**Chapter II**

**1. Gradient Echo (GRE) Simulation:**

**Main function for Gradient echo simulation - mrirecon\_gradechoV2\_dMBydT.m**

function[S,Ke,t]=mrirecon\_gradechoV2\_dMBydT(alpha,dt,rhomat,T1mat,T2mat,flip,TE,TRep, Fov)

%Output Variables

%t=time stamps of echo

%tau = lam x dt Duration of Phase Encoding pulse

%If lam>1, the resolution of the reconstructed image reduces since

%the effective echo time is reduced. This is because tau=lam x dt is taken

%off from TE/2, prior to refocusing.

%--------------------------------------------------------------------------------------------------------------

Gama=47.6e6;

[Ke,t]=Kspace\_gradechoV2\_dMBydT(alpha,Gama,dt,rhomat,T1mat,T2mat,flip,TE,TRep,Fov);

S=fftshift( ifft2( ifftshift( Ke ) ) );

S=real(S);

S=fliplr(S);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**2. Calling functions in GRE simulation**

**Kspace\_gradechoV2\_dMBydT.m**

function[K,t]=Kspace\_gradechoV2\_dMBydT(alpha,Gama,dt,rhomat,T1mat,T2mat,flip,TE,

TRep, Fov)

%If Fov\_y=Fov\_x, delgy= Gx/lam (The phase encoding gradient step-size)

%Fov is actually Fov\_x =[ -x0 x0 ] and Fov\_y is taken as [ y0 -y0]

%----------------------------------------------------------------------------------------------------------------

[M,N] = size(T1mat); %M: No. of read out samples

%N: No. of Phase encode steps

Gx=1e3/Gama/dt/( Fov(2)-Fov(1) );

MX=zeros(size(rhomat));

MY=MX;

MZ=rhomat;

[K0,t]=sliceout\_gradechoV2\_dMBydT(alpha,Gama,dt,MX,MY,MZ,T1mat,T2mat,flip,TE,

TRep,Fov,Gx,0);

lam=1;

Nx=find( abs( K0 )== max( abs( K0 ) ) );

delgy= Gx /lam; %Assuming Fov\_x = Fov\_y

%delgy represents the phase-encoding gradient step

Ny=N+1/2;

Gy=Ny\*delgy;

gy=linspace( -Gy,Gy,255 );

K=zeros( N,N );

for j=1:N,

[k,t,MXo,MYo,MZo]=sliceout\_gradechoV2\_dMBydT(alpha,Gama,dt,MX,MY,MZ,T1mat,

T2mat,flip,TE,TRep,Fov,Gx,gy(j));

K(j,:)=k(1: N);

MX=MXo;

MY=MYo;

MZ=MZo;

end

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**sliceout\_gradechoV2\_dMBydT.m**

function[S,t,MXo,MYo,MZo]=sliceout\_gradechoV2\_dMBydT(alpha,Gama,dt,MX,MY,MZ,

T1mat,T2mat,flip,TE,TRep,Fov,Gx,Gy)

%This program computes the summated saturation recovery signal from the

%slice in one excitation period (For a given Gy Value)

%S: Echo signal

%t: Echo time stamps

%-------------------------------------------------------------------------------------------------------------

Res=size(MX,1); %Phantom is of size Res x Res

y0=Fov(2); % It is assumed that Fov\_x = Fov\_y

xx=linspace( Fov(1),Fov(2), Res );yy=linspace(y0,-y0,Res);

[X,Y]=ndgrid(xx,yy);

X=X';Y=Y';

S=0;

MXo=[];

MYo=[];

MZo=[];

for i=1:Res,

for j=1:Res,

x= double( X(i,j) ); y= double( Y(i,j) );

mxorg=MX(i,j);

myorg=MY(i,j);

mzorg=MZ(i,j);

Morg=[mxorg;myorg;mzorg];

T1=double( T1mat(i,j) );

T2=double( T2mat(i,j) );

[M,Me,t,Mout]=gradecho\_wgradV2\_dMBydT(alpha,Gama,dt,Morg,flip,TE,TRep,T1,T2,x,y,Gx, Gy);

MXo(i,j)=Mout(1);

MYo(i,j)=Mout(2);

MZo(i,j)=Mout(3);

S=S+Me;

end

end

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**gradecho\_wgradV2\_dMBydT.m**

function[S,Se,t,Mout]=gradecho\_wgradV2\_dMBydT(alpha,Gama,dt,Morg,flip,TE,TRep,T1,T2, x,y,Gx,Gy)

%Each excitation is such that the magnetization vector is rotated about y-axis

%TE : Echo Time

%TRep= Sequence repetition time ( TRe >= TE + TE(alpha+eta) )

%Gama=47.6e6 Hz/T;

%------------------------------------------------------------------------------------------------------------------

t=0:dt:TRep;

Ne= length(0:dt:TE); % No. of samples during readout

Nrep=length(0:dt:(alpha+2)\*TE+1); %No. of samples during one RF repetition.

Rflipy=yrot(flip); %Rotation about y-axis

beta=1/2/alpha;

M=zeros(3,Nrep);

[A0,B0]=freeprecess( dt,T1,T2,0 ); %Free precess in the absence of gradient

[Ax,Bx]=freeprecess( dt,T1,T2,-Gama\*x\*beta\*Gx); % Negative frequency traversal

[Ay,By]=freeprecess( dt,T1,T2,Gama\*y\*Gy); % Phase

[Ar,Br]=freeprecess( dt,T1,T2,Gama\*x\*Gx); % Readout frequency traversal

[Af,Bf]=freeprecess( TRep-Nrep\*dt,T1,T2,0 );

M(:,1)=Rflipy\*Morg;

Nxy=length(0:dt:alpha\*TE); % Number of samples during negative frequency traversal.

for j=2:Nxy+1,

M(:,j)= Ax\*M(:,j-1)+Bx ; %Pre echo Frequency during negative frequency traversal.

end

M(:,Nxy+2)= Ay\*M(:,Nxy+1)+By ;% During phase Encoding

for j=Nxy+3 : Nxy+3+Ne-1,

M(:,j)=Ar\*M(:,j-1)+Br; % Readout

end

M0= Af\*M(:,Nxy+3+Ne-1)+Bf; %Free Precession

S0=M0;

Su=M;

Mout=S0;

if(T2>0),

S=(-1/T2)\*Su(1,:)+1i\*(-1/T2)\*Su(2,:);

else

S=zeros(1,size(Su,2));

end

Se=S( Nxy+3 : Nxy+3+Ne-1 );

t=t( Nxy+3 : Nxy+3+Ne-1);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**3. Spin echo simulation**

**Main function for Spin echo simulation - mrirecon\_spinechoV2\_dMBydT.m**

function[S,Ke,L,t]=mrirecon\_spinechoV2\_dMBydT(Nf,F,lam,dt,rhomat,T1mat,T2mat,flip,TE,

TRep,Fov)

%Output Variables

%t=time stamps of echo

%tau = lam x dt Duration of Phase Encoding pulse

%If lam>1, the resolution of the reconstructed image reduces since the

%effective echo time is reduced. This is because tau=lam x dt is taken

%off from TE/2, prior to refocusing.

%--------------------------------------------------------------------------------------------------------

Gama=47e6;

[Ke,L,t]=Kspace\_spinechoV2\_dMBydT(Nf,F,Gama,lam,dt,rhomat,T1mat,T2mat,flip,TE,TRep,

Fov);

S=fftshift( ifft2( ifftshift( Ke ) ) );

S=real(S);

S=fliplr(S);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**4. Calling functions in spin echo simulation**

**Kspace\_spinechoV2\_dMBydT.m**

function[K,L,t]=Kspace\_spinechoV2\_dMBydT(Nf,F,Gama,lam,dt,rhomat,T1mat,T2mat,flip,TE, TRep, Fov)

% r: Duration of Phase Encoding gradient tau= r\*TE/2

%lam: is an intermediate variable tau=lam\*dt (lam=1 here)

%If Fov\_y=Fov\_x, delgy= Gx/lam (The phase encoding gradient step)

%Fov is actually Fov\_x =[ -x0 x0 ] and Fov\_y is taken as [ y0 -y0]

%----------------------------------------------------------------------------------------------------------------

Gx=2\*pi\*1e3/Gama/dt/( Fov(2)-Fov(1) );

MX=zeros(size(rhomat));

MY=MX;

MZ=rhomat;

[K0,t]=sliceout\_spinechoV2\_dMBydT(Nf,F,Gama,lam,dt,MX,MY,MZ,T1mat,T2mat,flip,TE,

TRep,Fov,Gx,0);

lam=1;

Nx=find( abs(K0)== max( abs(K0) ) );

N\_ky=2\*Nx-1 % No. of y-gradient levels

delgy= Gx/lam ; %Assuming Fov\_x = Fov\_y

%delgy represents the phase-encoding gradient step

Gy=(Nx-1)\*delgy;

gy=linspace(Gy,-Gy,2\*Nx-1);

K=zeros( 2\*Nx-1, 2\*Nx-1 );

L=zeros(1,N\_ky);

for j=1:N\_ky,

j,

[k,t,MXo,MYo,MZo]=sliceout\_spinechoV2\_dMBydT(Nf,F,Gama,lam,dt,MX,MY,MZ,T1mat,

T2mat,flip,TE,TRep,Fov,Gx,gy(j) );

L=length(abs(k));

if L<=N\_ky

k=[k,zeros(1,N\_ky-L)];

end

K(j,:)=k(1: N\_ky);

MX=MXo;

MY=MYo;

MZ=MZo;

end

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**sliceout\_spinechoV2\_dMBydT.m**

function[S,t,MXo,MYo,MZo]=sliceout\_spinechoV2\_dMBydT(Nf,F,Gama,lam,dt,MX,MY,MZ, T1mat,T2mat,flip,TE,TRep,Fov,Gx,Gy)

%This program computes the summated saturation recovery signal from the

%slice in one excitation period (For a given Gy Value)

%Mxorg, Myorg, Mzorg are functions of spatial coordinates (i,j)

%tau= lam x dt: Duration of Phase Encoding Gradient, applied just prior to

%refocusing

%S: Echo signal

%t: Echo time stamps

%-----------------------------------------------------------------------------------------------------------------

Res=size(MX,1); %Phantom is of size Res x Res

y0=Fov(2); % It is assumed that Fov\_x = Fov\_y

xx=linspace( Fov(1),Fov(2), Res );yy=linspace(y0,-y0,Res);

[X,Y]=ndgrid(xx,yy);

X=X';Y=Y';

S=0;

MXo=[];

MYo=[];

MZo=[];

for i=1:Res,

for j=1:Res,

x= double( X(i,j) ); y= double( Y(i,j) );

mxorg=MX(i,j);

myorg=MY(i,j);

mzorg=MZ(i,j);

Morg=[mxorg;myorg;mzorg];

T1=double( T1mat(i,j) );

T2=double( T2mat(i,j) );

[M,Me,t,Mout]=spinecho\_wgradV2\_dMBydT(Nf,F,Gama,lam,dt,Morg,flip,TE,TRep,T1,T2,x,y, Gx,Gy);

MXo(i,j)=Mout(1);

MYo(i,j)=Mout(2);

MZo(i,j)=Mout(3);

S=S+Me;

end

end

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**spinecho\_wgradV2\_dMBydT.m**

function[S,Se,t,Mout]=spinecho\_wgradV2\_dMBydT(Nf,F,Gama,lam,dt,Morg,flip,TE,TRep,T1, T2,x,y,Gx,Gy)

%Each excitation is such that the magnetization vector is rotated about y-axis

%TE : Echo Time

%TR : readout Time ( TR <= TE )

%TRep= Sequence repetition time ( TRe >= TE + TE/2 )

%Nf: No. of spins

%F: Range of Off-Resonance frequencies

%Gama=47.6e6;

%r : Phase Encoding gradient duration ratio ( T=r x (TE/2) ) ,

%In this Version, r is not used as the Y-gradient is applied just before

%refocus with a duration tau=dt

%lam: is an intermediate variable tau=lam\*dt (lam=1 here)

%------------------------------------------------------------------------------------------------------------------

t=0:dt:TRep;

df=F\*( rand(1,Nf) - 0.5 ); %Set of Off-resonance Frequencies

Ne= length(0:dt:TE/2); % No. of samples till refocusing

% There will be 2Ne samples during frequency

% encoding after refocusing

Nrep=length(0:dt:TRep); %No. of samples during readout

Rflipy=yrot(flip); %Rotation about y-axis

Rflipx=xrot(pi);

M=zeros(3,Nrep);

Su=0;

S0=0;

for nf=1:Nf,

[A0,B0]=freeprecess( dt,T1,T2,df(nf) ); %Free precess in the absence of gradient

[Ax,Bx]=freeprecess( dt,T1,T2,df(nf)+Gama\*x\*Gx/2/pi );

[Ay,By]=freeprecess( dt,T1,T2,df(nf) +Gama\*y\*Gy/2/pi );

[Af,Bf]=freeprecess( TRep-(3\*Ne+1)\*dt,T1,T2,df(nf) );

M(:,1)=Rflipy\*Morg;

%---------------------------------------------Prior to Refocusing----------------------------

for j=2:Ne-(lam+1),

M(:,j)= Ax\*M(:,j-1)+Bx ; %Pre echo Frequency Encoding in the absence of phase encoding

end

for j=Ne-lam : Ne-1,

M(:,j)=Ay\*M(:,j-1)+By; %Phase Encoding Gradient

end

%----------------------------------------------Refocusing------------------------------------

M(:,Ne)=Rflipx\*M(:,Ne-1); %Refocusing applied

%---------------------------------------------------------------------------------------------------

for j= Ne+1 : 3\*Ne,

M(:,j)=Ax\*M(:,j-1)+Bx ; %Readout applied (N samples available during readout)

end

M0= Af\*M(:,3\*Ne)+Bf; %Free Precession

S0=S0+(1/Nf)\*M0;

Su=Su+(1/Nf)\*M;

end

Mout=S0;

if(T2>0),

S=(-1/T2)\*Su(1,:)+1i\*(-1/T2)\*Su(2,:);

else

S=zeros(1,size(Su,2));

end

Se=S( Ne+3 : 2+3\*Ne );

t=t( Ne+3 : 2+3\*Ne );

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**5. Simulation of SWI with air gap - simulateswiwithairgap.m**

function[psi]=simulateswiwithairgap

siz=[256,256,256];

sizm=[128,128,100];

cc=[0,0;-32,0;32,0;0,32;0,-32;-32,32;32,32;-32,-32;32,-32;-50,50;50,50;-50,-50;50,-50];

a=3;

psit=9\*10^-6;

psiv=9.45\*10^-6;

cencc=[0,0];

b=15;

[phantom,phi]=simulatephwithairgvol(siz,sizm,cc,cencc,a,b,psit,psiv);

psi=10\*real(phi(:,:,100));

%---------------------------------------CALLING FUNCTION---------------------------------

function[phantom,phi]=simulatephwithairgvol(siz,sizm,cc,cencc,a,b,psit,psiv)

%siz should be even num (N x M x O)

%sizm is the size of inner cube (n x m x o)

% cc :- coordinates of the circle

%a is the radius

N=siz(1);

M=siz(2);

O=siz(3);

phantom=zeros(N,M,O);

x=linspace(-M/2,M/2-1,M);

y=linspace(N/2-1,-N/2,N);

z=linspace(O/2-1,-O/2,O);

[kx,ky,kz]=meshgrid(x,y,z);

I=xcrphantom(sizm,cc,cencc,a,b,psit,psiv);

n=sizm(1);

m=sizm(2);

o=sizm(3);

InM=repmat(I,[1,1,sizm(3)]);

phantom(N/2+1-n/2:N/2+n/2,M/2+1-m/2:M/2+m/2,O/2+1-o/2:O/2+o/2)=InM;

gam = 47.6e6;

B0=1.5;

TE=7.55\*10^-3;

H = greensfn( siz,0.5,0.5,0.5,'' );

Kpsi=volumeKrecon(phantom);

FKpsi=Kpsi.\*H;

phi=-gam\*B0\*TE\*volumerecon(FKpsi,0);

%-----------------------------------------------------------------------------------------------------------

function[I]=xcrphantom(sizm,cc,cencc,a,b,psit,psiv)

% Program to create a phantom with cylindrical vessels

% cc :- coordinates of the circle within the field of view

N = sizm(1);

M =sizm(2);

c=size(cc,1);

x=linspace(-M/2,M/2-1,M);

y=linspace(N/2-1,-N/2,N);

[X Y] = meshgrid( x,y );

I=zeros(N,M);

for i=1:c,

R=sqrt( ( X-cc(i,1) ).^2 + ( Y-cc(i,2) ).^2 );

ind=find( R<=a );

I(ind)=1;

end

indz=find( I==0 );

indo=find( I==1 );

I(indz)=psit;

I(indo)=psiv;

R=sqrt( ( X-cencc(1) ).^2 + ( Y-cencc(2) ).^2 );

ind=find( R<=b );

I(ind)=0;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**6. Simulation of SWI without air gap – simulateswiwithoutag.m**

function[psi]=simulateswiwithoutag

siz=[256,256,256];

sizm=[128,128,100];

cc=[0,0;-32,0;32,0;0,32;0,-32;-32,32;32,32;-32,-32;32,-32;-50,50;50,50;-50,-50;50,-50];

a=3;

psit=9\*10^-6;

psiv=9.45\*10^-6;

[phantom,phi]=simulatephvol(siz,sizm,cc,a,psit,psiv);

psi=10\*real(phi(:,:,100));

%-----------------------------------------CALLING FUNCTION------------------------------

function[phantom,phi]=simulatephvol(siz,sizm,cc,a,psit,psiv)

%siz should be even num (N x M x O)

%sizm is the size of inner cube (n x m x o)

% cc :- coordinates of the circle

%a is the radius

N=siz(1);

M=siz(2);

O=siz(3);

phantom=zeros(N,M,O);

x=linspace(-M/2,M/2-1,M);

y=linspace(N/2-1,-N/2,N);

z=linspace(O/2-1,-O/2,O);

[kx,ky,kz]=meshgrid(x,y,z);

I=xcrphantom(sizm,cc,a,psit,psiv);

n=sizm(1);

m=sizm(2);

o=sizm(3);

InM=repmat(I,[1,1,sizm(3)]);

phantom(N/2+1-n/2:N/2+n/2,M/2+1-m/2:M/2+m/2,O/2+1-o/2:O/2+o/2)=InM;

gam = 47.6e6;

B0=1.5;

TE=7.55\*10^-3;

A=kz.^2;

B=(kx.^2+ky.^2+kz.^2);

H=1/3-(A./B);

ind=find(B==0);

[ H ] = greensfn( siz,0.5,0.5,0.5,'' );

Kpsi=volumeKrecon(phantom);

FKpsi=Kpsi.\*H;

phi=-gam\*B0\*TE\*volumerecon(FKpsi,0);

function[I]=xcrphantom(sizm,cc,a,psit,psiv)

% Program to create a phantom with cylindrical vessels

% cc :- coordinates of the circle within the field of view

N = sizm(1);

M =sizm(2);

c=size(cc,1);

x=linspace(-M/2,M/2-1,M);

y=linspace(N/2-1,-N/2,N);

[X Y] = meshgrid( x,y );

I=zeros(N,M);

for i=1:c,

R=sqrt( ( X-cc(i,1) ).^2 + ( Y-cc(i,2) ).^2 );

ind=find( R<=a );

I(ind)=1;

end

indz=find( I==0 );

indo=find( I==1 );

I(indz)=psit;

I(indo)=psiv;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Chapter III**

**1. Complex Difference Speed Image**

function [CDSpd] = CDspeed( )

%CDspeed function to generate speed image using Complex Difference method.

%CDSpd provides the speed image of PC-MRA data given.

% The PC-MRA data consist of 6 channels and each channel consist of 4 partitions.

%Channel wise computation is performed and final step is summation of speed images

%of all the 6 channels.

% MHA3D11 to MHA3D46 :Represents partitioned data sets of all 6 channels.

% mhom : Homodyne filtered data (Available from mat files provided)

% CD1x, CD1y, CD1z : variables storing product of magnitude and complex

% difference computed using balanced four point method.

% CDspeed1 : variable storing the root sum of squares of CD1x, CD1y & CD1z.

disp('channel1');

%channel 1

load MHA3D11

S11=mhom;

load MHA3D21

S12=mhom;

load MHA3D31

S13=mhom;

load MHA3D41

S14=mhom;

CD1x=0.5\*((S12+S13)-(S11+S14));

CD1y=0.5\*((S12+S14)-(S11+S13));

CD1z=0.5\*((S13+S14)-(S11+S12));

CDSpeed1=sqrt(CD1x.^2+CD1y.^2+CD1z.^2);

clear mhom S11 S12 S13 S14

disp('channel2');

%channel 2

load MHA3D12

S21=mhom;

load MHA3D22

S22=mhom;

load MHA3D32

S23=mhom;

load MHA3D42

S24=mhom;

S2x=0.5\*((S22+S23)-(S21+S24));

S2y=0.5\*((S22+S24)-(S21+S23));

S2z=0.5\*((S23+S24)-(S21+S22));

CDSpeed2=sqrt(S2x.^2+S2y.^2+S2z.^2);

clear mhom S21 S22 S23 S24

disp('channel3');

%channel 3

load MHA3D13

S31=mhom;

load MHA3D23

S32=mhom;

load MHA3D33

S33=mhom;

load MHA3D43

S34=mhom;

S3x=0.5\*((S32+S33)-(S31+S34));

S3y=0.5\*((S32+S34)-(S31+S33));

S3z=0.5\*((S33+S34)-(S31+S32));

CDSpeed3=sqrt(S3x.^2+S3y.^2+S3z.^2);

clear mhom S31 S32 S33 S34

disp('channel4');

%channel 4

load MHA3D14

S41=mhom;

load MHA3D24

S42=mhom;

load MHA3D34

S43=mhom;

load MHA3D44

S44=mhom;

S4x=0.5\*((S42+S43)-(S41+S44));

S4y=0.5\*((S42+S44)-(S41+S43));

S4z=0.5\*((S43+S44)-(S41+S42));

CDSpeed4=sqrt(S4x.^2+S4y.^2+S4z.^2);

clear mhom S41 S42 S43 S44

disp('channel5');

%channel 5

load MHA3D15

S51=mhom;

load MHA3D25

S52=mhom;

load MHA3D35

S53=mhom;

load MHA3D45

S54=mhom;

S5x=0.5\*((S52+S53)-(S51+S54));

S5y=0.5\*((S52+S54)-(S51+S53));

S5z=0.5\*((S53+S54)-(S51+S52));

CDSpeed5=sqrt(S5x.^2+S5y.^2+S5z.^2);

clear mhom S51 S52 S53 S54

disp('channel6');

%channel 6

load MHA3D16

S61=mhom;

load MHA3D26

S62=mhom;

load MHA3D36

S63=mhom;

load MHA3D46

S64=mhom;

S6x=0.5\*((S62+S63)-(S61+S64));

S6y=0.5\*((S62+S64)-(S61+S63));

S6z=0.5\*((S63+S64)-(S61+S62));

CDSpeed6=sqrt(S6x.^2+S6y.^2+S6z.^2);

clear mhom S61 S62 S63 S64

CDSpd=CDSpeed1+CDSpeed2+CDSpeed3+CDSpeed4+CDSpeed5+CDSpeed6;

end

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**2. Phase Difference speed image**

function [Phiorg] = PDspeed()

%PDspeed function to generate speed image using Phase Difference method.

%Phiorg provides the speed image of PC-MRA data given.

% The PC-MRA data consist of 6 channels and each channel consist of 4

% partitions. Channel wise computation is performed and final step is

% summation of speed images of all the 6 channels.

% MHA3D11 to MHA3D46 :Represents partitioned data sets of all 6 channels.

% mhom : Homodyne filtered data (Available from mat files provided)

% Phi11, Phi21, Phi31, Phi41 : variables to store phase.

% M11, M21, M31, M41 : variables to store Magnitude.

% Phio1x, Phio1y, Phio1z : variables storing product of magnitude and

% Phase difference computed using balanced four point method.

% Phio1xyz : variable storing the root sum of squares of Phio1x, Phio1y, and Phio1z.

% M1 is net magnitude of all partitions of channel 1.

% Phi1o stores the PDspeed image.

% Same proceess repeated in all channels.

% Phiorg: Final speed image of all channels.

%channel 1

load MHA3D11

Phi11=angle(mhom);

M11=abs(mhom);

clear mhom

load MHA3D21

Phi21=angle(mhom);

M21=abs(mhom);

clear mhom

load MHA3D31

Phi31=angle(mhom);

M31=abs(mhom);

clear mhom

load MHA3D41

Phi41=angle(mhom);

M41=abs(mhom);

clear mhom

Phio1x=Phi21+Phi31-(Phi11+Phi41);

Phio1y=Phi21+Phi41-(Phi11+Phi31);

Phio1z=Phi31+Phi41-(Phi11+Phi21);

clear Phi11 Phi21 Phi31 Phi41

Phio1xyz=sqrt(Phio1x.^2+Phio1y.^2+Phio1z.^2);

clear Phio1x Phio1y Phio1z

M1=0.25\*(M11+M21+M31+M41);

clear M11 M21 M31 M41

Phi1o=Phio1xyz.\*M1;

clear Phio1xyz M1

%channel 2

load MHA3D12

Phi12=angle(mhom);

M12=abs(mhom);

clear mhom

load MHA3D22

Phi22=angle(mhom);

M22=abs(mhom);

clear mhom

load MHA3D32

Phi32=angle(mhom);

M32=abs(mhom);

clear mhom

load MHA3D42

Phi42=angle(mhom);

M42=abs(mhom);

clear mhom

Phio2x=Phi22+Phi32-(Phi12+Phi42);

Phio2y=Phi22+Phi42-(Phi12+Phi32);

Phio2z=Phi32+Phi42-(Phi12+Phi22);

clear Phi12 Phi22 Phi32 Phi42

Phio2xyz=sqrt(Phio2x.^2+Phio2y.^2+Phio2z.^2);

clear Phio2x Phio2y Phio2z

M2=0.25\*(M12+M22+M32+M42);

clear M12 M22 M32 M42

Phi2o=Phio2xyz.\*M2;

clear Phio2xyz M2

%channel 3

load MHA3D13

Phi13=angle(mhom);

M13=abs(mhom);

clear mhom

load MHA3D23

Phi23=angle(mhom);

M23=abs(mhom);

clear mhom

load MHA3D33

Phi33=angle(mhom);

M33=abs(mhom);

clear mhom

load MHA3D43

Phi43=angle(mhom);

M43=abs(mhom);

clear mhom

Phio3x=Phi23+Phi33-(Phi13+Phi43);

Phio3y=Phi23+Phi43-(Phi13+Phi33);

Phio3z=Phi33+Phi43-(Phi13+Phi23);

clear Phi13 Phi23 Phi33 Phi43

Phio3xyz=sqrt(Phio3x.^2+Phio3y.^2+Phio3z.^2);

clear Phio3x Phio3y Phio3z

M3=0.25\*(M13+M23+M33+M43);

clear M13 M23 M33 M43

Phi3o=Phio3xyz.\*M3;

clear Phio3xyz M3

%channel 4

load MHA3D14

Phi14=angle(mhom);

M14=abs(mhom);

clear mhom

load MHA3D24

Phi24=angle(mhom);

M24=abs(mhom);

clear mhom

load MHA3D34

Phi34=angle(mhom);

M34=abs(mhom);

clear mhom

load MHA3D44

Phi44=angle(mhom);

M44=abs(mhom);

clear mhom

Phio4x=Phi24+Phi34-(Phi14+Phi44);

Phio4y=Phi24+Phi44-(Phi14+Phi34);

Phio4z=Phi34+Phi44-(Phi14+Phi24);

clear Phi14 Phi24 Phi34 Phi44

Phio4xyz=sqrt(Phio4x.^2+Phio4y.^2+Phio4z.^2);

clear Phio4x Phio4y Phio4z

M4=0.25\*(M14+M24+M34+M44);

clear M14 M24 M34 M44

Phi4o=Phio4xyz.\*M4;

clear Phio4xyz M4

%channel 5

load MHA3D15

Phi15=angle(mhom);

M15=abs(mhom);

clear mhom

load MHA3D25

Phi25=angle(mhom);

M25=abs(mhom);

clear mhom

load MHA3D35

Phi35=angle(mhom);

M35=abs(mhom);

clear mhom

load MHA3D45

Phi45=angle(mhom);

M45=abs(mhom);

clear mhom

Phio5x=Phi25+Phi35-(Phi15+Phi45);

Phio5y=Phi25+Phi45-(Phi15+Phi35);

Phio5z=Phi35+Phi45-(Phi15+Phi25);

clear Phi15 Phi25 Phi35 Phi45

Phio5xyz=sqrt(Phio5x.^2+Phio5y.^2+Phio5z.^2);

clear Phio5x Phio5y Phio5z

M5=0.25\*(M15+M25+M35+M45);

clear M15 M25 M35 M45

Phi5o=Phio5xyz.\*M5;

clear Phio5xyz M5

%channel 6

load MHA3D16

Phi16=angle(mhom);

M16=abs(mhom);

clear mhom

load MHA3D26

Phi26=angle(mhom);

M26=abs(mhom);

clear mhom

load MHA3D36

Phi36=angle(mhom);

M36=abs(mhom);

clear mhom

load MHA3D46

Phi46=angle(mhom);

M46=abs(mhom);

clear mhom

Phio6x=Phi26+Phi36-(Phi16+Phi46);

Phio6y=Phi26+Phi46-(Phi16+Phi36);

Phio6z=Phi36+Phi46-(Phi16+Phi26);

clear Phi16 Phi26 Phi36 Phi46

Phio6xyz=sqrt(Phio6x.^2+Phio6y.^2+Phio6z.^2);

clear Phio6x Phio6y Phio6z

M6=0.25\*(M16+M26+M36+M46);

clear M16 M26 M36 M46

Phi6o=Phio6xyz.\*M6;

clear Phio6xyz M6

Phiorg=Phi1o+Phi2o+Phi3o+Phi4o+Phi5o+Phi6o;

end

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**3. Higher dimensional Homodyne filter for PC-MRA**

**Main function of higher dimensional Homodyne filter for PC-MRA - Homodyne3D.m**

function[mhom]=homodyne3D(Kzfstk,m,n,o)

%Kzfstk is the zero filled 3D kspace

%m and n are the number of fractional lines in frequency and phase encode directions

[M,N,O]=size(Kzfstk);

if(mod(N,2)==0),

Nx=N/2;

else

Nx=(N+1)/2;

end

if(mod(M,2)==0),

Mx=M/2;

else

Mx=(M+1)/2;

end

if(mod(O,2)==0),

Ox=O/2;

else

Ox=(O+1)/2;

end

%weight in x-direction

d=mod(N,2);

w=prewtmtx(Nx,n,d);

W=repmat(w,M,1);

Wstkx=repmat(W,[1,1,O]);

%weight in y-direction

d=mod(M,2);

w=prewtmtx(Mx,m,d);

W=repmat(w,N,1);

W=W';

Wstky=repmat(W,[1,1,O]);

%weight in slice direction

d=mod(O,2);

w=prewtmtx(Ox,o,d);

temp=[];

temp(1,:,:)=w;

Wstkz=repmat(temp,[M,N,1]);

%Symmetric k-space in x direction

Ksymx=zeros(M,N,O);

Ksymx(:,Nx-n:Nx+n,:)=Kzfstk(:,Nx-n:Nx+n,:);

Ssymx=volumerecon(Ksymx,0);

%Symmetric k-space in y direction

Ksymy=zeros(M,N,O);

Ksymy(Mx-m:Mx+m,:,:)=Kzfstk(Mx-m:Mx+m,:,:);

Ssymy=volumerecon(Ksymy,0);

%Symmetric k-space in z direction

Ksymz=zeros(M,N,O);

Ksymz(:,:,Ox-o:Ox+o)=Kzfstk(:,:,Ox-o:Ox+o);

Ssymz=volumerecon(Ksymz,0);

WKstkx=Kzfstk.\*Wstkx;

WKstky=Kzfstk.\*Wstky;

WKstkz=Kzfstk.\*Wstkz;

WSstkx=volumerecon(WKstkx,0);

WSstky=volumerecon(WKstky,0);

WSstkz=volumerecon(WKstkz,0);

pstar1=cis(-angle(Ssymx));

pstar2=cis(-angle(Ssymy));

pstar3=cis(-angle(Ssymz));

mhom=(1/3)\*(pstar1.\*WSstkx+pstar2.\*WSstky+pstar3.\*WSstkz);

%------------------------------------CALLING FUNCTION------------------------------%

function[w]=prewtmtx(Xx,f,d)

if (d==0),

ind=-(Xx-1):Xx;

w=[2\*ones(1,Xx-(f+1)),(-1/f)\*ind(Xx-f:Xx+f)+1,zeros(1,Xx-f)];

else

ind=-Xx:Xx;

w=[2\*ones(1,Xx-(f+1)),(-1/f)\*ind(Xx-f:Xx+f)+1,zeros(1,Xx-(f+1))];

end

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**4. Calling functions for higher dimensional homodyne filter for PC-MRA - volumerecon.m**

function[Istk]=volumerecon(Kstk,fl)

%Kstk is the volume k space data of size N x M x slices x channel x partition

% if varargin is 1, the k space is rotated

Istk=[];

for nch=1:size(Kstk,4)

for partitions=1:size(Kstk,5)

temp=squeeze(Kstk(:,:,:,nch,partitions));

[N,M,O]=size(temp);

if (fl==1),

tempout=[];

for i=1:O,

sl=temp(:,:,i);

tempout(:,:,i)=sl';

end

temp=tempout;

end

Istk(:,:,:,nch,partitions) = fftshift(fftn(temp));

end

end

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Chapter V**

**1. Log-Likelihood computation using Expectation – Maximization algorithm.**

% Program gives the Log-Likelihood of a given set of intensity values.

% data : varibale to store the intensity values.

% i : varible to specify no of iterations

% gauss\_dist function for gaussian ditribution.

% Expectation computes responsibilities

% Maximization computes the weighted means and variances

data=[0.06 0.09 0.11 0.13 0.16 0.19 0.21 0.24 0.27 0.31 0.34 0.4 0.45 0.5 0.67 0.85 0.98 1.12 1.29 1.42 1.72 1.9 2.1 2.3 2.5 2.6 0.063 0.0858 0.1536 0.1761 0.1987 0.2213 0.3438 0.2664 0.3116 0.3341 0.3567 0.4244 0.5147 0.6501 0.808 0.9886 1.147 1.778 1.982 2.275 2.298 0.0627

0.0827 0.1026 0.1226 0.1625 0.1824 0.2024 0.2423 0.2623 0.2822 0.3022 0.3221 0.3421 0.362 0.402 0.5217 0.7811 1.001 1.12 1.32 1.5 1.839 1.998]; % DataSet1.

% Step 1 : Initial estimation

temp=randperm(length(data));

piecap(1)=0.5;

meucap1(1)=data(temp(1));

meucap2(1)=data(temp(2));

sigmacap1(1)=var(data);

sigmacap2(1)=var(data);

for i= 1:20

% Step 2 : Expectation Step

Qq1=gauss\_dist(data,meucap1(i),sigmacap1(i));

Qq2=gauss\_dist(data,meucap2(i),sigmacap2(i));

log\_likelihood(i)=sum(log(((1-piecap(i))\*Qq1) + (piecap(i)\*Qq2)));

responsibilities(i,:)=(piecap(i)\*Qq2)./(((1-piecap(i))\*Qq1)+(piecap(i)\*Qq2));

% Step 3 : Maximization Step

meucap1(i+1)=sum((1-responsibilities(i,:)).\*data)/sum(1-responsibilities(i,:));

meucap2(i+1)=sum((responsibilities(i,:)).\*data)/sum(responsibilities(i,:));

sigmacap1(i+1)=sum((1-responsibilities(i,:)).\*((data-meucap1(i)).^2))/sum(1- responsibilities(i,:));

sigmacap2(i+1)=sum((responsibilities(i,:)).\*((data-meucap2(i)).^2))/sum(responsibilities(i,:));

piecap(i+1)=sum(responsibilities(i,:))/length(data);

end

**----------------------------------------**Calling function **-------------------------------------------------**

function [ y ] = gauss\_dist(x,meu,sigma )

%gauss\_dist function for gaussian distribution

y=(1/(sqrt(2\*pi\*sigma)))\*exp((-(x-meu).^2)/(2\*sigma));

end