2).*(100 + (X - 0.2).^2 + 2*Y.^2).^(-3));


%% ================================================================================
%% ================================================================================
% Solving PDE using RBF


epsilons = 0.001:0.001:10;
maxError = zeros(length(epsilons),3);

for i = 1:3
```

```
    switch i
        case 1
            rbf = 'GA'
        case 2
            rbf = 'IQ'
        case 3
            rbf = 'MQ'
    end

for count = 1:length(epsilons)
    ep = epsilons(count);

    %% ==========================================================================
    %% ==========================================================================
    % Kansa's Formulation
    % ----------------------------------------------------------
    % Determine Vectors

    gridR = gridX.^2 + gridY.^2;

    % Identify index of points that are located on UNIT CIRCLE BOUNDARY
    bdryID = find(gridR==1);
    intrID = find(gridR<1);

    % Rearrange Points such that first values are on Boundary and remaining
    % are in the interior

    newOrder = [bdryID; intrID];
    gridX = gridX(newOrder);
    gridY = gridY(newOrder);

    numBoundaryPts = length(bdryID);

    % Evaluate Boudary Constraint
    gridX_BND = gridX(1:numBoundaryPts);
    gridY_BND = gridY(1:numBoundaryPts);
    gGrid = g(gridX_BND,gridY_BND);

    % Evaluate Interior Constraint
    gridX_INT = gridX(numBoundaryPts +1:end);
    gridY_INT = gridY(numBoundaryPts +1:end);
    fGrid = f(gridX_INT,gridY_INT);

    % ----------------------------------------------------------
    % Compute R for the A matrix

    [x1, x2] = meshgrid(gridX);
    [y1, y2] = meshgrid(gridY);
    d2 = (x1 - x2).^2 + (y1 - y2).^2;
    r = sqrt(d2);

    % Generate the A matrix for points on Boundary
    A = fi(rbf,ep, r(1:numBoundaryPts,:) );
    % Generate Laplacian_A matrix for points in Interior
    LA = Lfi(rbf,ep, r(numBoundaryPts+1:end,:) );
```

```
    % Matrix for Kansa's Method
    Ahat = [A; LA];

    % Evaluate the weighting coefficients
    lambda = Ahat\[gGrid;fGrid];

    % Grid points tst_i - grid_j
    [tX gX] = ndgrid(tstX, gridX);
    [tY gY] = ndgrid(tstY, gridY);

    tstR2 = (tX-gX).^2 + (tY - gY).^2;
    tstR = sqrt(tstR2);


    tstA = fi(rbf,ep,tstR);

%     fprintf('tstU_rbf\n')
    tstU_rbf = tstA*lambda;
    size(tstU_rbf);

%     fprintf('tstU_true\n')
    tstU_true = g(tstX,tstY);
    size(tstU_true);

    error = tstU_true - tstU_rbf;
    maxError(count,i) = max(abs(error));
end

end
fig2 = figure(2)
plot(log10(epsilons),log10(maxError(:,1)),'--',log10(epsilons),log10(maxError
    (:,2)),'-.', log10(epsilons),log10(maxError(:,3)),'-' )
title('Max norm Error')
xlabel('log10(epsilon)')
ylabel('log10(error)')
legend('GA','MQ','IQ')

% ==========================================================================
% Computation of Laplacian of RBF
function LPhi = Lfi(type,ep,r)

switch type
    case 'GA'
    % Gaussian RBF
    LPhi =  ( 4*ep^4*r.^2.*exp(-(ep*r).^2));

    case 'MQ'
    % MQ
    LPhi =  -(ep*r).^2.*(1 + (ep*r).^2).^(-3/2);

    case 'IQ'
    % IQ
    LPhi = (8*ep^4*r.^2.*(1 + (ep*r).^2).^(-3) );

```

```matlab
end
end

% ==========================================================================
% RBF Function computation
function Phi = fi(type,ep,r)
switch type
    case 'GA'
    % Gaussian RBF
    Phi = (exp(-(ep*r).^2));

    case 'MQ'
    % MQ
    Phi = (1 + (ep*r).^2).^(1/2);

    case 'IQ'
    % IQ
    Phi = (1 + (ep*r).^2).^(-1);
end
end
```

2 Time-dependent PDE: Global RBF

Solid body convection around a Unit Sphere

$$\frac{\partial h}{\partial t} = -\left(\frac{u}{a\cos\theta}\frac{\partial}{\partial\phi} + \frac{v}{a}\frac{\partial}{\partial\theta}\right)h \tag{9}$$

$$u = u_0(\cos\theta\cos\alpha - \sin\theta\sin\theta\sin\phi\sin\alpha) \tag{10}$$

$$v = u_0\cos\phi\sin\alpha \tag{11}$$

In the given exercise, we choose $u_0/a = 1 = 2\pi/T$, where T is the Time Period of revolution around the sphere. For the specific case of simulations related to the earth, we have a represents the radius of the earth, and $T = 12$ days.

2.1 Initial Conditions on Sphere

Cosine bell function

$$h(\theta,\phi) = \begin{cases} \frac{h_0}{2}\left(1 + cos(\frac{\pi r}{R})\right) & r < R \\ 0 & r \geq R \end{cases} \tag{12}$$

$$\begin{aligned} r &= a\cos^{-1}\left(\sin\theta_c\sin\theta + \cos\theta_c\cos\theta\cos(\phi - \phi_c)\right) \qquad (13)\\ R &= a/3 \\ h_0 &= 1 \\ (\theta_c,\phi_c) &= (0,0). \end{aligned}$$

2.2 Derivation of the method of applying RBF for solving the PDE

The system if solved as follows. We represent the time-varying PDE as follows.

$$
\begin{aligned}
\frac{dh}{dt} &= -\left(\frac{u}{\cos\theta}\frac{\partial}{\partial\phi} + v\frac{\partial}{\partial\phi}\right)h(\theta,\phi) \\
\frac{dh}{dt} &= -L(h)
\end{aligned}
$$

Further, we try to derive an RBF based spatial-stencil to approximate the Linear Operator L. This involves computing the weights(w) in the equation:

$$[A][w] = [f] = [L\Phi(||x - x_j||)|_{x=x_i}] \tag{14}$$

which f corresponds to the function we wish to approximate using RBFs. The above equation $[w]$ corresponds to the weights associated with the neighbourhood points, when the stencil is centered at point $x = x_i$.

Since, we have the data given in cartesian coordiantes (x, y, z) we first try to convert the Operator expression into cartesian coordinates, before working with RBFs. Since the Linear Operator,L is given by:

$$L = \frac{u}{\cos\theta}\frac{\partial}{\partial\phi} + v\frac{\partial}{\partial\phi} \tag{15}$$

we need to compute $\frac{\partial}{\partial}\Phi(||x - x_j||)$ and $\frac{\partial}{\partial}\Phi(||x - x_j||)$.

$$
\begin{aligned}
\phi &\equiv \phi(x, y, z) \\
\theta &\equiv \theta(x, y, z) \\
\frac{\partial}{\partial\phi} &= \frac{\partial}{\partial x}\frac{\partial x}{\partial\phi} + \frac{\partial}{\partial y}\frac{\partial y}{\partial\phi} + \frac{\partial}{\partial z}\frac{\partial z}{\partial\phi} \\
\frac{\partial}{\partial\theta} &= \frac{\partial}{\partial x}\frac{\partial x}{\partial\theta} + \frac{\partial}{\partial y}\frac{\partial y}{\partial\theta} + \frac{\partial}{\partial z}\frac{\partial z}{\partial\theta}
\end{aligned}
$$

Further, we have the conversion relations from spherical to cartesian coordinates for coordinates on a Unit Sphere

$$
\begin{aligned}
x &= \cos\phi\cos\theta \\
y &= \sin\phi\cos\theta \\
z &= \sin\theta
\end{aligned}
$$

This gives us the following relationships for the terms included in the above equation

$$
\begin{array}{cc}
\frac{\partial x}{\partial\phi} = -\sin\phi\cos\theta & \frac{\partial x}{\partial\theta} = -\cos\phi\sin\theta \\
\frac{\partial y}{\partial\phi} = \cos\theta\cos\phi & \frac{\partial y}{\partial\theta} = -\sin\phi\sin\theta \\
\frac{\partial z}{\partial\phi} = 0 & \frac{\partial z}{\partial\theta} = \cos\theta.
\end{array}
$$

Further, we also have the relation $r^2 = (x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2$, from which we derive

$$\frac{\partial r}{\partial x} = \frac{x - x_i}{r}$$
$$\frac{\partial r}{\partial y} = \frac{y - y_i}{r}$$
$$\frac{\partial r}{\partial z} = \frac{z - z_i}{r}.$$

Substituting, all the above expansion into the equation for Linear Operator L (15) obtain its representation in cartesian-coordinates. Further, we proceed to evaluate the cartesian-coordinate based respresentation of the Operator applied to the RBF-approximation of function (solution of PDE, $h(x, y, z, t)$) under study.

The weight$[w_i]$ vectors computed for each stencil centered at x_i form the rows of the Differentiation Matrix.

For solving the time-varying PDE, we use an RK4 based time-integrator to solve the following ODE

$$\frac{dh(x, y, z, t)}{dt} = -[DM]h(x, y, z, t) \tag{16}$$

where, $[DM]$ is the Differentiation Matrix computed using the code provided in the course-website.

For stability of the time-integration, we need to ensure that the eigenvalues of the Differentiation Matrix,$[DM]$, should be contained within the stability-domain of time-integrator for the given choice of time-step.

2.3 Matlab Code Implementation

2.3.1 Time-stepping using RK4

```

% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% PDE - TIME STEPPING
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------

% Convert from Cartesian Coordinates to Spherical Cordinates
ptsXYZ = xyz(:,1:3);
ptsX = xyz(:,1);
ptsY = xyz(:,2);
ptsZ = xyz(:,3);

Phi = xyz(:,4);
The = xyz(:,5);

% ------------------------------------------------------------------------
% Plot the Cosine Bell Function
m = 0:0.01:1;
R = 1/3;
```

```matlab
h0 = 1;
ratio = m/R;
indx = (ratio<1);
hfun  = h0/2*(1+cos(pi.*ratio)).*indx;
fig2 = figure(2)
plot(m,hfun)
xlabel('r')
ylabel('Cosine Bell: h(r)')

% Evaluate the Cosine Bell for Initial Condition
r = acos(cos(The).*cos(Phi));
ratio = r/R;
indx = (ratio<1);
initH  = h0/2*(1+cos(pi.*ratio)).*indx;          % Initialize Function

% ------------------------------------------------------------------------
% Plotting using RegularizeData3D (Matlab Central)
theta = -2:0.1:2;
phi = -3.5:0.1:3.5;
Smoothness = 0.00005;

fig4 = figure(4)
z0 = RegularizeData3D(The,Phi,initH, theta, phi, 'interp', 'bicubic', 'smoothness', Smoothness);
surf(theta, phi, z0, 'facealpha', 0.50);
hold on
scatter3(The,Phi,initH, 'fill');
hold off
xlabel('\theta')
ylabel('\phi')
zlabel('h(\theta,\phi,t = 0)')
title('Initialization of Cosine Bell Curve')
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% Time-Step using RK4
fractionOfRevolution = 1000;            % Ensure divisible by 4
T1rev = 2*pi;
t0 = 0;
tf = (10)*T1rev; % around 100 revolutions
dT = T1rev/fractionOfRevolution;
[t,H] = rk4_hw4(@fun_dudt_hw4, t0:dT:tf, initH, ptsXYZ);

% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% Plots of Time-Revolution: RegularizeData3D (Matlab Central FileID: #46223)

fig10 = figure(10)
subplot(2,2,1)
data = initH;
z0 = RegularizeData3D(The,Phi,data, theta, phi, 'interp', 'bicubic', 'smoothness', Smoothness);
surf(theta, phi, z0, 'facealpha', 0.50);
hold on
scatter3(The,Phi,data, 'fill');
hold off
```

```
xlabel('\theta')
ylabel('\phi')
zlabel('h(\theta,\phi,t)')
title('Initialization of Cosine Bell Curve')
% -------------------------------------
subplot(2,2,2)
data = H(:,fractionOfRevolution/4);
zHalfT = RegularizeData3D(The,Phi,data, theta, phi, 'interp', 'bicubic', 'smoothness', Smoothness);
surf(theta, phi, zHalfT, 'facealpha', 0.50);
hold on
scatter3(The,Phi,data,'fill')
hold off
xlabel('\theta')
ylabel('\phi')
zlabel('h(\theta,\phi,t)')
title('PDE Solution @t = T/4')
% -------------------------------------
subplot(2,2,3)
data = H(:,2*fractionOfRevolution/4);
zHalfT = RegularizeData3D(The,Phi,data, theta, phi, 'interp', 'bicubic', 'smoothness', Smoothness);
surf(theta, phi, zHalfT, 'facealpha', 0.50);
hold on
scatter3(The,Phi,data,'fill')
hold off
xlabel('\theta')
ylabel('\phi')
zlabel('h(\theta,\phi,t)')
title('PDE Solution @t = T/2')
% -------------------------------------
subplot(2,2,4)
data = H(:,3*fractionOfRevolution/4);
zHalfT = RegularizeData3D(The,Phi,data, theta, phi, 'interp', 'bicubic', 'smoothness', Smoothness);
surf(theta, phi, zHalfT, 'facealpha', 0.50);
hold on
scatter3(The,Phi,data,'fill')
hold off
xlabel('\theta')
ylabel('\phi')
zlabel('h(\theta,\phi,t)')
title('PDE Solution @t = 3*T/4')
% -------------------------------------
% Plot Error after full revolution
fig11 = figure(11)
error = abs(H(:,fractionOfRevolution) - initH);
data = error;
Smoothness = 0.00005;

zFullT = RegularizeData3D(The,Phi,data, theta, phi, 'interp', 'bicubic', 'smoothness', Smoothness);
surf(theta, phi, zFullT, 'facealpha', 0.75);
hold on
scatter3(The,Phi,data,'fill')
```

```
hold off
xlabel('\theta')
ylabel('\phi')
zlabel('error(\theta,\phi,t)')
title('Error after 1 revolutions')

% ----------------------------------------
% Plot Error after 10 full revolution
fig12 = figure(12)
error = abs(H(:,end) - initH);
data = error;
Smoothness = 0.00005;

zFullT = RegularizeData3D(The,Phi,data, theta, phi, 'interp', 'bicubic', '
    smoothness', Smoothness);
surf(theta, phi, zFullT, 'facealpha', 0.75);
hold on
scatter3(The,Phi,data,'fill')
hold off
xlabel('\theta')
ylabel('\phi')
zlabel('error(\theta,\phi,t)')
title('Error after 10 revolutions')

%==========================================================================
```

2.4 Simulations and results after 1/2, 1 and 10 revolutions around sphere

It is noted that after about 10 revolutions error of the order of 3 percent is observed. The graphs show the time-evolution of the cosine bell curve.

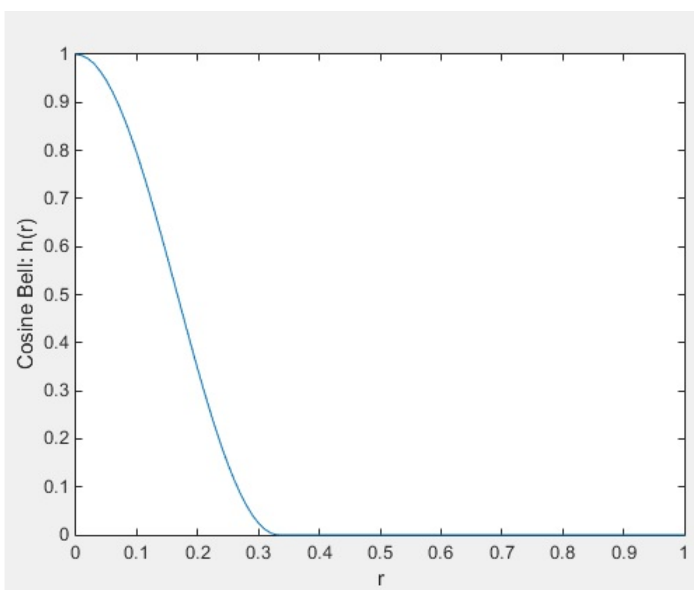

Figure 3: Cosine Bell Curve

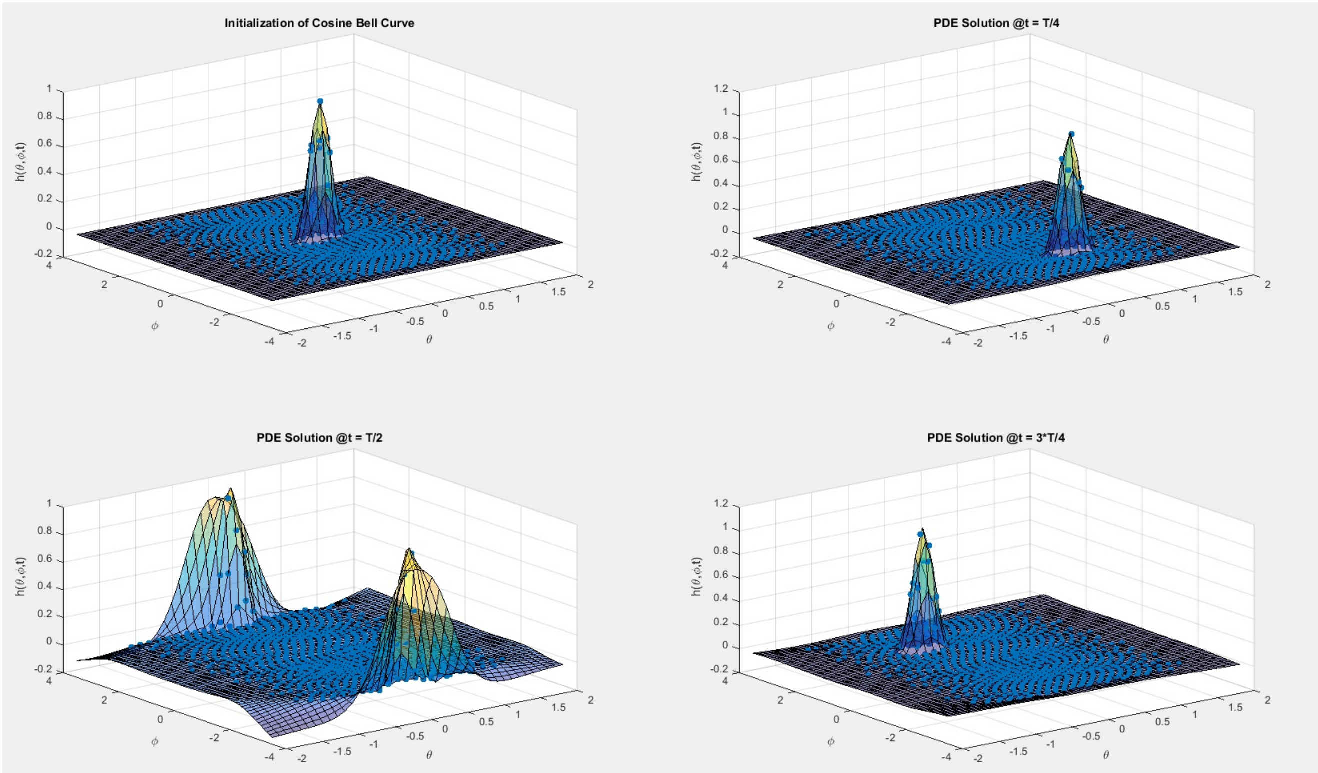

Figure 4: PDE Evolution over 1 revolution

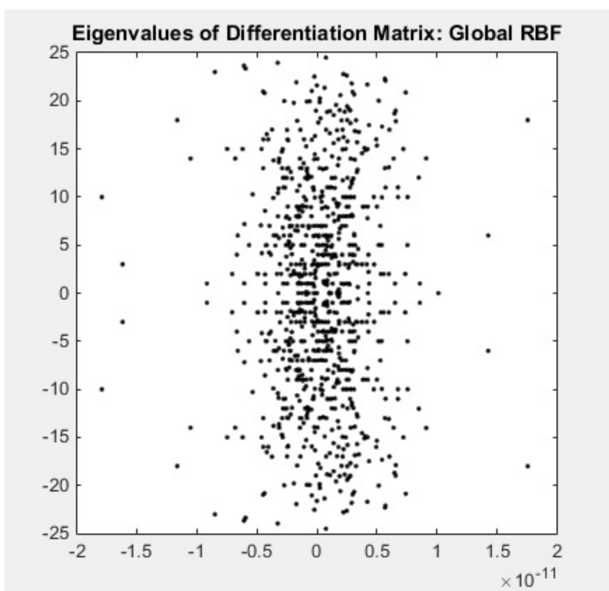

Figure 5: Eigenvalues of the Differentiation Matrix

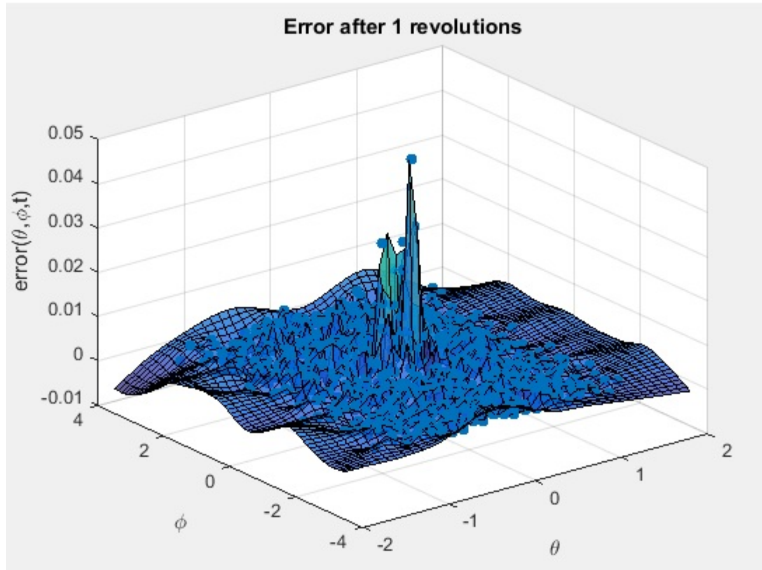

Figure 6: Errors after 1 rev

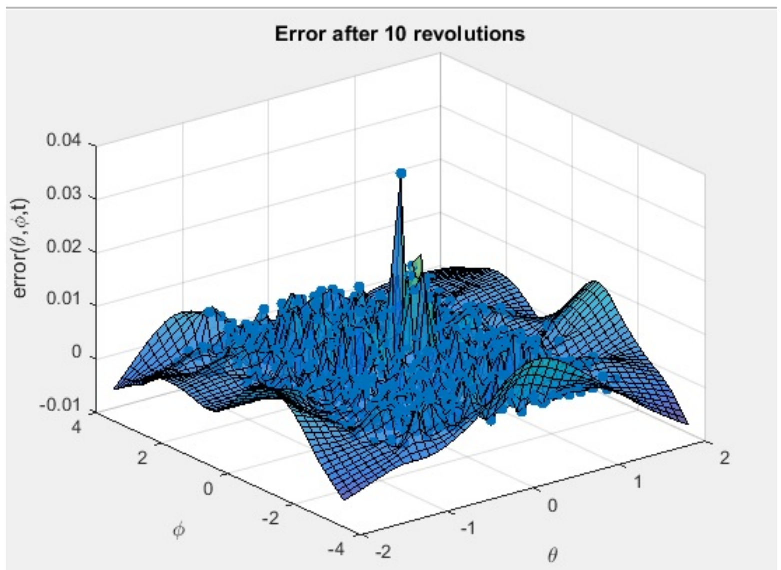

Figure 7: Errors after 10 revs

3 Time-dependent PDE: RBF- Finite Differences

For the RBF-FD based approach towards solving the PDE for convection on solid-sphere surface, the derivations are similar. The only major difference is in the choice of a local-stencil which comprises of $n = 20$ nearest neighbours of the center-point of the stencil.

The choice of local stencil leads to creation of a Sparse-Differentiation Matrix.

We observe that the sparse-DM created by RBF-FD has positive eigenvalues in the Right-Half Complex plane, which leads to instabilities in the time-stepping process.

3.1 Structure of RBF-FD Differentiation Matrix

3.2 Simulations and results after 1/2, 1 and 10 revolutions around sphere

It is observed that RBF-FD simulations become unstable even before completion of 1 revolution around the sphere. Further, it is to be noted that the present implementation does not consider combination of RBF and Polynomial approximation of the solution. This was because, for $\epsilon = 2$ we noted in class, that using adding polynomial terms does not add significantly to the accuracy.(Figure 5.7 of text-book.)

3.3 Matlab Code Implementation

3.3.1 Time-stepping using RK4

```matlab
% RBF-FD Differentiation Matrix
global D;
D = zeros(N,N);
D = sparse(D);

% Determine n-nearest neighbors
n = 20;
IDX = knnsearch(xyz(:,1:3),xyz(:,1:3),'K',n,'Distance','euclidean');

for i = 1:N % Evaluate L at xyz(i,:) when RBF centered at xyz(j,:), j=1:m
            % Note that we only need a single loop

    nbrs = IDX(i,:);

    % Compute values only to particular neighbors
    dx_dfi = -xyz(i,7)*xyz(i,8);        % To obtain the weights (which form
    dy_dfi =  xyz(i,6)*xyz(i,8);        % the rows of the DM) we need
    dz_dfi =  0;                        % for each RBF to calculate its
                                        % derivatives at node i with
    dx_dth = -xyz(i,6)*xyz(i,9);        % respect to fi and th.
    dy_dth = -xyz(i,7)*xyz(i,9);        % We obtain these via the chain
    dz_dth =  xyz(i,8);                 % rule after first calculating
                                        % derivatives of the mapping from


    % Analysis on only the neighbors : IS THIS NEEDED HERE or globally?
    XYZ = xyz(nbrs,1:3);    % Note: XYZ(1,:) = xyz(i,1:3)
    ONE = ones(length(nbrs),1);
    size(XYZ)
    size(ONE)
    % Compute the Distance Table for all pair-wise distances
    [x1,x2] = meshgrid(XYZ(:,1));           % Calculate a distance table for
    [y1,y2] = meshgrid(XYZ(:,2));           % all pairwise distances between
    [z1,z2] = meshgrid(XYZ(:,3));           % nodes (as measured straight
    d2 = (x1-x2).^2+(y1-y2).^2+(z1-z2).^2;  % through the sphere
    R = sqrt(d2);
    Ahat = fi(R);

    % Computing L_h: for the Local Stencil
    r = R(1,:)';
```

```
    size(r)

    dh_dr = dfi_dr(r);
    size(dh_dr)

    dh_dx = dh_dr.*(XYZ(:,1)-XYZ(1,1)*ONE)./r;
    dh_dy = dh_dr.*(XYZ(:,2)-XYZ(1,2)*ONE)./r;
    dh_dz = dh_dr.*(XYZ(:,3)-XYZ(1,3)*ONE)./r;
    dh_dx(1) = 0; dh_dy(1) = 0; dh_dz(1) = 0;   % Reset error from divByZero

    dh_dfi = dh_dx*dx_dfi + dh_dy*dy_dfi + dh_dz*dz_dfi;
    dh_dth = dh_dx*dx_dth + dh_dy*dy_dth + dh_dz*dz_dth;

    L_h = -(cos(al)-tan(xyz(i,5))*xyz(i,7)*sin(al))*dh_dfi + ...
            sin(al)*xyz(i,6)*dh_dth;    % L-operator at node i evaluated
                                        % for the different RBFs

    % Use the RBF-FD code here to determine the weights!
    size(Ahat)
    size(L_h)
    stencilWts = Ahat\L_h;

    % Insert values into Sparse Matrix: Is this correct?
    D(i,nbrs) = stencilWts';                      % Row i of the DM computed
end

fig1 = figure(1)
E = full(D);
subplot(2,1,1)
plot(eig(E),'k.'); axis square           % Plot the eigenvalues of the DM
title('Eigenvalues: RBF-FD')
subplot(2,1,2)
spy(D)
title('Sparsity Pattern: RBF-FD')
fprintf('Check D for symmetry\n')
issymmetric(D)

fig2 = figure(2)
rcm = symrcm(D);
Drcm = D(rcm,rcm);  % Sparse matrix
Ercm = full(Drcm);  % Full matrix
subplot(2,1,1)
plot(eig(Ercm),'k.'); axis square          % Plot the eigenvalues of the DM
title('Eigenvalues: RCMK ordering ')
subplot(2,1,2)
spy(Drcm)
title('Sparsity Pattern: Reverse Cuthill-McKee Ordering ')

fprintf('Check Drcm for symmetry\n')
issymmetric(Drcm)
% ---------------------------------------------------------------------------
% ---------------------------------------------------------------------------
% Plotting local stencil weights
close all;
selNodes = 4;
```

```
nodes = floor(rand(selNodes,1)*N);

nbrNodes = IDX(nodes,:);
minMarkerSize = 10;
maxMarkerSize = 80;

fig30 = figure(30)

for i=1:selNodes/2
    % ----------------------------------------------------
    subplot(selNodes/2,2,2*i-1)
    node = 2*i-1;
    stencilFocus = nodes(node);
    nbrs = nbrNodes(node,:);
    nbrsX = xyz(nbrs,1);
    nbrsY = xyz(nbrs,2);
    nbrsZ = xyz(nbrs,3);
    nbrWeights = D(stencilFocus,nbrs)';
    % scatter all points on sphere
    scatter(xyz(:,1),xyz(:,2),'g+')
    hold on
    % ------------------------
    pz = abs(nbrWeights);
    markerSizes = minMarkerSize + floor(pz/max(pz)*maxMarkerSize);
    red = (nbrWeights<0);
    green = zeros(size(nbrWeights));
    blue = (nbrWeights>=0);
    colorspec = [red green blue];
    % scatter Local Stencil points
    scatter(nbrsX,nbrsY,markerSizes,colorspec,'filled')
    hold off
    % ----------------------------------------------------
    subplot(selNodes/2,2,2*i)
    node = 2*i;
    stencilFocus = nodes(node);
    nbrs = nbrNodes(node,:);
    nbrsX = xyz(nbrs,1);
    nbrsY = xyz(nbrs,2);
    nbrsZ = xyz(nbrs,3);
    nbrWeights = D(stencilFocus,nbrs)';
    % scatter all points on sphere
    scatter(xyz(:,1),xyz(:,2),'g+')
    hold on
    % ------------------------
    pz = abs(nbrWeights);
    markerSizes = minMarkerSize + floor(pz/max(pz)*maxMarkerSize);
    red = (nbrWeights<0);
    green = zeros(size(nbrWeights));
    blue = (nbrWeights>=0);
    colorspec = [red green blue];
    % scatter Local Stencil points
    scatter(nbrsX,nbrsY,markerSizes,colorspec,'filled')
    hold off
end
```

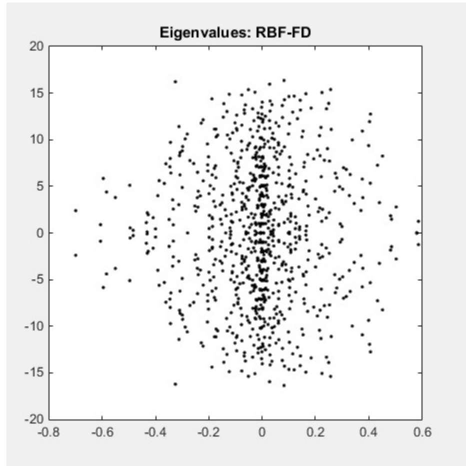

Figure 8: Eigenvalues of RBF-FD DM

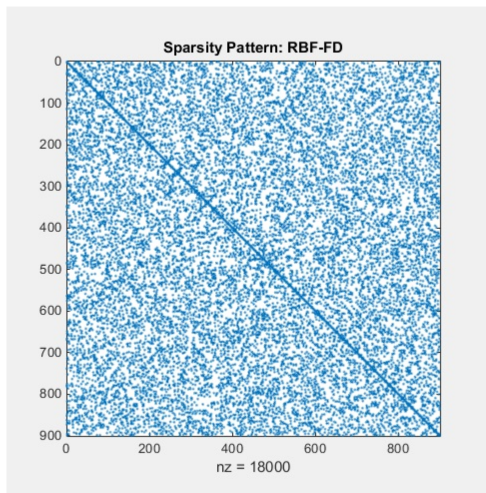

Figure 9: Sparsity Pattern of RBF-FD Differentiation Matrix

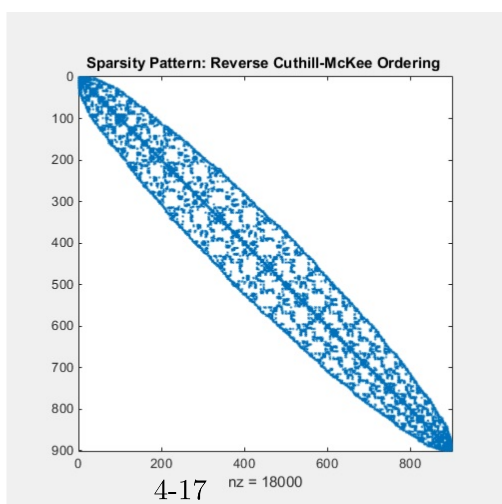

Figure 10: Sparsity Pattern of RBF-FD DM: Reverse Cuthill-McKee Ordering

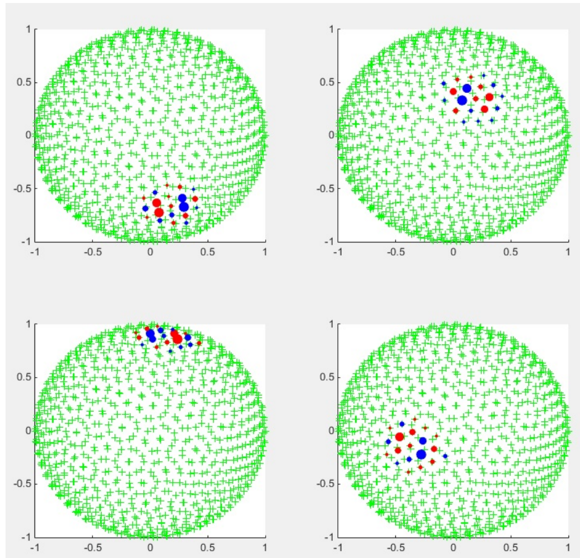

Figure 11: Plot of Local Stencils

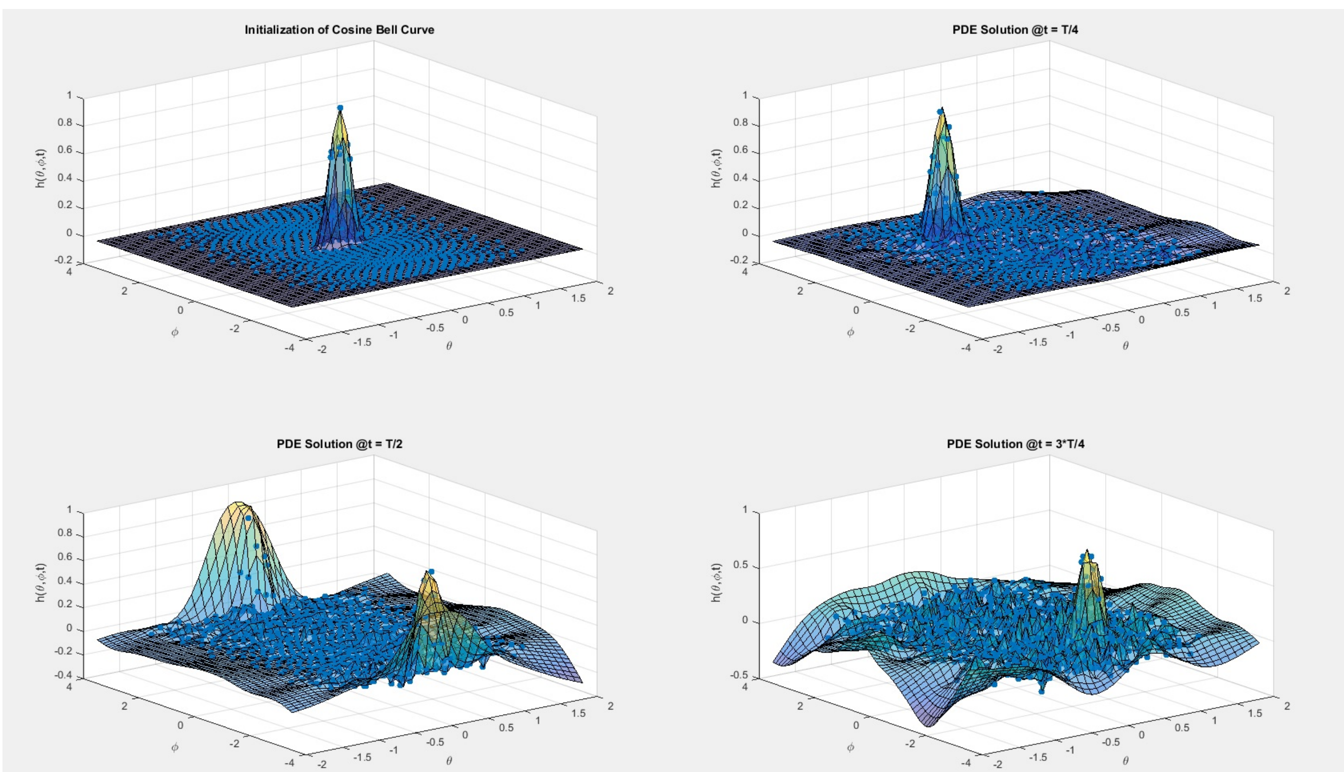

Figure 12: PDE Evolution over 1 revolution

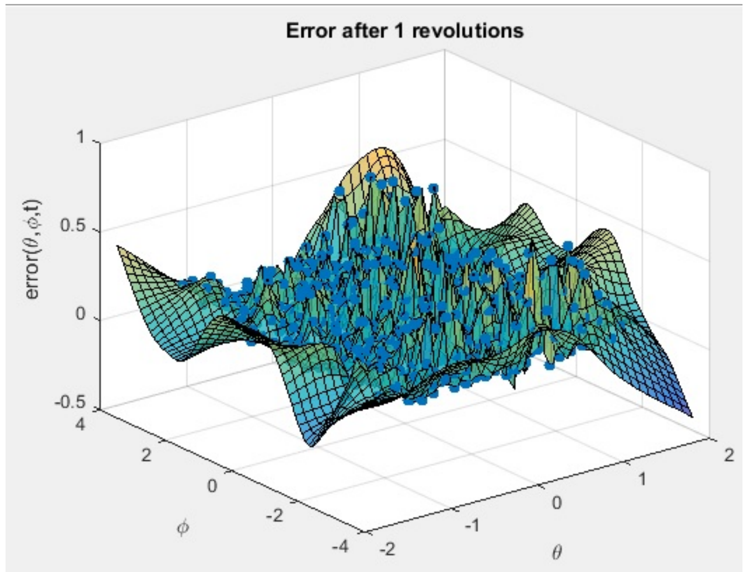

Figure 13: Errors after 1 rev

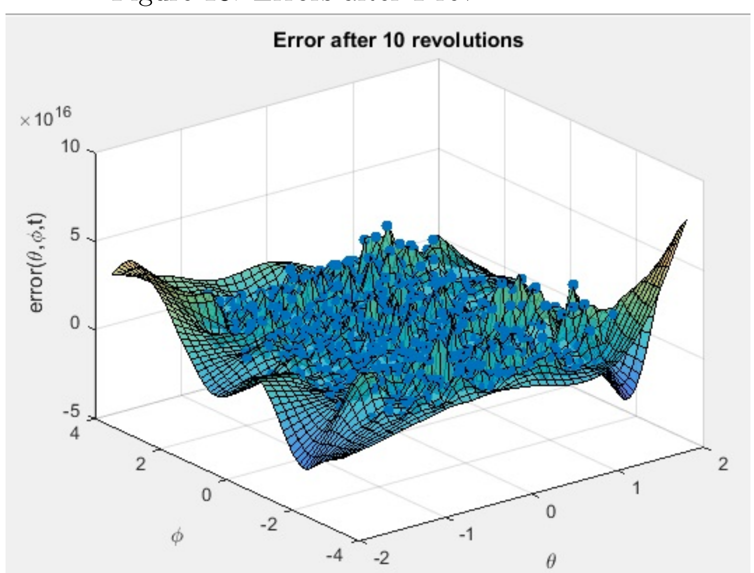

Figure 14: Errors after 10 revs