Center for EPR in vivo physiology

MATLAB toolboxes

# Information

Header of m-file should include

% Author(s): XXXX XXXX 2012; YYY YYY 2013; ZZZ ZZZ 2013.

% Center for EPR imaging in vivo physiology

% University of Chicago, 2013

% Contact: epri.uchicago.edu

# Additional materials

demo\_recon\_3D.mat

# radon

Phantoms and radon transformation toolbox

## [P] = radon\_phantom(Ph)

Create a digital phantom.

**Ph** – phantom description

* nBins – matrix size

## [P] = radon\_d2d(M, matrix\_size, radon\_pars)

Radon transformation of a digital phantom.

**matrix\_size** – matrix size in cm

**radon\_pars** – structure

* x,y,z – unit vectors for each projection [n-elements]
* size – length of projection (by default matrix\_size\*sqrt(3))

## [radon\_pars] = radon\_angle2xyz(radon\_pars)

**radon\_pars** – structure

* x,y,z – unit vectors for each projection [n-elements]
* Theta, Phi – angles for each projection [n-elements]

## [P] = radon\_c2d(P, radon\_pars)

Analytic radon transformation of phantoms

**P** – phantom description

* nBins – matrix size

**radon\_pars** – structure

* x,y,z – unit vectors for each projection [n-elements]
* size – length of projection (by default matrix\_size\*sqrt(3))

## [P] = radon\_c2d\_sphere(Phan, radon\_pars)

**[r, P] = radon\_c2d\_**sphere**(Phan, radon\_pars)**

Analytic radon transformation of sphere phantom

**Phan** - [structure] of phantom parameters

[].r - sphere radius [float, in cm]

[].offset - offset of sphere from center [float, in cm]

[].nBins - length of array [int]

**radon\_pars** - [structure] of radon transformation parameters

[].x - unit vector coordinates [array, 1D]

[].y - unit vector coordinates [array, 1D]

[].z - unit vector coordinates [array, 1D]

[].size - length of resulting projection [cm]

**P** - Projection data [array, 2D], size(P) = [projection\_size, number\_of\_projection]

**r** - Projection axis [array, 1D, in cm]

## [P] = radon\_c2d\_cube(P, radon\_pars)

Analytic radon transformation of cube phantom

**P** – phantom description

* nBins – matrix size
* a – side of cube [cm]

**radon\_pars** – structure

* x,y,z – unit vectors for each projection [n-elements]
* size – length of projection

# Example 1: Generation of analytic phantom and multistage reconstruction.

%% Generate equal angular solid angle layout of projections

fbp\_struct.nAz = 36;

fbp\_struct.nPolar = 36;

fbp\_struct.nSpec = 1;

fbp\_struct.imtype = iradon\_GetFBPImageType('XYZ');

fbp\_struct.MaxGradient = 1;

% fbp\_struct.angle\_sampling = 'UNIFORM\_ANGULAR\_FLIP';

fbp\_struct.angle\_sampling = 'UNIFORM\_SPATIAL\_FLIP';

[pars, pars\_ext] = iradon\_FBPGradTable(fbp\_struct);

% assign parameters for projection generation

radon\_pars.x = pars\_ext.kx;

radon\_pars.y = pars\_ext.ky;

radon\_pars.z = pars\_ext.kz;

radon\_pars.w = pars\_ext.w;

%% Projection generation for the 6-spheres phantom

R = 2.2; % Radius of the big sphere

Rs = 0.5; % Radius of the small spheres

a = 1.3; % offset value compared to the center

nBins = 64;

phan1 = struct('nBins', nBins, 'r', R, 'offset', [0,0,0]); % sphere data

phan2 = struct('nBins', nBins, 'r', Rs, 'offset', [0,0,0]); % sphere data

phan3 = struct('nBins', nBins, 'r', Rs, 'offset', [a,0,0]); % sphere data

phan4 = struct('nBins', nBins, 'r', Rs, 'offset', [-a,0,0]); % sphere data

phan5 = struct('nBins', nBins, 'r', Rs, 'offset', [0,a,0]); % sphere data

phan6 = struct('nBins', nBins, 'r', Rs, 'offset', [0,-a,0]); % sphere data

radon\_pars.size = 5;

P1 = radon\_c2d\_sphere(phan1, radon\_pars);

P2 = radon\_c2d\_sphere(phan2, radon\_pars);

P3 = radon\_c2d\_sphere(phan3, radon\_pars);

P4 = radon\_c2d\_sphere(phan4, radon\_pars);

P5 = radon\_c2d\_sphere(phan5, radon\_pars);

P6 = radon\_c2d\_sphere(phan6, radon\_pars);

P = 1.0\*P1 + (2.0 - 1.0)\*P2 + (1.8 - 1.0)\*P3 ...

+ (1.6 - 1.0)\*P4 + (1.4 - 1.0)\*P5 + (1.2 - 1.0)\*P6;

# iradon, iradon\_mstage, iradon\_sstage\_v2, iradon\_UIUC

Inverse radon image reconstruction toolboxes

## [object] = iradon\_d2d\_mstage(p, radon\_pars, recon\_pars)

Algorithm: multi\_stage backprojection method

Projeciton sampling method: equal angle

Number of bins in reconstructed matrix in all dimensions is equal to the number of points in projections

**p** - [array, 4D] of projections

for 3D - size(p)=[points\_in\_projection, 1, nTheta, nPhi]

for 4D - size(p)=[points\_in\_projection, nTheta, nPhi, nAlpha]

**radon\_pars** - [structure] projection parameters

[].ELA - [structure] of equal angle gradient scheme parameters

[].imtype - Image type [int, 1 for 4D, 14 for 3D], see also **iradon\_GetFBPImageType**

[].nPolar - Number of polar angles [int]

[].nAz - Number of azimuthal angles [int]

[].nSpec - Number of spectral angles[int]

[].size - length of the spacial projection [float, in cm]

**recon\_pars** - [structure] reconstruction parameters

[].nBins - Image size in voxels [int]

[].Filter - [string, ram\_lak/shepp-logan/cosine/hamming/hann]

[].FilterCutOff - Filter cut off, part of full bandwidth [float, 0 to 1]

[].InterpFactor - Projection interpolation factor, [int, 1/2/4/etc]

[].Interpolation - Inerpolation method, [string, (none)/sinc/spline/linear]

[].CodeFlag - Reconstruction code [string, C/MATLAB/FORTRAN]

**object** - reconstructed object

## [object] = iradon\_3d\_sstage\_v2(p, radon\_pars, recon\_pars)

Inverse radon single stage transformation by Zhiwei

**radon\_pars** – structure

* x,y,z – unit vectors for each projection [n-elements]
* w – projection weight coefficients [n-elements]
* size – length of projection (by default matrix\_size\*sqrt(3)

## [object] = iradon\_uiuc\_ls (p, radon\_pars, recon\_pars)

Inverse radon transformation using re-gridding in k-space and LS optimization

[object] = iradon\_uiuc\_ls(p, radon\_pars, recon\_pars);

Algorithm: see inside /iradon\_UIUC directory

**p** - [array, 2D] of projections in spatial domain

for 3D - size(p)=[points\_in\_projection, n\_projections]

**radon\_pars** - [structure] projection parameters

[].x - [array, 1D] projection unit vector component

[].y - [array, 1D] projection unit vector component

[].z - [array, 1D] projection unit vector component

[].size - length of projection [float, in cm]

**recon\_pars** - [structure] reconstruction parameters

[].nBins - Image size in voxels [int]

[].size - Image size [float, in cm]

**object** - reconstructed object

## [stat]=iradon\_verify(M, Mprime)

Compare two images, report statistics

## [Pzp, zeropadding] = iradon\_zeropadding(P, zeropadding)

zeropadding of the projection data first dimension (used in many reconstruction routines)

**P** - Data, [array, any dimensions]

**zeropadding** - Zeropadding factor [int, >= 1]

**Pzp** - Output data [array, any dimensions]

## [nimtype, simtype] = iradon\_GetFBPImageType(imtype);

Returns numerical and string representation of image type

**imtype** - Image type [int, 1 to 14]

or

- Image type [XB/YB/ZB/XYB/XZB/YZB/XYZB/ X/Y/Z/XY/XZ/YZ/XYZ]

**nimage** - Image type [int, 1 to 14]

**simtype** - Image type [XB/YB/ZB/XYB/XZB/YZB/XYZB/ X/Y/Z/XY/XZ/YZ/XYZ]

## [out, suplementary\_out] = iradon\_FBP\_grad\_table(FBP)

uniform angular and uniform spatial gradient table for FBP images

**FBP** - [structure] of gradient scheme parameters

[].imtype - Image type [int, 1 for 4D, 14 for 3D]

[].nPolar - Number of polar angles [int]

[].nAz - Number of azimuthal angles [int]

[].nSpec - Number of spectral angles[int]

[].size - length of the spatial projection [float, in cm]

[].CoordPole - [1/2/3] or <X/Y/(Z)>

[].MaxGradient - Maximum gradient [float, in G/cm]

[].angle\_sampling - type of angle sampling

UNIFORM\_ANGULAR - uniform angular

UNIFORM\_ANGULAR\_FLIP - uniform angular with optimized jumps

UNIFORM\_SPATIAL\_FLIP - uniform solid angle with optim. jumps

**out** - [structure] of radon transformation parameters

[].GradX - Gradient component [array, 1D]

[].GradY - Gradient component [array, 1D]

[].GradZ - Gradient component [array, 1D]

**suplementary**\_out - [structure] of radon transformation parameters

[].kx - k-space unit vector component [array, 1D]

[].ky - k-space unit vector component [array, 1D]

[].kz - k-space unit vector component [array, 1D]

[].w - projection weight factor [array, 1D]

# Example 2: Image reconstruction using different algorithms

% see example 1 for projection generation

%% FBP multistage image reconstruction

PP = zeros(phan1.nBins, fbp\_struct.nAz\*fbp\_struct.nPolar);

PP(:,pars.gidx) = P;

% convert serial projection layout into matrix form required

% for multistage reconstruction

Pela = reshape(PP, [size(P,1), 1, fbp\_struct.nAz, fbp\_struct.nPolar]);

% Interpolate to uniform angular

switch fbp\_struct.angle\_sampling

case {'UNIFORM\_SPATIAL','UNIFORM\_SPATIAL\_FLIP'}

Pela=iradon\_InterpToUniformAngle(Pela, 'imgData');

end

% projection visualization

% MatrixGUI(phantom)

radon\_pars.ELA = fbp\_struct;

recon\_pars.size = 5; % this par is ignored, radon\_pars are used

recon\_pars.nBins = 128; % this par is ignored, radon\_pars are used

recon\_pars.Filter = 'ram-lak';

recon\_pars.FilterCutOff = 1.0;

recon\_pars.Interpolation = 'spline';

recon\_pars.InterpFactor = 2;

recon\_pars.CodeFlag = 'C';

recon\_pars.zeropadding = 2; % any number >= 1

% call the reconstruction program and display result

mat\_recFXD = iradon\_d2d\_mstage(Pela, radon\_pars, recon\_pars);

ImageBrowserGUI(mat\_recFXD);

%% UIUC code least square + re-gridding in k-space code

addpath(genpath('Z:\CenterMATLAB\iradon\_UIUC'));

recon\_pars.size = 5;

recon\_pars.zeropadding = 2; % 1,3,5 only

recon\_pars.nBins = 64;

% call the reconstruction program and display result

mat\_recFXD = iradon\_uiuc\_ls(P, radon\_pars, recon\_pars);

ImageBrowserGUI(real(mat\_recFXD))

%% Zhiwei's single-stage 3D FBP algorithm on CPU/GPU

addpath('Z:\CenterMATLAB\iradon\_sstage\_v2');

% set up the reconstruction parameters

recon\_pars.nBins=64;

recon\_pars.FilterCutOff=1; % [0-1] full bandwidth of projections

recon\_pars.size=5;

recon\_pars.display=0;

recon\_pars.Filter=2; %%% 2 means 3-points method, which is faster and has a good compromise. 4 means R-L filter, which has the best spatial resolution. There are 6 methods. see in the reoconstruction function.

recon\_pars.processor=2; % 1 is for cpu; 2 is for GPU

recon\_pars.interp\_method=1; % 0 is zero-rank interpolation method; 1 is linear interpolaiton method,2 is the spline interpolation.

recon\_pars.tasksliced=1; % [0/1] use 1 to break task in multiple executions

% call the reconstruction program and display result

object=iradon\_3d\_sstage\_v2(P,radon\_pars,recon\_pars);

ImageBrowserGUI(object);

# Reports (GUIs)

Visualization software

## MatrixGUI(any\_matrix)

MatrixGUI(any\_matrix, any\_matrix2)

GUI for visualization 1D traces of arbitrary matrix

**any\_matrix** – any matrix 2D to 4D

**any\_matrix2** – any matrix 2D to 4D

## ImageBrowserGUI(matrix)

GUI for visualization of 3D and 4D matrices. Contains 3 orthogonal plot 2D displays and 1D display for fourth dimension. Number of utilities for simple data processing.

**matrix** – any matrix 3D to 4D

# Fitting

Data fitting toolbox

## [fitted] = fit\_exp\_no\_offset(fit\_y, tt)

T2 fitting function using built in matlab function “fminsearch.m” which uses unconstrained nonlinear optimization to minimize the difference between the data and the fitted function, starting from an initial guess. Fits to find A and T2 using function:

(where tt = 2τ).

**fit\_y** – matrix of voxel decay data

* fit\_y has dimensions [# voxels to be fit, # of points in decay to be fit]
  + For T2 data usually 5 points are acquired and a 6th point at a long tau (~30μs) is added at y = 0

**tt** –2\*( taus used)

**fitted** – structure, output of results from fitting

* fitted.amp – amplitute (without any concentration corrections) found from fitting
* fitted.T2 – T2 found from fitting
* fitted.error – difference between data points and fit using built in “fminsearch.m”
* fitted.err\_mask – keeps track of voxels to throw out if fminsearch could not fit and needed to exit

## [fitted] = fit\_recovery\_3par(fit\_y, TT)

T1 fitting function using built in matlab function “fminsearch.m” which uses unconstrained nonlinear optimization to minimize the difference between the data and the fitted function, starting from an initial guess. Fits to find A, B, and T1 using function:

(where TT = T delays used).

**fit\_y** – matrix of voxel decay data

* fit\_y has dimensions [# voxels to be fit, # of points in decay to be fit]
  + For T1 data usually 8 points are acquired (final point has no inversion to simulate T = infinity)

**TT** – T delays used

**fitted** – structure, output of results from fitting

* fitted.amp – amplitute (without any concentration corrections) found from fitting (A)
* fitted.T1 – T1 found from fitting
* fitted.error – difference between data points and fit using built in “fminsearch.m”
* fitted.err\_mask – keeps track of voxels to throw out if fminsearch could not fit and needed to exit
* fitted.inv – Inversion found from fitting (B)
* fitted. recovery – shows the function used to fit

## [fitted] = fit\_recovery\_simultaneous(fit\_y, tt, TT)

T1/T2 fitting function that fits T1 and T2 simultaneously using built in matlab function “fminsearch.m” which uses unconstrained nonlinear optimization to minimize the difference between the data and the fitted function, starting from an initial guess. Fits to find A, B, T1 and T2 using function:

(where TT = T delays used and tt = 2τ).

**fit\_y** – matrix of voxel decay data

* fit\_y has dimensions [# voxels to be fit, # of points in decay to be fit]
  + For T1/T2 data usually 12 points are acquired (8th point has no inversion to simulate T = infinity)

**tt** – 2τ used

**TT** – T delays used

**fitted** – structure, output of results from fitting

* fitted.amp – amplitute (without any concentration corrections) found from fitting (A)
* fitted.T1 – T1 found from fitting
* fitted.T2 – T2 found from fitting
* fitted.error – difference between data points and fit using built in “fminsearch.m”
* fitted.err\_mask – keeps track of voxels to throw out if fminsearch could not fit and needed to exit
* fitted.inv – Inversion found from fitting (B)
* fitted. ff – ???

## [fitted] = fit\_recovery\_saturated(fit\_y, tt, TT, TRs)

T1/T2 fitting function that fits T1 and T2 simultaneously and tries to account for saturation from finite repetition time (TR) using built in matlab function “fminsearch.m” which uses unconstrained nonlinear optimization to minimize the difference between the data and the fitted function, starting from an initial guess. Fits to find A, B, T1 and T2 using function:

(where TT = T delays used, tt = 2τ, and TRs are the repetition times used).

**fit\_y** – matrix of voxel decay data

* fit\_y has dimensions [# voxels to be fit, # of points in decay to be fit]
  + For T1/T2 data usually 12 points are acquired (8th point has no inversion to simulate T = infinity)

**tt** – 2τ used

**TT** – T delays used

**fitted** – structure, output of results from fitting

* fitted.amp – amplitute (without any concentration corrections) found from fitting (A)
* fitted.T1 – T1 found from fitting
* fitted.T2 – T2 found from fitting
* fitted.error – difference between data points and fit using built in “fminsearch.m”
* fitted.err\_mask – keeps track of voxels to throw out if fminsearch could not fit and needed to exit
* fitted.inv – Inversion found from fitting (B)
* fitted. ff – ???

## [fitted] = fit\_lookuptable\_T2(fit\_y, tt)

T2 fitting function using look up table. Curves with known relaxation from the look up table are compared to data by finding which entry in the look up table has the largest dot product with the data. Look up table curves are generated using the function:

(where tt = 2τ, and A = 1).

Once normalized data has been matched to a look up table entry to find T2, true amplitude (A) is found by calculating:

where Si are the data points.

**fit\_y** – matrix of voxel decay data

* fit\_y has dimensions [# voxels to be fit, # of points in decay to be fit]
  + For T2 data usually 5 points are acquired and a 6th point at a long tau (~15μs) is added at y = 0

**tt** – 2\*( taus used)

**fitted** – structure, output of results from fitting

* fitted.amp – amplitute (without any concentration corrections) found from fitting
* fitted.T2 – T2 found from fitting

## [fitted] = fit\_lookuptable\_T1(fit\_y, tt, TT)

T1 fitting function using look up table. Curves with known relaxation from the look up table are compared to data by finding which entry in the look up table has the largest dot product with the data. Look up table curves are generated using the function:

(where TT = T delay, and A = 1).

The inversion recovery curves in the dictionary are then normalized to the maximum point to simulate full recovery. They are then inverted and normalized by dividing by (1 – min) to simulate full inversion. Similary the data is first normalized to the maximum point, inverted and then divided by (1 – min). Once normalized data has been matched to a look up table entry to find T1, true amplitude (A) is approximated as the maximum data point corrected for T2 influence by assuming T2 = T1 and dividing by:

**fit\_y** – matrix of voxel decay data

* fit\_y has dimensions [# voxels to be fit, # of points in decay to be fit]
  + For T2 data usually 5 points are acquired and a 6th point at a long tau (~30μs) is added at y = 0

**tt** – 2\*( taus used)

**TT** – T delays used

**fitted** – structure, output of results from fitting

* fitted.amp – amplitute (without any concentration corrections) found from fitting
* fitted.T1 – T1 found from fitting

## [fitted] = fit\_lookuptable\_T1T2(fit\_y, tt, TT)

T1/T2 fitting function using look up table. Curves with known relaxation from the look up table are compared to data by finding which entry in the look up table has the largest dot product with the data. Separate look up tables are generated for T1 and T2. Curves for T2 are generated using the function:

(where tt = 2τ, and A = 1).

Curves for T1 are generated using the function:

(where TT = T delays, and B = 1).

First the T2 portion of the data are matched to the T2 look up table to find T2 values. Then true amplitude (A) is found using:

where Si are the data points. Then the T1 portion of the data are corrected for amplitude and the T2 effect by dividing by:

(where tt’ is 2τ used for T1 measurements)

The data is then inverted (data’ = 1-data) and compared to the T1 look up table.

**fit\_y** – matrix of voxel decay data

* fit\_y has dimensions [# voxels to be fit, # of points in decay to be fit]
  + F For T1/T2 data usually 12 points are acquired (8th point has no inversion to simulate T = infinity)

**tt** – 2\*( taus used)

**TT** – T delays used

**fitted** – structure, output of results from fitting

* fitted.amp – amplitute (without any concentration corrections) found from fitting
* fitted.T1 – T1 found from fitting
* fitted.T2 – T2 found from fitting