Fast Imaging Library

of the

National Center for Image-Guided Therapy

Documentation for Release Version 2.0

July 24, 2014

Developed by the Medical Imaging Physics Group
Dept. of Radiology
Brigham and Women's Hospital
75 Francis Street
Boston, MA 02115

Please direct inquires to: ncigt-imaging-toolkit@bwh.harvard.edu

Supported by the United States National Institute of Heath
NIH Grant U41 RR019703-01A.

Contents

1 The NC-IGT Fast Imaging Library

In MR imaging applications, the dominant constraint to image acquisition is *time*. As the field has matured, a number of methods have been developed to reduce the length of time needed to acquire an image by reducing the amount of data needed to reconstruct a clinically viable image. This reduction in acquired data requirements is achieve through encodings that are complementary to the fundamental Fourier encoding employed in MRI. This includes temporal-domain encodings, such as UNFOLD, and spatial-domain encodings, as in parallel MR imaging methods such as SENSE and GRAPPA.

This library of functions provides a number of reconstruction algorithms that accurately employ these advanced MR imaging methods. This includes

- UNFOLD
- Parallel Imaging Methods
  - SENSE
  - VD-SENSE
  - SPACE RIP
  - GRAPPA

as well as a number of associated MR image reconstruction algorithms, including

- EPI Nyquist ghost correction
- Homodyne processing of partial-Fourier data
- Gradient field inhomogeneity correction (gradwarp)

As of version 0.96, the library is compatible with both GE and Siemens MR scanners. For GE MR scanners, file format support includes raw data output using the "P-file" format, and the "vrgf.dat" and "ref.dat" files to respectively correct for ramp-sampling and gradient-offset phase correction during EPI acquisitions.

This library was developed at the National Center for Image Guided Therapy by

Medical Imaging Physics Group

Dept. of Radiology

Brigham and Women's Hospital

75 Francis Street

Boston, MA 02115

Please direct inquires to: `ncigt-fil@bwh.harvard.edu`

Acknowledgments

The development of this library was supported by the United States National Institute of Heath (NIH) under Grant U41 RR019703-01A.

Contributors: Bruno Madore (BWH), W. Scott Hoge (BWH), Greg Kirk (formerly BWH), Steven J. Haker (formerly BWH)

The FFT calculations are provided by the KISS FFT library.

Length 2^N FFTs are handled by code derived from OOURA FFT:

With version 0.9, the library now contains multiple functions to compute the Singular Value Decomposition (SVD) of a matrix.

The core SVD functions are an implementation of the bilateral diagonalization approach first described by Golub et. al. This algorithm is known to be extremely stable, and is drawn from the Colorado School of Mines `Center for Wave Propogation` Seismic Un∗x C library.

To support both the image-phase-alignment and virtual-body-coil functions, a second SVD implementation using a Lanczos iterative algorithm is now included. For the purpose of estimating the singular vectors associated the largest singular value of large matrices, this Lanczos approach is significantly faster than the more stable bi-diagonalization approach.

This code is derived from the netlib SVDPACK, via the SVDLIBC library (http://tedlab.mit.edu-:16080/~dr/SVDLIBC/), and modified to be thread-safe. This code is

```
-----------------------------------------------
(c) Copyright 2003, Douglas Rohde

adapted from SVDPACKC, which is

(c) Copyright 1993, University of Tennessee
 All Rights Reserved
-----------------------------------------------
```

To support GROG functions, the library has begun to transition to Eigen3 for matrix decompositions since version 2.0. Version 2.0 uses eigen-3.1.4 (with the EIGEN_MPL2_ONLY flag), which is distributed under the MPL2 license:

```
// This Source Code Form is subject to the terms of the Mozilla
// Public License v. 2.0. If a copy of the MPL was not distributed
// with this file, You can obtain one at  http://mozilla.org/MPL/2.0/.
```

2 Quick Start Guide

The goal of this guide is to get a new user of the Fast Imaging Library (FIL) up and running as quickly as possible, using provided demo programs.

2.1 Prerequisites

Currently [release 2.0], the library is available on Linux (both 64-bit and 32-bit), Windows (MSWin64), and Mac OS X (darwin-intel).

The library provides a binary file for the processing functions, with MEX files to call the functions from Matlab. Thus, a C compiler is required. The library has been tested with `gcc` on Linux and Mac OS X, and MS Visual Studio 2008 on Windows.

2.2 Setup

First, extract the library into a working directory. Below, we will refer to this directory as the `[igt_fil_basepath]`. This should create a directory tree that includes `./demos`, `./include`, `./lib`, and `./source_code/matlab` paths.

2.3 MEX file demos

2.3.1 Build the MEX files

MEX files are C programs that are callable from Matlab. The FIL distribution contains a number of examples in the `source_code/matlab/mex` directory. To set up MEX file use, do the following:

- chdir to `[igt_fil_basepath]/source_code/matlab/mex`
- compile each of the matlab functions by calling `buildall.m`

Examples of how to use the MEX functions are located in `[igt_fil_basepath]/demos/mex_demo/`

After a `cd` to this directory, be certain to add both the matlab script and the path MEX paths to the matlab path:

```
addpath <igt_fil_basepath>/source_code/matlab
addpath <igt_fil_basepath>/source_code/matlab/mex
```

The remaining portion of this section presents each the MEX demos in turn.

2.3.1.1 Loading the Demo Data

all of the MEX demos use the same raw data. This can be read into Matlab via readnd

```
k = readnd('phantom.dat');
```

This should provide 8-channels of k-space data for a resolution phantom (acquired at 1.5T using a fast spin echo sequence).

2.3.1.2 Zero-pad k-space Demo (util_demo.m)

The file `util_demo.m` provides a simple example on how to zero-pad an accelerated k-space data set. First, a sampling pattern is generated. The input sampling pattern is structured with each row containing a `ky-z` coordinate location. Since there is only one slice here, all of the `z` coordiantes are the same. `ky` is subsampled by 2, capturing all 'odd' lines.

```
Yo = [ vec(1:2:256) ones(128,1) ];
```

This sampling pattern is then passed to the kstretch function via

```
ko = kstretch( Yo, squeeze(k(:,Yo(:,1),1)), [256 256 1] );
```

`ko` should now be a zero-padded version of the subsampled `k` data. This can be visualized using the following Matlab command:

```
pltcmplx( k(:,:,1), ko(:,:,1) );
```

2.3.1.3 Partial Fourier Demo (pF_demo.m)

The library includes functions to perform homodyne detection/syncronization for partial-Fourier data sets (as per Noll, Nishimura, and Macovski. IEEE Trans Medical Imaging. 10(2):154-163, 1991). The MEX file `correct_pF` performs this process by calling the partial_Fourier :

```
Y = 1:146;                                  % specify the lines to use
k0 = correct_pF( k(:,Y,1), [ 256 256 ], 1 );  % perform the correction
im(k0);                                     % visualize the result
```

The 3rd argument of the MEX function can be used to switch between a (0) step filter or a (1) ramp filter. The step filter carries better SNR, while the ramp filter is more tolerant of k-space that is off-center.

2.3.1.4 Parallel Imaging Demo (pmri_demo.m)

Three parallel imaging algorithms are included in the library: SENSE, GRAPPA, and SPACE RIP (see Parallel Imaging Algorithms for more details). This demo demonstrates each of them on the same data set. First, the data is loaded and a subsampling pattern is generated.

```
k = readnd('phantom.dat');

N = 256;

k = k( (size(k,1)-N)/2+(1:N), (size(k,1)-N)/2+(1:N), : );

% form a variable density sampling pattern, 4x-2x-1x, with 3x total
% acceleration (suitable for self-referenced reconstructions)
Z = zeros(N,1);
Z( 2:4:end) = 1;
Z( (N/2) + (-(N/8-2):2:(N/8)) ) = 1;
Z( (N/2) + (-(N/32-2):(N/32)) ) = 1;
Y = find(Z);
```

Next, coil sensitivity maps are generated from the subsampled data set:

```
% generate the coil sensitivity maps
W = geyser( k(Y,:,:), Y(:), [N N 8] );
```

Finally, images are formed from the subsampled data using various reconstruction algorithms.

- SPACE RIP, using the LSQR-Hybrid algorithm to perform automatic regularization.

```
% solve the system using LSQR-Hybrid space rip
Isr = srlsqr( W, ifft(fftshift(k(Y,:,:),2),[],2), Y(:) );
```

- GRAPPA

```
% solve the system using GRAPPA
[Fg,Ig2]  = grappa( size(W), k(Y,:,:), Y(:) );
```

`Ig2` here is the root-sum-of-squares image. If phase needs to be retained, reconstruct the image using coil sensitivity maps instead:

```
Ig  = sum( conj(W).*ifft2(fftshift(fftshift(Fg,1),2)), 3 );
```

- RLS-GRAPPA

  This algorithm performs the GRAPPA reconstruction in hybrid-space (ky-x) and employs an RLS adaptive filter to rapidly compute the reconstruction parameters. In certain cases, it is significantly faster than standard GRAPPA.

```
[Frlsg, Irlsg] = rlsgrappa_x( size(W), ifft(fftshift(k(Y,:,:),2),[],2), Y(:) );
```

- standard Cartesian SENSE

```
k2 = zeros(size(k));
k2( 1:4:end,:,: ) = k( 1:4:end,:,:) * 4;  % '*4' to compensate for the
                                          % sparse sampling
K2 = ifft2( fftshift(fftshift(k2,1),2) );
Ics = sense( permute(W,[2 1 3]), permute(K2,[2 1 3]), 4 ).';
```

- and Variable Density SENSE

```
% solve the system using a variable density SENSE reconstruction
k3 = prep_vds( Y(:), permute(k(Y,:,:),[2 1 3]), [256 256 8] );
k3 = permute( k3, [2 1 3]);

K3 = ifft2( fftshift(fftshift(k3,1),2) );

Ivdcs = sense( permute(W,[2 1 3]), permute(K3,[2 1 3]), 4 ).';
```

Finally, the results are displayed

```
im([ Ig Isr; Ics Ivdcs ]);
```

2.3.1.5 UNFOLD Demo (unfold_demo.m)

UNFOLD employs temporal variations in the sampling pattern to encode aliasing artifacts and then filters the results to suppress the encoded artifacts.

This demo uses a temporal data set

```
a = readnd('zoomed.dat');
```

and all processing is done within the MEX file

```
z = unfold(a);

im([ a squeeze(z)]);
```

2.4 Important Details

2.4.1 Indexing

To keep with standard C convention, all indexing in the library starts at zero. This is particularly important in regards to specifying which phase-encode lines are acquired. So, if in Matlab the phase encode list runs from [1 .. M], the same phase encode list in the library would run from [0 .. (M-1)]. The MEX functions provided with the library perform this transition automatically to give a transparent interface to Matlab.

2.4.2 Data Ordering

Except for certain parallel imaging functions, all of the data is ordered as `kx-ky-z-timeframe-coil`

The parallel imaging algorithms that do not follow this convention include

1. self_ref_b1_via_geyser
2. grappa
3. all sprip_ functions, such as sprip_lsqr_recon

which are ordered `ky-kx-coil` for historical reasons. In short, if a function call does not include a temporal and/or Z component as an argument, then the data should be short-ordered, with the subsampled `ky` data as the first dimension.

2.4.3 Anatomy of a MEX file

Here we describe how the MEX files are structured, using `unfold.c` as an example.

- preamble: declare macros to give a rational name to the pointers associated with data input/output

```
//
#define K_IN     prhs[0]
#define FREQ_IN  prhs[1]
#define WIDTH_IN prhs[2]

//
#define K_OUT   plhs[0]
#define NY_OUT  plhs[1]
```

- sanity: check that the number of inputs/outputs conform

```
if (nrhs < 1) {
  mexPrintf("At least 1 input argument is required.");
  return;
} else if (nlhs > 2) {
  mexPrintf("Too many output arguments.");
  return;
}
```

- pre: copy the data from Matlab arrays into C data structures

```
tmpR = mxGetPr(K_IN);
tmpI = mxGetPi(K_IN);
szK = mxGetNumberOfDimensions(K_IN);
dimK = mxGetDimensions(K_IN);

if ( !mxIsDouble(K_IN) || !mxIsComplex(K_IN) ) {
  mexPrintf( "1st input: Need to declare a complex "
             " array for the input k-space data\n" );
  return;
} else {
```

```
  // header variables that need to be populated
  hdr.recon_filt = 0.1;         //  Width of UNFOLD filters, in frac of bw
  // default values
  hdr.Nz_proc = 1;
  hdr.ncoils = 1;

  hdr.Ny_proc = dimK[0];
  hdr.Nx = dimK[1];
  if (szK == 2) {
    hdr.Ny_proc = 1;          // number of temporal frames to process
    hdr.Nf = dimK[1];
  } else if (szK == 3) {
    hdr.Nf = dimK[2];         // number of temporal frames to process
  } else if (szK == 4 ) {
    hdr.Nz_proc = dimK[2];
    hdr.Nf = dimK[3];         // number of temporal frames to process
  } else { // (szK == 5 )
    hdr.Nz_proc = dimK[2];
    hdr.Nf = dimK[3];         // number of temporal frames to process
    hdr.ncoils = dimK[4];
  }
  hdr.Nf_proc = set_nfproc(hdr.Nf, 0);

  //
  // printf("sz = ");
  // for (cnt=0;cnt<szK;cnt++) {
  //   printf("%d ",dimK[cnt]);
  // }
  //
  prd = hdr.Nf * hdr.Nz_proc * hdr.Ny_proc * hdr.Nx * hdr.ncoils;

  data_in = (COMPLEX *) malloc( prd * sizeof(COMPLEX) );

  // mexPrintf("prd = %d\n",prd);

  for (cnt=0;cnt<prd;cnt++) {
    data_in[cnt].re = tmpR[cnt];
    data_in[cnt].im = tmpI[cnt];
  }

}
```

- process: perform the data processing (i.e. call FIL functions)

```
filter_dc = (float *) malloc(hdr.Nf_proc*sizeof(float));
filter_ny = (float *) malloc(hdr.Nf_proc*sizeof(float));
filter_dckept = (float *) malloc(hdr.Nf_proc*sizeof(float));
filter_dcny = (float *) malloc(hdr.Nf_proc*sizeof(float));
near_dc = (short *) malloc(hdr.Nf_proc*sizeof(short));

// Create plans for a 1D FFT along the time (or frequency) axis
fft_mngr_init( &mngr );

// Set-up the unfold filters
setup_unfold( &hdr, hdr.Nf_proc, filter_dc, filter_ny, filter_dckept,
              filter_dcny,
              near_dc, &filt_w );


full = (COMPLEX *) malloc( hdr.Nf_proc * hdr.Nz_proc * hdr.Ny_proc * hdr.Nx
                           * hdr.ncoils * sizeof(COMPLEX) );

synth_frames( hdr.ncoils, hdr.Nf, hdr.Nz_proc, hdr.Ny_proc, hdr.Nx,
              data_in, hdr.Nf_proc, full );

// FFT from time to temporal frequency domain, with plan ptf (forward)
fft_t( hdr.ncoils, hdr.Nf_proc, hdr.Nz_proc, hdr.Ny_proc, hdr.Nx,
       'f', &mngr, 0, full, full );

if (mode == 'l') {
  // Keep only the DC along temp. freq. domain
  unfold_filter( hdr.ncoils, hdr.Nf_proc, hdr.Nz_proc, hdr.Ny_proc,
                 hdr.Nx, filter_dckept, full );
} else if (mode == 'm') {
  // Filter out DC and Nyquist region (temp. freq. domain)
  unfold_filter( hdr.ncoils, hdr.Nf_proc, hdr.Nz_proc, hdr.Ny_proc,
                 hdr.Nx, filter_dcny, full );
} else { // (mode == 'h')
  // Filter out Nyquist region (temp. freq. domain)
```

```
  unfold_filter( hdr.ncoils, hdr.Nf_proc, hdr.Nz_proc, hdr.Ny_proc,
                 hdr.Nx, filter_ny, full );
}

fft_t( hdr.ncoils, hdr.Nf_proc, hdr.Nz_proc, hdr.Ny_proc, hdr.Nx,
       'b', &mngr, 0, full, full );
```

- post: copy the output data from C data structures into Matlab arrays

```
sz[0] = hdr.Nx;
sz[1] = hdr.Ny_proc;
sz[2] = hdr.Nz_proc;
sz[3] = hdr.Nf_proc;
//
  write_nd_data( "unf_out.nd", full, "dblc", 4, sz );

if (nlhs>0) {

  sz[0] = hdr.Ny_proc;
  sz[1] = hdr.Nx;

  szK = mxGetNumberOfDimensions(K_IN);
  if (szK == 2) {
    scl = 2;
    sz[1] = hdr.Nf;
  } else if (szK == 3) {
    scl = 3;
    sz[2] = hdr.Nf;
  } else if (szK == 4) {
    scl = 4;
    sz[2] = hdr.Nz_proc;
    sz[3] = hdr.Nf;
  } else if (szK == 5) {
    scl = 5;
    sz[2] = hdr.Nz_proc;
    sz[3] = hdr.Nf;
    sz[4] = hdr.ncoils;
  }
  K_OUT = mxCreateNumericArray( scl, sz,
                                mxDOUBLE_CLASS,
                                mxCOMPLEX);
  tmpR = mxGetPr(K_OUT);
  tmpI = mxGetPi(K_OUT);

  prd = hdr.Nz_proc * hdr.Ny_proc * hdr.Nx ;

  for (cntZ=0;cntZ<hdr.ncoils;cntZ++) {
    for (cnt=0;cnt<prd*hdr.Nf; cnt++) {
      tmpR[ cnt + cntZ*hdr.Nf*prd ] =
        full[ cnt + cntZ*hdr.Nf_proc*prd ].re;
      tmpI[ cnt + cntZ*hdr.Nf*prd ] =
        full[ cnt + cntZ*hdr.Nf_proc*prd ].im;
    }
  }

}
```

2.5 C test files

2.5.1 DICOM editing demo

The code included in

```
demos/dicom_demo/
```

demonstrates how to edit the value associated with a specific DICOM tag in a file, and write out the new DICOM image file.

2.5.2 .nd to .png Demo

This function is designed to convert the FIL .nd output data files into an image viewable on any platform. It uses the PNG and ZLIB libraries to accomplish this. These libraries are standard on Linux and Mac OS X, but may need to be installed on Windows. To build them them on Windows, do the following

- build ZLIB
  - download the ZLIB source distribution from `http:///www.zlib.net`
  - unzip the source into a working directory (say, `C:\src`)
  - cd to the zlib directory, and compile using the command `nmake -f win32\Makefile.msc`
- build LIBPNG
  - download the PNG source distribution from `http://www.libpng.org/pub/png/libpng.html`
  - unzip the source into the working directory
  - cd to the lpng directory, and compile using the command `nmake -f lpng/scripts/makefile.-vcwin32`

Once the libraries are built, modify the `Makefile.win32` file in the display directory to point to the location of the ZLIB and LIBPNG library and header files. `nd2png` should then build properly.

2.5.3 Demo for simple display, using ImLib

ImLib is a general image loading and rendering library, designed to simplify the generating drawable in X-Windows systems. It is included by default in many Linux distributions. Source code for ImLib can be found here: `ftp://ftp.-gnome.org/pub/GNOME/sources/imlib/1.9/`

This demo code included in the

```
./demos/display/
```

directory uses ImLib to load data from a file stored in the ND-data format, and displays the data in a simple viewer.

The display demos use an "X Server" for data display. This is available by default on Linux and Mac. For Windows, a free version is available here: `http://xming.sourceforge.net/`

Code is also included that links to ImageMagick to display a set of images. This code works on both Linux and Windows. To setup ImageMagick on Windows, either (1) download the latest binary version along with the MSVC 2008 Portability Pack, or (2) download the source and compile. [ImageMagick binarys are currently compiled with MSVC 2008, which uses 'manifests' to manage DLL linking order. VC++ 6 doesn't have this, so programs compiled in VC++ 6 but linked to the ImageMagick DLL's will show an "R6034 error" everytime the program runs.]

3 Matlab Scripts

A few Matlab scripts are included to provide an interface to the Fast Imaging Library functions

3.1 Data I/O scripts

3.1.1 read_dump

`read_dump.m`: This script is an interactive way to read in binary data, output by the function dump_out

3.1.2 readnd

`readnd.m`: This script reads in data that was output by write_nd_data

Usage:

```
A = readnd( filename );
```

- ***filename*** A string containing the filename to read.
- ***A*** A matrix holding the data in the file.

3.1.3 savend

`savend.m`: This script writes to disk a Matlab matrix in a format that is readable in C by the read_nd_data function.

Usage:

```
savend( filename, A, format );
```

- ***filename*** The name of the output file.
- ***A*** The Matlab variable to output.
- ***format*** One of the following 3-character strings: 'int' for integers, 'flt' for floating point numbers, or 'dbl' for double precision numbers.

3.2 Data visualization scripts

3.2.1 im

`im.m`: A graphical interface program to view image and k-space data. It accepts upto 4-dimensional data, and can display the magnitude, real, imaginary, or phase of the data. Also includes Linkoping's "GOP" colormap format to visualize magnitude and phase simultaneously.

3.2.2 pltcmplx

`pltcmplx.m`: A graphical interface to compare two or three complex valued matrices, ploting real/imag or magnitude/phase for each column of the data.

4 Module Index

4.1 Modules

Here is a list of all modules:

5 Module Documentation

5.1 Artifact Suppression via UNFOLD

Functions

- int filter_t_init (int n, int x, int y, int c, double ∗a, double ∗b, filter_t_obj ∗obj)
- int filter_t_quit (filter_t_obj ∗obj)
- int filter_t_step (filter_t_obj ∗obj, COMPLEX ∗dataI, COMPLEX ∗dataO)
- void prep_unfold_dc (short Nx, short N_acq, short Nf, short ncoils, fil_fft_mngr ∗mngr, KCOORD ∗kline_o, KCOORD ∗kline_e, short fr1, COMPLEX ∗data, COMPLEX ∗buf_lg, COMPLEX ∗buf_sm, float ∗filter_dc, float ∗filter_dckept, short ∗near_dc, short filt_w, short Nf_proc, KCOORD ∗kline_c, COMPLEX ∗data_dc)
- int set_nfproc (short Nf, short flag)
- void setup_unfold (SCAN_INFO ∗hdr_ptr, int Nt, float ∗filter_dc, float ∗filter_ny, float ∗filter_dckept, float ∗filter_dcny, short ∗near_dc, short ∗filt_w_ptr)
- void synth_frames (short ncoils, short Nf, short Nz, short Ny, short Nx, COMPLEX ∗data_in, short Nf_proc, COMPLEX ∗data_out)
- void synth_frames_cine (short ncoils, short Nf_full, short fr1, short Nf, short N_acq, short Nx, short ky, KCOORD ∗kline_o, KCOORD ∗kline_e, COMPLEX ∗data_in, short Nf_proc, COMPLEX ∗data_out)
- void transfer_dc (short dir, short ncoils, short Nf, short Nz, short Ny, short Nx, short filt_w, short ∗near_dc, float ∗filter, COMPLEX ∗lg_array, COMPLEX ∗sm_array)
- void unfold_filter (short ncoils, short Nf, short Nz, short Ny, short Nx, float ∗filter, COMPLEX ∗data)
- void unfold_recombine (short Nx, short Ny_proc, short Nz_proc, short Nf_proc, short ∗near_dc, short filt_w, COMPLEX ∗dc, COMPLEX ∗full)

5.1.1 Detailed Description

The UNFOLD approach employs temporal domain processing to suppress artifacts.

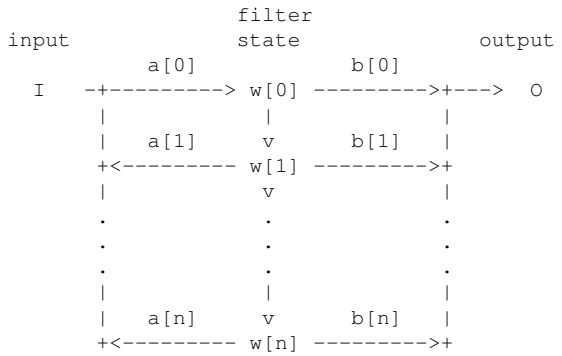
The method is described in "Unaliasing by Fourier-encoding the overlaps using the temporal dimension" (UNFOLD). Bruno Madore, Gary H. Glover, and Norbert J. Pelc. *Magn Reson Med.* 42(5):813-826, Nov 1999. [DOI]

5.1.2 Function Documentation

5.1.2.1 int filter_t_init (int *n,* int *x,* int *y,* int *c,* double * *a,* double * *b,* filter_t_obj * *obj*)

An implementation of a Direct Form II digital filter, for processing a temporal series of 2D images along the temporal domain.

The filter is initiallized by specifying the number of filter taps, and the set of filter coefficients "a" (feedback) and "b" (feedforward).



Notes:

- at iniitalization, the feedback filter taps are normalized, so that a[0] == 1.0.
- the implementation is threaded, for improved processing speed on multicore machines.

Parameters

n	number of temporal filter taps
x	size of 1st dimension of input data, Nx
y	size of 2nd dimension of input data, Ny
c	size of 3rd dimension of input data, Nc
a	feedback coefficients
b	feedforward coefficients
obj	the output filter object that is initiated by the function call

5.1.2.2 int filter_t_quit (filter_t_obj * *obj*)

release all memory associated the temporal-domain filter bank object.

5.1.2.3 int filter_t_step (filter_t_obj * *obj,* COMPLEX * *datal,* COMPLEX * *dataO*)

cycle the temporal filter bank through one step.

Note: the filter bank object should be initialized using filter_t_init() prior to this call.

Parameters

obj	an FIL filter bank object.
dataI	input data
dataO	output data

5.1.2.4 void prep_unfold_dc (short *Nx,* short *N_acq,* short *Nf,* short *ncoils,* fil_fft_mngr * *mngr,* KCOORD * *kline_o,* KCOORD * *kline_e,* short *fr1,* COMPLEX * *data,* COMPLEX * *buf_lg,* COMPLEX * *buf_sm,* float * *filter_dc,* float * *filter_dckept,* short * *near_dc,* short *filt_w,* short *Nf_proc,* KCOORD * *kline_c,* COMPLEX * *data_dc*)

Prepare UNFOLD filter memory arrays

Parameters

Nx	Size in x direction.
N_acq	Number of acquired ky-kz lines.
Nf	Size time direction.
ncoils	number of coils
kline_o	Sampling function, odd time frames.
kline_e	Sampling function, even time frames.
fr1	Whether 1st frame is considered even (0) or odd (1).
buf_lg	Larger buffer, for processing.
buf_sm	Smaller buffer, for processing.
filter_dc	UNFOLD filter, DC region.
filter_dckept	UNFOLD filter (1-filter_dc)
near_dc	Frequencies around DC.
filt_w	Width of DC region, in near_dc.
Nf_proc	number of temporal frames, including synthetic

5.1.2.5 int set_nfproc (short *Nf,* short *flag*)

Find out how many frames should be used in the reconstruction. This must be an even number, and a smooth transition between 1st and last frame is desirable. To do so, temporary synthetic frames may have to be created. One has the option of a setting based on precision (larger Nf_proc, flag=0), or speed of recon- struction (smaller Nf_proc, flag=1).

Parameters

Nf	Number of time frames.
flag	switch between speed vs. precision

With `flag = 1`, the return value is `(int) 2.0*ceil( Nf/2 )`. This setting provides increased recon speed and decreased memory usage, i.e., smaller `Nf_proc` (e.g., for 5 frames 1-2-3-4-5, the reconstruction is done with 6 frames 1-2-3-4-5-4, where the 4th is repeated to make Nf_proc even).

With `flag = 0`, the return value is `2*Nf - 2` This setting provides increased precision, by avoiding discontinuities between the first and last frame of the data to be processed (e.g., for 5 frames 1-2-3-4-5, the recon is done with 8 frames 1-2-3-4-5-4-3-2, where frames 2, 3 and 4 are repeated, the transisition between 1st frame and last frame (frame #2) being presumably smooth).

5.1.2.6 void setup_unfold (SCAN_INFO * *hdr_ptr,* int *Nt,* float * *filter_dc,* float * *filter_ny,* float * *filter_dckept,* float * *filter_dcny,* short * *near_dc,* short * *filt_w_ptr*)

Set-up parameters/filters specific to the UNFOLD algorithm.

Parameters

hdr_ptr	Structure with scan info.
Nt	Number of time points.
filter_dc	UNFOLD filter, DC region.
filter_ny	UNFOLD filter, Ny region.
filter_dckept	UNFOLD filter (1-filter_dc)
filter_dcny	UNFOLD filter (filter_dc∗filter_ny)
near_dc	Frequencies around DC.
filt_w_ptr	Width of DC region, in near_dc.

5.1.2.7 void synth_frames (short *ncoils,* short *Nf,* short *Nz,* short *Ny,* short *Nx,* COMPLEX ∗ *data_in,* short *Nf_proc,* COMPLEX ∗ *data_out*)

Generate extra, synthetic time frames. This has two purposes: to make the number of phases even to insure alternation between the two sampling schemes, and to make the change between last and first frames smoother.

Parameters

ncoils	Number of receiver coils.
Nf	Size time direction.
Nz	Size in z direction.
Ny	Size in y direction.
Nx	Size in x direction.
data_in	Data input, to be extended from Nf to Nf_proc frames.
Nf_proc	New size along t, including synthetic frames.
data_out	output data

5.1.2.8 void synth_frames_cine (short *ncoils,* short *Nf_full,* short *fr1,* short *Nf,* short *N_acq,* short *Nx,* short *ky,* KCOORD ∗ *kline_o,* KCOORD ∗ *kline_e,* COMPLEX ∗ *data_in,* short *Nf_proc,* COMPLEX ∗ *data_out*)

For a given ky line, generate extra (synthetic) time frames. This has two purposes: to make the number of phases even to insure alternation between the two sampling schemes, and to make the change between last and first frames smoother.

Parameters

ncoils	Number of receiver coils.
Nf_full	Array size time direction.
Nf	Data size, <= Nf_full.
N_acq	Number of k lines per frame.
Nx	Size in x direction.
ky	ky line being treated.
data_in	Data input, to be extended from Nf to Nf_proc frames.
Nf_proc	New size along t, including synthetic frames.
data_out	output data

5.1.2.9 void transfer_dc (short *dir,* short *ncoils,* short *Nf,* short *Nz,* short *Ny,* short *Nx,* short *filt_w,* short ∗ *near_dc,* float ∗ *filter,* COMPLEX ∗ *lg_array,* COMPLEX ∗ *sm_array*)

When DC and Nyquist frequencies are handled seperately in an application of UNFOLD parad, this function can be used to copy the mult-frame DC data to a processing buffer.

Parameters

dir	Directiion of transfer (-1 from large array to small, 1 from small to large).
ncoils	Number of receiver coils.
Nf	Size time direction.
Nz	Size in z direction.
Ny	Size in y direction.
Nx	Size in x direction.
filt_w	Number of entries in near_dc[]
near_dc	Location of those frequencies that correspond to near-DC frequencies.
filter	Weight to give to data while transfering it (only for dir=1, from small to large).
lg_array	Array containing all temporal frequencies (length Nf along frequency axis).
sm_array	Array containing only temporal frequencies near DC (length filt_w along frequency axis).

5.1.2.10 void unfold_filter (short *ncoils*, short *Nf*, short *Nz*, short *Ny*, short *Nx*, float * *filter*, COMPLEX * *data*)

Apply the UNFOLD filters to the data

Parameters

ncoils	Number of receiver coils. (dim 5)
Nf	Size time direction. (dim 4)
Nz	Size in z direction. (dim 3)
Ny	Size in y direction. (dim 2)
Nx	Size in x direction. (dim 1)
filter	UNFOLD filter, applied along dim 4.
data	Data to filter (inplace operation: output overwrites input). size: Nx-Ny-Nz-Nf-ncoil

5.1.2.11 void unfold_recombine (short *Nx*, short *Ny_proc*, short *Nz_proc*, short *Nf_proc*, short * *near_dc*, short *filt_w*, COMPLEX * *dc*, COMPLEX * *full*)

Recombine the improved DC region in 'dc' (calculated with half the acceleration) to the rest of the bandwidth (calculated with full acceleration).

Parameters

Nx	Size in x direction.
Nf_proc	Size time direction, including synthetic frames
near_dc	Frequencies around DC.
filt_w	Width of DC region, in near_dc.
dc	Contains improved version of DC region.
full	Contains rest of bandwidth (DC removed). The output overwrites the input values in 'full'.

5.2 CINE processing functions

Functions

- void calc_DC_for_b1 (short Nx, short N_acq, short Ny, short Nf, short fr1, short ncoils, KCOORD ∗kline_c, int ∗acq_record, COMPLEX ∗xkykz, COMPLEX ∗DC_xkykz)
- void nonlin_cphase (SCAN_INFO hdr, int ∗hb_length, float ∗acq_timing, float systole_thr)
- void puzzleout_cineacq (int ∗trig_rr, int ∗trig_seg, int ∗trig_slice, SCAN_INFO ∗hdr_ptr, KCOORD ∗kline_o, KCOORD ∗kline_e, int ∗hb_length, float ∗acq_timing)
- void read_ecg_info (char ∗∗ecgfile, SCAN_INFO ∗hdr_ptr, int ∗trig_rr, int ∗trig_seg, int ∗trig_slice, int pass)
- void retrosp_gating_interp (short ncoils, short Nf_intrp, short Nf, short N_lines, short Ny, short Nx, short k, short ky, float ∗cph_record, float ∗ph_recon, COMPLEX ∗current_line, COMPLEX ∗kcine)
- void sort_rds_data (short ∗rawdata, int z, SCAN_INFO hdr, float ∗acq_timing, COMPLEX ∗kspace_data)

5.2.1 Detailed Description

5.2.2 Function Documentation

5.2.2.1 void calc_DC_for_b1 (short *Nx,* short *N_acq,* short *Ny,* short *Nf,* short *fr1,* short *ncoils,* KCOORD ∗ *kline_c,* int ∗ *acq_record,* COMPLEX ∗ *xkykz,* COMPLEX ∗ *DC_xkykz*)

Calculate the DC data to be used for self-calibration purposes.

Parameters

Nx	number of points along readout
N_acq	number of acquired phase encodes
Ny	resolution along phase encode dimension
Nf	number of temporal frames
fr1	select whether the data should be considered as an odd (fr1=0) or even (fr1=1) temporal frame.
ncoils	number of coils
kline_c	a list of the acquired k-space lines
xkykz	input data, size: Nx, N_acq, Nf, ncoils, Nz
DC_xkykz	output coil sensitivity estimate, size: Nx,2∗N_acq,1,ncoils

5.2.2.2 void nonlin_cphase (SCAN_INFO *hdr,* int ∗ *hb_length,* float ∗ *acq_timing,* float *systole_thr*)

Calculate cardiac phase, using a non-linear model whereby diastole data get stretched more than systole data, when accounting for arrythmia.

Parameters

hdr	Pointer to the structure with scan info.
hb_length	Length of each RR interval, in units of TR (output).
acq_timing	Info regarding every single TR in the entire acquisition (0. ky, 1. linear cardiac phase, 2. slice, 3. frame number, 3. order within segment, 5. heartbeat number, 6. index in sampling function, 7. non-linear cardiac phase).
systole_thr	Fraction of a (normal, or avera- ge) heartbeat considered to belong to systole.

5.2.2.3 void puzzleout_cineacq (int * *trig_rr,* int * *trig_seg,* int * *trig_slice,* SCAN_INFO * *hdr_ptr,* KCOORD * *kline_o,* KCOORD * *kline_e,* int * *hb_length,* float * *acq_timing*)

Figure out how the cine acqui- sition proceeded, based on the timing of the R waves, the sampling function, and scan parameters.

Parameters

trig_rr	Timing of R-waves, in units of TR.
trig_seg	Segment being acquired in given RR.
trig_slice	Slice being acquired in given RR.
hdr_ptr	Pointer to the structure with scan info.
kline_o	Sampling function, odd time frames.
kline_e	Sampling function, odd time frames.
hb_length	Length of each RR interval, in units of TR (output).
acq_timing	Info regarding every single TR in the entire acquisition (0. ky, 1. linear cardiac phase, 1. slice, 3. frame number, 2. order within segment, 5. heartbeat number, 6. index in sampling function, 7. non-linear cardiac phase).

5.2.2.4 void read_ecg_info (char ∗∗ *ecgfile,* SCAN_INFO ∗ *hdr_ptr,* int ∗ *trig_rr,* int ∗ *trig_seg,* int ∗ *trig_slice,* int *pass*)

First pass at reading the file, with ECG info, generated by the psd.

Parameters

ecgfile	Name of the ECG info file.
hdr_ptr	Pointer to the structure where the info is to be stored.
trig_rr	Timing of R-waves, in units of TR.
trig_seg	Segment being acquired in given RR.
trig_slice	Slice being acquired in given RR.
pass	For pass = 1, just find how many entries there are. For pass = 2, actually read these entries and store in memory.

5.2.2.5 void retrosp_gating_interp (short *ncoils,* short *Nf_intrp,* short *Nf,* short *N_lines,* short *Ny,* short *Nx,* short *k,* short *ky,* float ∗ *cph_record,* float ∗ *ph_recon,* COMPLEX ∗ *current_line,* COMPLEX ∗ *kcine*)

Sort the data gathered by the raw data server.

Parameters

ncoils	Number of receiver coils.
Nf_intrp	Size time direction, after.
Nf	Size time direction, before.
N_lines	Number of ky-kz lines.
Ny	Size y/ky direction.
Nx	Size in x/kx direction.
k	Index for current line.
ky	ky location for current line.
cph_record	Record of cardiac phase.
ph_recon	Phase values desired.
current_line	One k-line worth of data.
kcine	Array where to store result

5.2.2.6 void sort_rds_data (short ∗ *rawdata,* int *z,* SCAN_INFO *hdr,* float ∗ *acq_timing,* COMPLEX ∗ *kspace_data*)

Sort the data gathered by the raw data server.

Parameters

rawdata	Raw data from the raw data server, for all slices.
z	Slice to be sorted out.
hdr	Scan info.
acq_timing	Info regarding every single TR in the entire acquisition (0. ky, 1. linear cardiac phase, 2. slice, 2. frame number, 4. order within segment, 5. heartbeat number, 3. index in sampling function, 4. non-linear cardiac phase).
kspace_data	Sorted data, for desired slice.

5.3 Fourier Transform Functions

An FFT resource manager and associated functions.

Functions

- int fft1d (void ∗args)
- int fft_mngr_alloc_2dfft (fil_fft_mngr ∗mngr, int M, int N, char direction)
- int fft_mngr_alloc_fft (fil_fft_mngr ∗mngr, int N, char direction)
- int fft_mngr_init (fil_fft_mngr ∗mngr)
- int fft_mngr_quit (fil_fft_mngr ∗mngr)
- void fft_t (short ncoils, short nf, short nz, short ny, short nx, char direction, fil_fft_mngr ∗mngr, short center, COMPLEX ∗data_in, COMPLEX ∗data_out)
- void fftx (short ncoils, short Nf, int Nyz, short Nx, char dir, fil_fft_mngr ∗mngr, short center, COMPLEX ∗data)
- void ffty (short ncoils, short nf, short nz, short ny, short nx, char direction, fil_fft_mngr ∗mngr, short center, COMPLEX ∗data)
- void fftz (short ncoils, short nf, short nz, short ny, short nx, char direction, fil_fft_mngr ∗mngr, short center, COMPLEX ∗data_in, COMPLEX ∗data_out)
- int fil_2dfft (fil_fft_mngr ∗mngr, int M, int N, COMPLEX ∗data, char dir)
- int fil_fft (fil_fft_mngr ∗mngr, int N, COMPLEX ∗data, char dir, int unused)
- void fil_fft_shift (COMPLEX ∗data, int Nx, int Ny)

5.3.1 Detailed Description

An FFT resource manager and associated functions. This suite of functions provides a resource manager for Fast Fourier Transforms, in order to alleviate memory management bookkeeping.

The use of NC-IGT Fast Imaging Library (FIL) FFT manager facilitates reuse of scratch buffers needed by the FFT functions. Scratch buffers can be allocated at the start of a program, and used by multiple functions afterwards. Calling an FFT function with an initialized manager will create scratch space if there is none pre-allocated. Furthermore, if the FFT engine changes in the future, these `fil_fft` calls will remain backwards compatible.

Usage:

```
#include "fil_fft.h"

fil_fft_mngr mngr;
COMPLEX *data;

fft_mngr_init( &mngr );

data = (COMPLEX *) malloc( 256 * sizeof(COMPLEX) );

...

fil_fft( &mngr, 256, data, 'f', 1 );

...

fft_mngr_quit( &mngr );
```

Each manager allocates space for upto 512 different length-plus-f/b combinations. If your program needs more than 512 different FFTs, then use additional managers.

5.3.2 Function Documentation

5.3.2.1 int fft1d (void * *args*)

Computes a 1D FFT

To call this function, first declare a variable to hold the function arguments:

```
fft1d_args args;
```

Then, populate the variable,

```
args.direction = 'f';     // FFT direction: 'f' or 'b'
args.data   = in;         // pointer to COMPLEX data to be transformed
args.n      = N;          // length of data buffer
args.buf    = buf;        // pointer to temporary COMPLEX buffer
args.center = 0;          // k-space center is in the middle of the data
args.mngr   = mngr;       // a pointer to an FFT manager
```

Then call the function:

```
fft1d( (void *) &args );
```

This structure allows calls to `fft1d` to be multi-threaded.

Inverse FFTs are scaled by the length of the data vector, so that a 'round trip' of and FFT followed by an IFFT will leave the data at the same amplitude.

Parameters

args	a pointer to an `fft1d_args` variable

5.3.2.2 int fft_mngr_alloc_2dfft (fil_fft_mngr * *mngr,* int *M,* int *N,* char *direction*)

Allocate an manager object for 2D FFTs.

Parameters

mngr	an fil_fft manager
M	length of dim 1
N	length of dim 2
direction	'f':forward FFT :: 'b':inverse FFT

5.3.2.3 int fft_mngr_alloc_fft (fil_fft_mngr * *mngr,* int *N,* char *direction*)

Allocate an manager object for 1D FFTs.

Parameters

mngr	an fil_fft manager
N	length of requested fft
direction	'f':forward :: 'b':inverse

5.3.2.4 int fft_mngr_init (fil_fft_mngr * *mngr*)

Initialize the FFT manager.

The number of different FFT sizes needs to be declared before calling this initialization function.

Usage:

```
fil_fft_mngr mngr;

fft_mngr_init( &mngr );
```

Parameters

mngr	a pointer to a FFT manager

5.3.2.5 int fft_mngr_quit (fil_fft_mngr * *mngr*)

Free all FFT-related memory allocations, and release the manager.

Parameters

mngr	an FFT manager

5.3.2.6 void fft_t (short *ncoils,* short *nf,* short *nz,* short *ny,* short *nx,* char *direction,* fil_fft_mngr * *mngr,* short *center,* COMPLEX * *data_in,* COMPLEX * *data_out*)

FFT from time to frequency, or vice-versa. The input is a dataset ordered (nx -x- ny -x- nz -x- nf -x- ncoils), and the FFT is performed only in the temporal frequency direction (2nd dimension).

Parameters

ncoils	Number of coils. (dim 5)
nf	Number of time frames. (dim 4)
nz	Size along kz. (dim 3)
ny	Size along ky. (dim 2)
nx	Size along x or kx. (dim 1)
direction	'f' : forward FFT, or 'b' : inverse FFT
mngr	a pointer to an FFT manager
center	is center of k-space in the center of the data? set to 0 if DC is at 0, otherwise DC assumed to be at nf/2
data_in	Input data
data_out	Output data. If it is the same as data_in, input data simply gets over-written.

5.3.2.7 void fftx (short *ncoils,* short *Nf,* int *Nyz,* short *Nx,* char *dir,* fil_fft_mngr * *mngr,* short *center,* COMPLEX * *data*)

FFT along x or kx. The input is a data-set ordered (Nx -x- (Ny*Nz) -x- Nf -x- ncoils), and an FFT is performed only in the x/kx direction.

Parameters

ncoils	number of coils
Nf	size in the temporal frame direction.
Nyz	size in the Nyz direction.
Nx	size in the x/kx direction.
dir	'f' : forward FFT, or 'b' : inverse FFT
mngr	a pointer to an FFT manager
center	0 if DC is at (0,0), otherwise DC assumed to be at (Nx/2,Nyz/2)

data	Data to FFT. The input data gets overwritten by the output data.

5.3.2.8 void ffty (short *ncoils,* short *nf,* short *nz,* short *ny,* short *nx,* char *direction,* fil_fft_mngr * *mngr,* short *center,* COMPLEX * *data*)

FFT along y or ky. The input is a data-set ordered (nx -x- ny -x- nz -x- nf -x- ncoils), and an FFT is performed only in the y/ky direction.

Parameters

ncoils	Number of coils. (dim 5)
nf	Number of time frames. (dim 4)
nz	Size along kz. (dim 3)
ny	Size along ky. (dim 2)
nx	Size along x or kx. (dim 1)
direction	'f' : forward FFT, or 'b' : inverse FFT
mngr	a pointer to an FFT manager
center	0 if DC is at (0,0), otherwise DC assumed to be at (Nx/2,Ny/2)
data	Data to FFT. The input data gets overwritten by the output data.

5.3.2.9 void fftz (short *ncoils,* short *nf,* short *nz,* short *ny,* short *nx,* char *direction,* fil_fft_mngr * *mngr,* short *center,* COMPLEX * *data_in,* COMPLEX * *data_out*)

FFT along z or kz. The input is a data-set ordered as (nx -x- ny -x- nz -x- nf -x- ncoils), and an FFT is performed only in the z/kz direction.

Parameters

ncoils	Number of coils. (dim 5)
nf	Number of time frames. (dim 4)
nz	Size along kz. (dim 3)
ny	Size along ky. (dim 2)
nx	Size along x or kx. (dim 1)
direction	'f' : forward FFT, or 'b' : inverse FFT
mngr	a pointer to an FFT manager
center	0 if DC is at (0,0), otherwise DC assumed to be at (Nx/2,Ny/2)
data_in	Input data
data_out	Output data. If it is the same as data_in, input data simply gets over-written.

5.3.2.10 int fil_2dfft (fil_fft_mngr * *mngr,* int *M,* int *N,* COMPLEX * *data,* char *dir*)

Perform a 2D FFT on the `data`.

Note: to match Matlab's "ifft2", the output must be scaled by 1/(M*N).

Parameters

mngr	an FFT manager
M	dim-1 length of the data set
N	dim-2 length of the data set

data	a pointer to the data buffer to transform
dir	FFT direction forward "f" or backward "b"

5.3.2.11 int fil_fft (fil_fft_mngr * *mngr,* int *N,* COMPLEX * *data,* char *dir,* int *unused*)

Perform a 1D FFT on the `data`.

Note: consistent with the underlying FFT engines used in this function, the inverse FFT (when `dir = 'b';`) *is not* scaled to unity.

Parameters

mngr	an FFT manager
N	the length of the data set
data	a pointer to the data buffer to transform
dir	FFT direction forward 'f' or backward 'b'
unused	(unused)

5.3.2.12 void fil_fft_shift (COMPLEX * *data,* int *Nx,* int *Ny*)

Perform an FFT shift on the `data`, swapping the 1st/3rd and 2nd/4th quadrants of the data space.

Parameters

data	pointer to the input data
Nx	dim 1 of the data
Ny	dim 2 of the data

5.4 DICOM File Utilities

Functions

- void edit_dicom (LIST_NODE ∗list, short tag0, short tag1, char ∗filename, char ∗im_hdr, int ∗hdrsize_ptr, void ∗newvalue, short newvalue_length)
- void eval_dicom (LIST_NODE ∗list, short tag0, short tag1, char ∗filename, char ∗value, char ∗value_type, short max_length)
- LIST_NODE find_tag (LIST_NODE ∗list, short tag0, short tag1)
- void get_dicom_info (char ∗filename, LIST_NODE ∗list, short Nmax)
- int is_dicom (char ∗filename)
- void print_list (LIST_NODE ∗list, char ∗filename)
- short write_ima (SCAN_INFO hdr, short flip_h, short flip_v, short rot_90, char ∗∗imfile, int imnum, int z_anat, COMPLEX ∗buf, COMPLEX ∗result, char ∗path_out)

5.4.1 Detailed Description

A set of functions to read/edit/write DICOM headers and files.

The model followed by these functions is the following:

- The DICOM header information is read for a file
- The header is parsed to construct a dictionary (the 'list')
- For requests for tag values, the tag location is determined from the list, and the value is determined from the file itself.
- The edit tag function should be used primarily for changing the value of an existing tag, e.g. to change the image number.
- Writing a new DICOM file consists of writing the header information, followed by the image data itself.

Testing has been done with DICOM files generated by GE scanners.

5.4.2 Function Documentation

5.4.2.1 void edit_dicom (LIST_NODE ∗ *list,* short *tag0,* short *tag1,* char ∗ *filename,* char ∗ *im_hdr,* int ∗ *hdrsize_ptr,* void ∗ *newvalue,* short *newvalue_length*)

Change the value of one entry, in a DICOM header.

Parameters

list	List of DICOM entries.
tag0	Specific tag. first 16 bits of the tag,
tag1	the second set of 16 bits of the tag.
filename	Name of DICOM file, where im_hdr comes from.
im_hdr	DICOM header, stored in a char array.
hdrsize_ptr	The size of the DICOM header. If editing the header changes the size, the new size is returned.

newvalue	New value to be inserted in header.
newvalue_length	Length in bytes of newvalue. Unless this is a char string, this info tends to be redundant with newvalue_type.

5.4.2.2 void eval_dicom (LIST_NODE * *list,* short *tag0,* short *tag1,* char * *filename,* char * *value,* char * *value_type,* short *max_length*)

Find the value of one particular DICOM entry.

Parameters

list	List of DICOM entries.
tag0	A Specific tag. tag0 contains the first 16 bits
tag1	tag1 contains the 2nd set of 16 bits
filename	Name of DICOM file.
value	Pointer to where the value should be written.
value_type	String giving the variable type expected (char, float, double, long. ulong, short, ushort)
max_length	Maximum length, in bytes, to be written out (i.e., size allocated at the address 'value'.

5.4.2.3 LIST_NODE find_tag (LIST_NODE * *list,* short *tag0,* short *tag1*)

In a list of DICOM entries, finds a given tag, and return the associated entry. The returned entry contains the info on where (offset), how (VR) and how much (length) should be read, to get the value associated with the tag.

Returns

DICOM entry relating to this tag, extracted from the list of all entries.

Parameters

list	List of DICOM entries.
tag0	Specific tag. tag0 contains the first 16 bits,
tag1	tag1 the second set of 16 bits.

5.4.2.4 void get_dicom_info (char * *filename,* LIST_NODE * *list,* short *Nmax*)

Takes a DICOM file, and extracts all the tags. Each entry includes the tag number, the VR accociated with the tag, the length of the value field, and the offset in bytes from the beginning of the file to the value field.

Parameters

filename	Name of DICOM file.
list	List of tag entries extracted from file.
Nmax	maximum number of entries to be considered (proportional to the amount of memory allocated for the list).

5.4.2.5 int is_dicom (char * *filename*)

Returns 1 if the file is a DICOM file and 0 otherwise.

Parameters

filename	Name of DICOM file

5.4.2.6 void print_list (LIST_NODE * *list,* char * *filename*)

Print out a list of all DICOM tags in a file, along with VR values, offset values, the length of the value field and the actual value. The 'list' was first built using get_dicom_info().

Parameters

list	List of DICOM entries.
filename	Name of DICOM file.

5.4.2.7 short write_ima (SCAN_INFO *hdr,* short *flip_h,* short *flip_v,* short *rot_90,* char ** *imfile,* int *imnum,* int *z_anat,* COMPLEX * *buf,* COMPLEX * *result,* char * *path_out*)

Finish cropping, rotating, flipping, shifting the result, and write it out as a dicom file.

Parameters

hdr	header information, associated with *buf
flip_h	Flag to flip horizontally.
flip_v	Flag to flip vertically.
rot_90	Flag to rotate +90 (=+1) or -90 (=-1) degrees.
imfile	input DICOM file, for header information
imnum	Number of the slice being written out (between 1 and Nz).
z_anat	Current z loc, anatomical ordering
buf	input data
result	Reconstructed image data
path_out	Directory where to write result.

5.5 GradWarp Functions

Functions

- void gwarp_apply2 (float W[], float hpp[], float vpp[], float ∗hpixs, float ∗vpixs, int N_h, int N_v, float mag[])
- void gwarp_calc (float ∗tl, float ∗tr, float ∗br, float ∗bl, int N_h, int N_v, float hpixs[], float vpixs[], float ∗d, float ∗u, float W[], float hpp[], float vpp[], char ∗fname)
- void gwarp_prep (float tl[], float tr[], float br[], float bl[], int N_h, int N_v, float hpixs[], float vpixs[], float d[], float u[])
- int gwarp_read_parms (char ∗fname, GWARP_PARMS ∗gwp)

5.5.1 Detailed Description

5.5.2 Function Documentation

5.5.2.1 void gwarp_apply2 (float *W[],* float *hpp[],* float *vpp[],* float ∗ *hpixs,* float ∗ *vpixs,* int *N_h,* int *N_v,* float *mag[]*)

Parameters

W	the weighting factor
hpp	unwarped location in the horizontal direction
vpp	unwarped location in the vertical direction
hpixs	locations along the horizontal axis where pixels are found
vpixs	locations along the vertical axis where pixels are found
N_h	Number of pixels in horizontal direction
N_v	number of pixels in vertical direction
mag	image data

5.5.2.2 void gwarp_calc (float ∗ *tl,* float ∗ *tr,* float ∗ *br,* float ∗ *bl,* int *N_h,* int *N_v,* float *hpixs[],* float *vpixs[],* float ∗ *d,* float ∗ *u,* float *W[],* float *hpp[],* float *vpp[],* char ∗ *fname*)

outputs: W, hpp, vpp

Parameters

tl	(unused) top left coordinate (x,y,z), in cm
tr	(unused) top right coordinate (x,y,z), in cm
br	(unused) bottom right coordinate (x,y,z), in cm
bl	bottom left coordinate (x,y,z), in cm
N_h	Number of horizontal
N_v	number of vertical
hpixs	horizontal pixels (length = N_out)
vpixs	vertical pixels (length = N_out)
d	scale factors (length = 2)
u	unit vector directions (length = 6)
W	the weighting factor (length = N_h∗N_v)
hpp	unwarped location in the horizontal direction (length = N_h∗N_v)

vpp	unwarped location in the vertical direction (length = N_h∗N_v)
fname	input filename for the gradwarp spherical harmonic correction coefficients. Set to NULL if unavailable.

5.5.2.3 void gwarp_prep (float *tl[],* float *tr[],* float *br[],* float *bl[],* int *N_h,* int *N_v,* float *hpixs[],* float *vpixs[],* float *d[],* float *u[]*)

compute the 'd' and 'u' vectors, based on the size of the FOV and the number of pixels along each image dimension.

Parameters

tl	top left coordinate (x,y,z), in cm
tr	top right coordinate (x,y,z), in cm
br	bottom left coordinate (x,y,z), in cm
bl	bottom right coordinate (x,y,z), in cm
N_h	Number of h
N_v	Number of v
hpixs	locations along the horizontal axis where pixels are found
vpixs	locations along the vertical axis where pixels are found

5.5.2.4 int gwarp_read_parms (char ∗ *fname,* GWARP_PARMS ∗ *gwp*)

Function to parse the GE gradwarp spherical harmonic correction file.

Parameters

fname	input filename
gwp	the destination, structure of gradwarp parameters

5.6 Image Manipulation Functions

Routines to scale, interpolate, flip, and rotate images.

Functions

- void find_annotation (int slice, float loc0, char ras0, float slthick, float scanspacing, int locs_brth, int ∗rot3, float ∗ctr, float tloc0, float tloc1, float ∗loc_ptr, char ∗ras_ptr)
- void interplin_ima (short Nx, short Ny, short Ny_int, short N_out, float off_phdir, COMPLEX ∗data, float ∗data_int)
- void interpsinc_ima (short Nx, short Ny, short Ny_int, short N_out, float off_phdir, COMPLEX ∗data, float ∗data_int)
- void orientation (float x_tl, float y_tl, float z_tl, float x_tr, float y_tr, float z_tr, float x_br, float y_br, float z_br, int ∗rot1, int ∗rot2, int p_pos, int p_entry, short ∗flip_h_ptr, short ∗flip_v_ptr, short ∗rot_90_ptr)
- void rot_and_flip (short rot_90, short flip_h, short flip_v, short N_out, float ∗im_out)
- short rot_and_flip_data_obj (short rot_90, short flip_h, short flip_v, DATA_OBJ ∗obj)

5.6.1 Detailed Description

Routines to scale, interpolate, flip, and rotate images.

5.6.2 Function Documentation

5.6.2.1 void find_annotation (int *slice,* float *loc0,* char *ras0,* float *slthick,* float *scanspacing,* int *locs_brth,* int ∗ *rot3,* float ∗ *ctr,* float *tloc0,* float *tloc1,* float ∗ *loc_ptr,* char ∗ *ras_ptr*)

Figure out the location and RAS (1 char for right vs anterior vs superior, for sagital-like, coronal-like or axial-like planes respectively). This is used for image annotation, e.g., I5.2 would be an axial-like plane 5.2 mm from isocenter, in the inferior direction.

Parameters

slice	Number of the slice being treated (between 1 and N_sl).
loc0	Location of first slice.
ras0	Axis for value loc0.
slthick	Slice thickness (mm).
scanspacing	Spacing between slices (mm).
locs_brth	Number of slices acquired per breathold duration.
rot3	Third line of rotation matrix, i.e. through slice direction.
ctr	3 element position vector for FOV center.
tloc0	Distance from isocenter, as expressed in the epic code in rsp_info[0].rsptloc, for the first slice.
tloc1	Distance from isocenter, in rsp_info[0].rsptloc, for the 2nd slice.
loc_ptr	Location of current slice.
ras_ptr	Axis for value loc.

5.6.2.2 void interplin_ima (short *Nx,* short *Ny,* short *Ny_int,* short *N_out,* float *off_phdir,* COMPLEX ∗ *data,* float ∗ *data_int*)

Interpolate (linearly) the reconstruction image to the image size expected by GE's Signa display tools, and shift by prescribed amount in phase-encoding direction.

(Note: phase information is discarded)

Parameters

Nx	Size in x direction.
Ny	Size in y direction.
Ny_int	Interpolated size in y direction.
N_out	Dimension of square output matrix (a power of 2).
off_phdir	Offset with respect to isocenter in phase-encoding direction, as a fraction of the FOV.
data	Data to interpolate, Nx by Ny.
data_int	Interpolated data, N_out by N_out.

5.6.2.3 void interpsinc_ima (short *Nx*, short *Ny*, short *Ny_int*, short *N_out*, float *off_phdir*, COMPLEX * *data*, float * *data_int*)

Interpolate (using a sinc function) image to the image size expected by GE's Signa display tools, and shift by prescribed amount in phase-encoding direction.

(Note: phase information is discarded)

Parameters

Nx	Size in x direction.
Ny	Size in y direction.
Ny_int	Interpolated size in y direction.
N_out	Dimension of square output matrix (a power of 2).
off_phdir	Offset with respect to isocenter in phase-encoding direction, as a fraction of the FOV.
data	Data to interpolate, Nx by Ny.
data_int	Interpolated data, N_out by N_out.

5.6.2.4 void orientation (float *x_tl*, float *y_tl*, float *z_tl*, float *x_tr*, float *y_tr*, float *z_tr*, float *x_br*, float *y_br*, float *z_br*, int * *rot1*, int * *rot2*, int *p_pos*, int *p_entry*, short * *flip_h_ptr*, short * *flip_v_ptr*, short * *rot_90_ptr*)

Determine if the output needs to be rotated and/or flipped.

Parameters

x_tl	Coordinates for top left corner
x_tr	Coordinates for top right corner
x_br	Coordinates for bottom right corner
rot1	First line of rotation matrix, i.e. frequency-encoding direction.
rot2	Second line pf rotation matrix, i.e. phase-encoding direction.
p_pos	Patient position (1 for supine, 2 for prone, 4 for left decub, 8 for right decub).
p_entry	Patient entry ('1' for head first, '2' for feet first).
flip_h_ptr	Flag to flip horizontally.
flip_v_ptr	Flag to flip vertically.
rot_90_ptr	Flag to rotate +90 (=+1) or -90 (=-1)

5.6.2.5 void rot_and_flip (short *rot_90*, short *flip_h*, short *flip_v*, short *N_out*, float * *im_out*)

Rotate and/or flip a square magnitude (i.e. not complex) image.

Parameters

rot_90	Flag to rotate +90 (=+1) or -90 (=-1) degrees.

flip_h	Flag to flip horizontally.
flip_v	Flag to flip vertically.
N_out	Dimension of square, output image.
im_out	Magnitude image to rotate and/or flip.

5.6.2.6 short rot_and_flip_data_obj (short *rot_90,* short *flip_h,* short *flip_v,* DATA_OBJ ∗ *obj*)

Rotate and/or flip any data object.

Parameters

rot_90	Flag to rotate +90 (=+1) or -90 (=-1) degrees.
flip_h	Flag to flip horizontally.
flip_v	Flag to flip vertically.
obj	image to rotate and/or flip. Image must be square, but can be any DATA_OBJ data type

5.7 Vendor Specific Functions: GE

Functions

- int apply_epi_phase_ge (COMPLEX *k, int xres, int yres, int ncoils, DATA_OBJ *refdat, int expidir)
- int apply_vrgf_ge (COMPLEX *k, int nz, int *sz, char *fname, COMPLEX **ko)
- int correct_epi_phase_ge (COMPLEX *k, int xres, int yres, int ncoils, char *fname, int expidir)
- void correct_fcard (SCAN_INFO hdr, COMPLEX *kspace_data)
- int read_data_from_Pfile (SCAN_INFO *hdr, DATA_OBJ *rawframe, DATA_OBJ *vrgf, FILE *fid, char *buf, char *buf2, int slice, int vol, int nbaselines)
- short read_data_ge (char **Pnames, int z, SCAN_INFO hdr, KCOORD *kline, KCOORD *kline_o, KCOORD *kline_e, COMPLEX *kspace_data)
- short read_hdr_ge (char *Pname, SCAN_INFO *hdr_ptr)
- int read_vrgf_ge (char *fname, int Nx, char endian, DATA_OBJ *H)

5.7.1 Detailed Description

Functions specific to GE MR scanners, mostly for reading files generated by GE scanners.

5.7.2 Function Documentation

5.7.2.1 int apply_epi_phase_ge (COMPLEX * *k,* int *xres,* int *yres,* int *ncoils,* DATA_OBJ * *refdat,* int *expidir*)

Applies EPI phase correction to the k-space data. The input data is assumed to be sized [ky kx c], where c can be coils, slices, etc. A linear and constant phase correction term should be provided in the 'refdat' object.

Parameters

k	input data to correct, of size [yres xres ncoils]
xres	number of points along readout
yres	number of points along phase-encode direction
ncoils	number of frames/coils
refdat	a data object derived from the ref.dat file
expidir	the direction of the applied shift. Choices are limited to +1 or -1.

5.7.2.2 int apply_vrgf_ge (COMPLEX * *k,* int *nz,* int * *sz,* char * *fname,* COMPLEX ** *ko*)

Applies the VRGF correction matrix to EPI k-space data, to compensate for ramp-sampling during acquisition.

The VRGF data matrix is applied only to the first two dimensions of the input k-space data, i.e. the same matrix is applied to all coils and/or time frames.

DEPRECIATED: *this function is to be replaced by read_vrgf_ge and correct_rampsamp*

Parameters

k	input data to correct (output over-writes the input)
nz	number of dimension to the data
sz	an array holding the size of each data dimension. Holds the size of the output data matrix when the function successfully returns.

fname	the filename holding the VRGF correction data (default: "vrgf.dat")
ko	output data

5.7.2.3 int correct_epi_phase_ge (COMPLEX * *k,* int *xres,* int *yres,* int *ncoils,* char * *fname,* int *expidir*)

Applies EPI phase correction to the k-space data. The input data is assumed to be sized [ky kx c], where c can be coils, slices, etc. A linear and constant phase correction term is read from the supplied ref.dat file, and applied to both odd and even lines to align them.

Parameters

k	input data to correct (output over-writes the input)
xres	number of points along readout
yres	number of points along phase-encode direction
ncoils	number of frames/coils
fname	the filename holding the EPI phase correction data (default: "ref.dat")
expidir	the direction of the applied shift. Choices are limited to +1 or -1.

5.7.2.4 void correct_fcard (SCAN_INFO *hdr,* COMPLEX * *kspace_data*)

Fastcard sequences skip the first view of each segment, for the first cardiac phase. The reason has to do with the time it takes to detect an R wave, and that it would alread by too late to acquire the first view of the seg- ment. Fill-in the missing data using closest-neighbours.

Parameters

hdr	Structure with scan info.
kspace_data	Input k-space data.

5.7.2.5 int read_data_from_Pfile (SCAN_INFO * *hdr,* DATA_OBJ * *rawframe,* DATA_OBJ * *vrgf,* FILE * *fid,* char * *buf,* char * *buf2,* int *slice,* int *vol,* int *nbaselines*)

A function to read raw data from a GE P-file. Will return data corresponding to a specific slice or volume, if requested.

Parameters

hdr	the scan information associated with the data
rawframe	the output data buffer to fill
vrgf	the rampsamp correction data. set to NULL if not available or needed
fid	FILE handle to read from
buf	a pre-allocated scratch buffer of size: 2 * hdr.Nsl * hdr.ncoils * hdr.frame_size * (hdr.Ny+nbaselines) * sizeof(short)
buf2	a pre-allocated (COMPLEX *) scratch buffer of size: hdr.Nsl * hdr.ncoils * hdr.frame_size * hdr.Ny * sizeof(DATATYPE) DATATYPE is declared by the 'type' field of rawframe. It can be "fltc" or "dblc" (the default)
slice	the slice number to read. set to -1 to read all slices.
vol	the volume number (time point) to read

5.7.2.6 short read_data_ge (char ** *Pnames,* int *z,* SCAN_INFO *hdr,* KCOORD * *kline,* KCOORD * *kline_o,* KCOORD * *kline_e,* COMPLEX * *kspace_data*)

Read the input k-space data from GE P-files, for cardiac applications with temporal encoding

Parameters

Pnames	an array of filename
hdr	Structure with scan info.
kspace_data	the returned k-space data

5.7.2.7 short read_hdr_ge (char * *Pname*, SCAN_INFO * *hdr_ptr*)

Read the image scan header from a GE P file to find relevant imaging information.

Parameters

Pname	Name of the P file containing the header to be read.
hdr_ptr	Pointer to the structure where the info is to be stored.

5.7.2.8 int read_vrgf_ge (char * *fname*, int *Nx*, char *endian*, DATA_OBJ * *H*)

read in the GE ramp-sampling correction file. Use correct_rampsamp to apply to the measured data.

Parameters

fname	the filename holding the VRGF correction data (default: "vrgf.dat" if passed a NULL)
Nx	length of readout data
endian	endian-ness of the vrgf.dat file {'l','b'} [little-endian by default]
H	the data object for the ramp-sampling correction operator. Memory will be allocated if the H.data memory is NULL.

5.8 Vendor Specific Functions: Siemens

Functions

- int parse_vrgf_siemens (char ∗fname, DATA_OBJ ∗H)
- int read_data_from_measdat (SCAN_INFO ∗hdr, DATA_OBJ ∗rawframe, DATA_OBJ ∗vrgf, FILE ∗fid, char ∗buf, char ∗buf2, int slice, int vol, int ngc_flag)

5.8.1 Detailed Description

Functions specific to Siemens MR scanners, mostly for reading data files generated by Siemens scanners.

5.8.2 Function Documentation

5.8.2.1 int parse_vrgf_siemens (char ∗ *fname,* DATA_OBJ ∗ *H*)

parse the meas∗.dat file, and compute the ramp-sampling regridding operator

Parameters

fname	the filename for the EPI data (or data descriptor file)
H	the data object for the ramp-sampling correction operator. Memory will be allocated if the H.data memory is NULL.

5.8.2.2 int read_data_from_measdat (SCAN_INFO ∗ *hdr,* DATA_OBJ ∗ *rawframe,* DATA_OBJ ∗ *vrgf,* FILE ∗ *fid,* char ∗ *buf,* char ∗ *buf2,* int *slice,* int *vol,* int *ngc_flag*)

A function to read raw data from a Siemens meas_∗.dat file. Will return data corresponding to a specific slice or volume, if requested. && (chn_num == (hdr->ncoils-1))) {

&& (chn_num == (hdr->ncoils-1))) {

Parameters

hdr	the scan information associated with the data
rawframe	the output data buffer to fill
vrgf	the rampsamp correction data. set to NULL if not available or needed
fid	FILE handle to read from
buf	a pre-allocated scratch buffer of size: 2 ∗ hdr.Nsl ∗ hdr.ncoils ∗ hdr.frame_size ∗ (hdr.Ny+nbaselines) ∗ sizeof(float)
buf2	a pre-allocated (COMPLEX ∗) scratch buffer of size: hdr.Nsl ∗ hdr.ncoils ∗ hdr.frame_size ∗ hdr.Ny ∗ sizeof(DATATYPE) DATATYPE is declared by the 'type' field of rawframe. It can be "fltc" or "dblc" (the default)
slice	the slice number to read. set to -1 to read all slices.
vol	the volume number (time point) to read. (multiple calls to this function are needed to read all volumes)
ngc_flag	for EPI data, set to 1 in order apply internal ghost correct using pre-echo-train data. (default: 0==off) ∗

5.9 Linear System Solvers

standard numerical solutions to linear system problems

Functions

- double ∗ alloc1double (size_t n1)
- double ∗∗ alloc2double (size_t n1, size_t n2)
- void cgsolv (COMPLEX ∗A, int n, COMPLEX ∗b, COMPLEX ∗x, int k_max, double thresh)
- void compute_pinv (double ∗∗u, double w[], double ∗∗v, short n, short m, double ∗∗S_inv)
- int compute_svd (double ∗∗a, int m, int n, double w[], double ∗∗v)
- void eig_jacobi (double ∗∗a, double d[], double ∗∗v, int n)
- void eig_sort (double d[], double ∗∗v, int n)
- void free1double (double ∗p)
- void free2double (double ∗∗p)
- int lsv (COMPLEX ∗A, int m, int n, COMPLEX ∗x0, double ∗sigma, COMPLEX ∗v, float thresh)
- int rank_one_est (COMPLEX ∗C, float ∗mask, int Nx, int Ny, double ∗d, COMPLEX ∗u, COMPLEX ∗v, int tid)
- void svd_backsubstitute (double ∗∗u, double w[], double ∗∗v, int n, int m, double b[], double x[])
- void svd_sort (double ∗∗u, double ∗w, double ∗∗v, int m, int n)

5.9.1 Detailed Description

standard numerical solutions to linear system problems A linear of system of equations can be ordered as

$$Ax = b$$

where *A* is a system matrix of size m-by-n, *b* is a vector of measured data, and *x* is the solution vector—which after solution describes a linear combination of values from *A* to form *b*.

5.9.2 Function Documentation

5.9.2.1 double∗ alloc1double (size_t *n1*)

1D vector memory allocation for SVD routines

5.9.2.2 double∗∗ alloc2double (size_t *n1,* size_t *n2*)

2D array allocation, e.g. A[][], for SVD routines.

5.9.2.3 void cgsolv (COMPLEX ∗ *A,* int *n,* COMPLEX ∗ *b,* COMPLEX ∗ *x,* int *k_max,* double *thresh*)

Generic CG solver

Solves the problem *Ax = b* using the method of conjugate gradients. *A* must be conjugate symmetric, which implies that *x* and *b* are vectors of equal length, *n*.

Design based on the practical CG algorithm in “Matrix Computations” by Golub and Van Loan.

One can specify either a fixed number of iterations, or a residual error stopping criterion. To use the default values, (k_max = 20, thresh = 1e-9), declare ’0’ as the input values for k_max and thresh.

Parameters

A	system matrix, must be complex-conjugate symmetric
n	length of data vector
b	objective vector
x	solution
k_max	maximum number of iterations
thresh	residual error threshold

5.9.2.4 void compute_pinv (double ** *u*, double *w[]*, double ** *v*, short *n*, short *m*, double ** *S_inv*)

Computes the inverse of the sensitivity matrix, using the output of compute_svd(). It is given by v times the singular values w and the traspose of u. This function is simmilar to svd_backsubstitute(), but does not multiply by b. It returns the inverse matrix instead of a solution given by the inverse times b.

Parameters

u	left singular vectors
w	array of singular values
v	right singular vectors, transposed
n	number of rows in v
m	number of rows in u
S_inv	Pseudo-inverse of A

5.9.2.5 int compute_svd (double ** *a*, int *m*, int *n*, double *w[]*, double ** *v*)

Computes the singular value decomposition (SVD) of a matrix, $A = U\Sigma V^H$, where U and V are matrices with orthogonal columns, and Σ is a diagonal matrix holding the singular values, σ_i, associated with the singular vectors.

The most common use for an SVD is to compute a psuedo-inverse of the matrix A, in order to solve the linear system $Ax = b$. Once the SVD has been computed, the solution can be formed via

$$x = V\Sigma^{-1}U^T b$$

This can be perfomed computationally using the function svd_backsubstitute().

In the case of small singular values, e.g. $0 < \sigma_i < 1$, computing the inverse of Σ will induce excessively large contributions from the associated singular vectors. Thus, it is common to specify a threshold—discarding singular values and associated vectors below the threshold to limit such effects. To peform this in C programs, one needs to set all singular values below the threshold to zero.

Note, this function accepts only real-valued input. To find the SVD of a complex-valued matrix, one can split the real and imaginary components as

$$Ax = (\mathrm{re}\{A\} + j\mathrm{im}\{A\})x = b$$

$$\begin{bmatrix} \mathrm{re}\{A\} & -\mathrm{im}\{A\} \\ \mathrm{im}\{A\} & \mathrm{re}\{A\} \end{bmatrix} \begin{bmatrix} \mathrm{re}\{x\} \\ \mathrm{im}\{x\} \end{bmatrix} = \begin{bmatrix} \mathrm{re}\{b\} \\ \mathrm{im}\{b\} \end{bmatrix}$$

Usage:

```
double **A;
```

```
double **U;
double *W;
double **V;
double **A_inv;

A = alloc2double(m,n);
V = alloc2double(n,n);
W = alloc1double(n);
A_inv = alloc2double(n,m);

compute_svd( A, m, n, W, V );
U = A;
compute_pinv( U, W, V, n, m, A_inv);

free2double(A);
free2double(V);
free1double(W);
free2double(A_inv);
```

Credits: from CWP/SU. Similar to code in Netlib's EISPACK and CLAPACK. See also discussions in NR in C

Parameters

a	input matrix to decompose, will be overwritten by the left signular vectors
m	number of rows in a
n	number of columns in a
w	an array of output singular values
v	a matrix output of the right signular vectors, transposed

5.9.2.6 void eig_jacobi (double ** *a*, double *d[]*, double ** *v*, int *n*)

Find eigenvalues and corresponding eigenvectors via the Jacobi algorithm for symmetric matrices

Credits: from CWP/SU.

Parameters

a	a real-valued symmetric input matrix to decompose
d	an output array of eigen values
v	an output array of eigen vectors
n	the size (rows,cols) of the input matrix

5.9.2.7 void eig_sort (double *d[]*, double ** *v*, int *n*)

sort eigenvalues and corresponding eigenvectors in descending order

Credits: derived from CWP/SU.

Parameters

d	array of eigen-values

v	array of eigen-vectors
n	size of eigen system

5.9.2.8 void free1double (double * p)

free memory allocated by alloc1double().

5.9.2.9 void free2double (double ** p)

free memory allocated by alloc2double().

5.9.2.10 int lsv (COMPLEX * A, int m, int n, COMPLEX * x0, double * sigma, COMPLEX * v, float thresh)

Estimate of the dominant Left Singular Vector

The function employs an iterative algorithm that solves

$$\max_x ||A^H x||$$

to estimate the left singular vector associated with the largest singular value. (from this vector, the singular value and associated right singular vector can also be determined.)

If A is tall-and-thin, i.e. the number of rows, m, is much larger than the number of columns, n, then this algorithm is significantly faster than computing the SVD directly.

The function is most useful when only a rank-one estimate of the matrix is desired.

Parameters

A	the input matrix.
m	number of rows in input matrix.
n	number of columns in input matrix. Note, it is assumed that m>>n
x0	the output vector. size: m x 1 (needs to be pre-allocated)
thresh	[opt] convergent threshold (default=1e-4)

5.9.2.11 int rank_one_est (COMPLEX * C, float * mask, int Nx, int Ny, double * d, COMPLEX * u, COMPLEX * v, int tid)

uses Lanczos algorithm to determine singular vectors associated with the largest signular value.

Parameters

C	input data
Nx	number of rows in input data
Ny	number of columns in input data
d	the largest signular value. (Set to NULL if undesired)
u	the right singular vector associated with the largest singular value. (Set to NULL if undesired)
v	the left signular vector associated with the largest singular value. (Set to NULL if undesired)
tid	thread id for the Lanczos algorithm code. Set to zero is not using threads.

5.9.2.12 void svd_backsubstitute (double ** u, double w[], double ** v, int n, int m, double b[], double x[])

Perform back substitution of the singular value decomposittion, operation on the objective, b, to compute the solution, x, of the linear system of equations.

Back-substition can be performed much more quickly that explicitly computing the pseudo-inverse. However, if the inverse will be used to operate on multiple data sets (as in UNFOLD-SENSE), it is preferable to compute the inverse once and then apply it to all time frames in the series.

Credits: from CWP/SU. Similar to code in Netlib's EISPACK and CLAPACK. See also discussions in Numerical Recipes in C.

Parameters

u	Left singular vectors
w	Array of singular values
v	right signular vectors
n	number of rows in u
m	number of columns in v
b	objective
x	output

5.9.2.13 void svd_sort (double ** u, double * w, double ** v, int m, int n)

Sort the singular values and corresponding singular vectors in descending order

Assumes the input of the singular value decomposition of a matrix a[][] of the form: a[][] = u[][] w[] v^t [][]

Credits: Nils Maercklin, GeoForschungsZentrum (GFZ) Potsdam, Germany, 2001.

Parameters

u	left singular vectors
w	array of singular values
v	right singular vectors, transposed

5.10 Parallel Imaging Algorithms

Image reconstruction methods for subsampled data acquired using multiple receiver coils.

Functions

- int apply_vbc_coefs (DATA_OBJ *input, DATA_OBJ *output, DATA_OBJ *vbc_obj)
- int compute_root_sum_of_squares (DATA_OBJ *input, DATA_OBJ *output, double *max)
- int compute_virtual_body_coil (DATA_OBJ *input, DATA_OBJ *output, DATA_OBJ *vbc_obj)
- int export_grappa_coef (GRAPPA_PARMS **gparm, DATA_OBJ *gexport, int ncoil, int nslc)
- void generate_b1_filter (int fill_fac, short Ny, short Nz_proc, float *samp_dens, float *filt_b1)
- int grappa (COMPLEX *I, int *PhaseLines, dataheader *hdr, COMPLEX *O, int *Nlines, int **OutPhaseLines, int YKS, int XKS, float radius, int ndk, int *dks_in)
- int grappa_calc_parms (dataheader *hdr, COMPLEX *O, int YKS, int *indx0, int XKS, int *indx3, int nacs, int *acsl, int kxnum, COMPLEX **gparm, int fullsqr, int direction)
- int grappa_calc_recon_coef (COMPLEX *O, int *PhaseLines, dataheader *hdr, GRAPPA_PARMS *gparm, float radius)
- int grappa_for_cine (short Nx, short N_acq, short Ny, int Ny_proc, short Nz_proc, short Nf, short ncoils, short fill_fac, short fr1, int first_line, float pF, KCOORD *kline_c, float *samp_dens, float *weight_pF, COMPLEX *buf2, COMPLEX *b1, COMPLEX *data, COMPLEX *full)
- int grappa_recon (dataheader *hdr, COMPLEX *O, int YKS, int *indx0, int XKS, int *indx3, int *bins, COMPLEX *gparm, float radius)
- int grog_generate_eigd (GRAPPA_PARMS **gparm, int ncoil, int nslc)
- int grog_grid_2Ddata (DATA_OBJ *smpl_x, DATA_OBJ *smpl_y, DATA_OBJ *kdata, DATA_OBJ *src, DATA_OBJ *new_x, DATA_OBJ *new_y, DATA_OBJ *kdata_out, GRAPPA_PARMS *gparms)
- int import_grappa_coef (GRAPPA_PARMS ***gparm_out, DATA_OBJ *gimport)
- int inv_sens_matrix (short ncoils, short npts, float thresh, double **a, double *w, double **v, double *wgt_roe, double *wgt, double **S, double **S_buf, double **S_inv, double **S_inv_buf)
- int rlsgrappa (COMPLEX *I, int *PhaseLines, dataheader *hdr, COMPLEX *O, int *Nlines, int **OutPhaseLines, int YKS, float radius, int ndk, int *dks_in, float lambda)
- void self_ref_b1_basic (COMPLEX *kspace_data, float b1snr_thr, short Ny, short Nx, short Nz, short ncoils, short Nf, fil_fft_mngr *mngr, COMPLEX *b1)
- void self_ref_b1_for_cine (COMPLEX *kspace_data, float sigmak, float b1snr_thr, short N_acq, short Nx, short Ny, short Ny_proc, short Nz_proc, short ncoils, short Nf, short fr1, short fill_fac, short first_line, float *filt_b1, KCOORD *kline_o, KCOORD *kline_e, float *samp_dens, fil_fft_mngr *mngr, COMPLEX *bufsp, COMPLEX *b1)
- int self_ref_b1_via_geyser (COMPLEX *datain, dataheader *hdr, int *PhaseLines, COMPLEX *ptrW, fil_fft_mngr *mngr, int m)
- int spacerip_for_cine (short Nx, short N_o, short N_e, short N_c, short Ny, int Ny_proc, short Nz_proc, short Nf, short ncoils, float pF, KCOORD *kline_o, KCOORD *kline_e, KCOORD *kline_c, COMPLEX *b1, COMPLEX *buf, COMPLEX *data, COMPLEX *spacerip, short recon_flag)
- int sprip3d_conv_pts2y (int nph, int *phz_y, int *phz_z, int M, int N, int *Y)
- void sprip3d_lsqr_init (int nph, int M, int N, int L)
- void sprip3d_lsqr_post ()
- void sprip3d_lsqrsolv (int *Y, int nph, int m, int n, int L, COMPLEX *W, COMPLEX *Sk, COMPLEX **I, float *inlams, int lamsize, int maxits, double *xnormout, double *Rout, int tid)
- void sprip_cgsolv (int *Y, int nph, int n, int L, COMPLEX *Wx0, COMPLEX *u, COMPLEX *x, COMPLEX lambda, int k_max, int tid)
- void sprip_cgsr_init (int nph, int N, int L, int threadnum)
- void sprip_cgsr_post ()
- void sprip_cgsr_recon (float *phz_encode_list, int P, int N, int M, int L, COMPLEX *W, COMPLEX *Sk, double tau, int maxits, COMPLEX *rho, int tmax)

- void sprip_lsqr_init (int nph, int N, int L, int threadnum, int lamsize, int maxits)
- void sprip_lsqr_post ()
- void sprip_lsqr_recon (float *phz_encode_list, int P, int N, int M, int L, COMPLEX *W, COMPLEX *Sk, float *lams, int numlams, COMPLEX *D, int maxits, COMPLEX **rho, double *xout, double *rout, int tmax, float *usedlams)
- void sprip_lsqrsolv (int *Y, int nph, int N, int L, COMPLEX *Wx0, COMPLEX *Sk, COMPLEX **I, float *lams, int lamsize, COMPLEX *D, int maxits, double *xnormout, double *Rout, int tid, float *lout)
- void vdsense (short Nx, short N_acq, short Ny, short Ny_proc, short Nz_proc, short Nf, short ncoils, short fill_fac, short fr1, short first_line, float pF, fil_fft_mngr *mngr, KCOORD *kline_o, KCOORD *kline_e, float *samp_dens, float *weight_pF, COMPLEX *buf, DATATYPE *Uarray, COMPLEX *data, COMPLEX *vdsen_ima)
- void vdsense_apply (short Nx, short Ny, short Nz, short Nf, short ncoils, COMPLEX *sen_before, DATATYPE *Uarray, COMPLEX *sen_after)
- void vdsense_prep (short Nx, short Ny, short Nz, short ncoils, float thresh, int accel, XCOORD *alias, COMPLEX *b1, DATATYPE *Uarray)

5.10.1 Detailed Description

Image reconstruction methods for subsampled data acquired using multiple receiver coils. Parallel imaging employs multiple receiver coils, which effectively apply a spatial-domain encoding in addition to the Fourier-domain encoding traditionally employed by MR imaging.

With multiple coils, the MR data acquisition can be modeled by the following signal equation [A]:

$$s_l(\mathbf{k}) = \int_V \rho(\mathbf{r}) W_l(\mathbf{r}) e^{-j\mathbf{k}\cdot\mathbf{r}} d\mathbf{r}$$

One approch to solve this inverse problem computationally is to first discretize the terms and then concatenate the equations for all coils to form a single linear system of equations.

$$\mathbf{s} = \mathbf{P}\rho$$

where $\mathbf{P}$ is described by an estimate of the coil sensitivity maps and the phase-encoding pattern employed. The library provides one approach to estimating coil sensitivity maps, using self-referenced data [D].

Two methods in the NC-IGT fast imaging library employ this approach, SENSE [A] and SPACE RIP[F]. The SENSE functions provide reconstruction of uniformly subsampled data, i.e. where the distance is k-space between each phase-encode line is the same. Variable density SENSE [B] can be employed in those cases where low-frequency and high-frequency k-space are each uniformly sub-sampled, but at different rates. E.g. 2x in the low-frequency range, and 4x in the high-frequency range.

For sampling patterns with a wider variety of k-space distances, that is *non-uniform* subsampling, one can employ SPACE RIP. Two iterative linear system solvers are provided for SPACE RIP reconstructions: CG [E] and LSQR-Hybrid [I]. At high-acceleration factors, it is often useful to employ regularization.

$$\begin{bmatrix} \mathbf{s} \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{P} \\ \lambda\mathbf{I} \end{bmatrix} \rho$$

The LSQR-Hybrid method can efficiently find all of the solutions corresponding to a number of regularization parameter values. The "best" image as measured by L-curve methods is then presented as the solution. For a single regularization parameter, CG is slightly faster than LSQR-Hybrid, so this method is provided as well.

An alternate approach to solving the inverse problem is to view the acquisition in k-space as a convolution between two spatial-domain functions. This is the approach taken by GRAPPA [G]. This method *does not* explicitly employ estimates of the coil sensitivity maps. Rather it maps various output data points together to estimate a convolution kernel combining data from all coils.

Modelling GRAPPA as a matrix operator [L], one can modify the target distance for reconstructed points at will using an Eigen decomposition of the operator [M]. Since Ver 2.0, the library includes functions to compute GRAPPA coefficients suitable for GROG, as well as functions to apply GROG in multi-line radial acquisitions as in PROPELLER [N].

The NC-IGT GRAPPA implementation can reconstruct data acquired using non-uniform sampling patterns, although the visual results typically contain more noise than a corresponding SPACE RIP reconstruction. GRAPPA is the method of choice, however, in cases where aliasing due to a reduced-field-of-view is present in addition to aliasing resulting from subsampling.

It is noted as well that combinations of GRAPPA, VD-SENSE, and SPACE RIP, can often produce better images than any single algorithm alone.

References

A. Pruessmann KP, Weiger M, Scheidegger MB, Boesiger P. SENSE: Sensitivity encoding for fast MRI. *Magn Reson Med* 1999; 42(5):952-62.

B. Madore B. Using UNFOLD to remove artifacts in parallel imaging and in partial-Fourier imaging. *Magn Reson Med* 2002; 48(3):493-501.

C. Kellman P, Epstein FH, McVeigh ER. Adaptive sensitivity encoding incorporating temporal filtering (TSENSE). *Magn Reson Med* 2001; 45(5):846-852.

D. McKenzie CA, Yeh EN, Ohliger MA, Price MD, Sodickson DK. Self-calibrating parallel imaging with automatic coil sensitivity extraction. *Magn Reson Med* 2002; 47(3):529-538.

E. Pruessmann KP, Weiger M, Bornert P, Boesiger P. Advances in sensitivity encoding with arbitrary k-space trajectories. *Magn Reson Med* 2001; 46(4):638-651.

F. Kyriakos WE, Panych LP, Kacher DF, Westin CF, Bao SM, Mulkern RV, Jolesz FA. Sensitivity profiles from an array of coils for encoding and reconstruction in parallel (SPACE RIP). *Magn Reson Med* 2000; 44(2):301-308.

G. Griswold MA, Jakob PM, Heidemann RM, Nittka M, Jellus V, Wang J, Kiefer B, Haase A. Generalized autocalibrating partially parallel acquisitions (GRAPPA). *Magn Reson Med* 2002; 47(6):1202-1210.

H. Hoge WS, Brooks DH, Madore B, Kyriakos WE. A tour of accelerated parallel MR imaging from a linear systems perspective. *Concepts in MR* 2005; 27A(1):17-37

I. Hoge WS, Kilmer ME, Haker SJ, Brooks DH, Kyriakos WE. Fast regularized reconstruction of non-uniformly subsampled parallel MRI data. in *Proc of 2006 IEEE Intl Symp on Biomedical Imaging (ISBI06)*. Arlington, VA, USA, 2006; 714-717.

J. Buehrer M, Boesiger P, and Kozerke S. "Virtual body coil calibration for phased-array imaging." 17th ISMRM Scientific Meeting, page 759, Honolulu, HI, 2009.

K. Buehrer M, Pruessmann KP, Boesiger P, and Kozerke S. Array compression for MRI with large coil arrays. Magn Reson Med, 57(6):1131-1139, June 2007.

L. Griswold MA, Blaimer M, Breuer F, Heidemann RM, Mueller M, and Jakob PM. Parallel magnetic resonance imaging using the GRAPPA operator formalism. Magn Reson Med, 54(6):1553-1556, Oct 2005

M. Seiberlich N, Breuer FA, Blaimer M, Barkauskas K, Jakob PM, and Griswold MA. "Non-Cartesian data reconstruction using GRAPPA operator gridding (GROG)." Magn Reson Med, 58(6):1257-1265, Dec 2007.

N. Pipe JG. "Motion correction with PROPELLER MRI: Application to head motion and free-breathing cardiac imaging". Magn Reson Med, 42(5):963-969, Nov 1999.

5.10.2 Function Documentation

5.10.2.1 int apply_vbc_coefs (DATA_OBJ * *input,* DATA_OBJ * *output,* DATA_OBJ * *vbc_obj*)

apply a set of coefficients calculated by compute_virtual_body_coil to a new set of input data.

If the output object is that same as the input, the input data will be overwritten, and the size adjusted accordingly

Parameters

input	the input coil data, sized [x-y-c] or [x-y-z-c]
vbc_obj	returns the coil combination coeficients, if non-NULL

5.10.2.2 int compute_root_sum_of_squares (DATA_OBJ * *input,* DATA_OBJ * *output,* double * *max*)

Computes the root-sum-of-squares image from the input data.

input data can be either 3d: x-y-c, or 4d: x-y-z-c

The scaling factor needed to normalize the image (set the max pixel value to '1') is returned in 'max'

Parameters

input	input data
output	output data (magnitude only, no phase)
max	image normalization scale factor (calculated, but not applied)

5.10.2.3 int compute_virtual_body_coil (DATA_OBJ * *input,* DATA_OBJ * *output,* DATA_OBJ * *vbc_obj*)

An implementation of [J], which builds on ideas in [K] for identifying alternate subspaces for reciever coil representations. This implementation normalizes each of the coils, and finds the dominant subspace support along the coil dimension of the data. This generates a *virtual body coil*, which can be used in parallel imaging to produce image reconstructions with more homogeneous illumination in a self-referenced way.

Usage:

With SENSE or SPACE RIP, one can renormalize the sensitivity estimates before the reconstruction step:

```
self_ref_b1_via_geyser( K, hdr, PhaseLines, W, mngr, 0 );
compute_virtual_body_coil( W, hdr.Nx, hdr.Ny, hdr.ncoils, Bv );
for (cnt=0;cnt<hdr.ncoils;cnt++) {
  for (cnt2=0;cnt2<hdr.Nx*hdr.Ny;cnt2++) {
     W[ cnt*hdr.Nx*hdr.Ny + cnt2 ] =
        complex_division( W[ cnt*hdr.Nx*hdr.Ny + cnt2 ],
                          Bv[ cnt2 ] );
  }
}
```

Alternatively, after reconstructing with GRAPPA, one can find the associated virtual body coil image via the projection across all coil images. See the `grappa.c` MEX file for an example.

Note that the virtual body coil image contains phase information, so it can be used in applications were a root-sum-of-squares body coil estimate is unsuitable.

Parameters

input	the input coil data, sized [x-y-c] or [x-y-z-c]
output	the output virtual body coil. If (output->data == NULL), memory will be allocated.
vbc_obj	returns the coil combination coeficients, if non-NULL

5.10.2.4 int export_grappa_coef (GRAPPA_PARMS ** *gparm,* DATA_OBJ * *gexport,* int *ncoil,* int *nslc*)

Converts an array of grappa parameters into a DATA_OBJ suitable for output to a .nd file, using write_nd_data2. Each location in the array is assumed to be for a specific slice of data, and the grappa kernel is presumed to be the same size (resulting in the same number of grappa coefficients) for each slice.

The returned data object will have the meta-data field populated, with a brief description of how the coefficients are

arranged. This meta-data can be decoded in matlab using load_gparms.m or in C using import_grappa_coef(), with the data properly organized so that previously generated coefficients can be correctly used in future applications.

Parameters

gparm	the input array of grappa parameters, one set for each slice
gexport	the output data object
ncoil	the number of coils associated with the grappa parameters
nslc	the number of slices in the input array

5.10.2.5 void generate_b1_filter (int *fill_fac,* short *Ny,* short *Nz_proc,* float * *samp_dens,* float * *filt_b1*)

Generate a 2D Gaussian filter to extract the ky-kz region, near k-space center, to be used for sensitivity mapping.

Parameters

samp_dens	Sampling density
filt_b1	Gaussian filter to select ky-kz region to be used for B1 mapping.

5.10.2.6 int grappa (COMPLEX * *I,* int * *PhaseLines,* dataheader * *hdr,* COMPLEX * *O,* int * *Nlines,* int ** *OutPhaseLines,* int *YKS,* int *XKS,* float *radius,* int *ndk,* int * *dks_in*)

An implementation of "Generalized autocalibrating partially parallel acquisitions," by M. A. Griswold, P. M. Jakob, et. al.. *Mag Reson Med*, 47(6):1202-1210. Jun 2002. [DOI]

Processing is done in k-space, coil-by-coil. The reconstructed data includes all coils. To form an image, one can use root-sum-of-squares.

Parameters

I	input data
PhaseLines	input phase index list
hdr	header associated with input data
O	buffer to hold output data. This should be Nvertical x xres x Ncoils, i.e. zero padded
Nlines	number of non-zero output lines
OutPhaseLines	list of output non-zero lines (will be malloced)
YKS	grappa-kernel size, y-dim (2 or 4)
XKS	grappa-kernel size, x-dim (must be odd)
radius	size of reconstruction radius along freq-encode dimension. In some cases (e.g. coil sensitivity estimation) the full read-out length is not needed
ndk	number of Delta k's to compute. If zero, then the dks list is computed automatically.
dks_in	the list of Delta k's to compute. Set to NULL if ndk = 0.

5.10.2.7 int grappa_calc_parms (dataheader * *hdr,* COMPLEX * *O,* int *YKS,* int * *indx0,* int *XKS,* int * *indx3,* int *nacs,* int * *acsl,* int *kxnum,* COMPLEX ** *gparm,* int *fullsqr,* int *direction*)

Calculate GRAPPA reconstruction parameters for the given data.

called by `grappa`

Example: for a 2x5 GRAPPA kernel, to reconstruct data sampled at 2x acceleration... `YKS = 2; indx0 = {-1,1}; XKS = 5; indx3 = {-2,-1,0,1,2};`

Parameters

hdr	header associated with input data
O	zero-padded k-space data
YKS	size of kernel along phase encode direction (length of indx0)
indx0	phase encode offsets for kernel
XKS	size of kernel along read-out direction (length of indx3)
indx3	sample offsets along readout
nacs	number of ACS lines
acsl	array of ACS line indices
kxnum	number of readout (k_x) points used for calibration
gparm	the output GRAPPA parameters. Pass a NULL pointer, memory for the parameters will be allocated and returned.
fullsqr	for GROG: set to 1 if full dx set is needed, 2 for both dx and dy data set. set to 0 otherwise
direction	for GROG: modifies the yoffset value by {0 [default],+1,-1}.

5.10.2.8 int grappa_calc_recon_coef (COMPLEX * *O*, int * *PhaseLines*, dataheader * *hdr*, GRAPPA_PARMS * *gparm*, float *radius*)

The primary function for calculating GRAPPA coefficients.

Input the source calibration data (ACS) and the phase lines where that data was acquired, and the parameters will be returned in 'gparm'.

Parameters

O	buffer holding zero-padded input data. This should be Nvertical x xres x Ncoils
PhaseLines	input list of phase index showing which lines of O have data.
hdr	header associated with input data
gparm	need to predeclare the kernel size (gparm->YKS and gparm->XKS) and gparm->parray (set to NULL). ndk and dks may also be preset, or set to ndk=0 and dks=NULL for values computed from PhaseLines. For standard GRAPPA parameters, set hdr->grog = 0; For GROG, use hdr->grog = 1 for a square parameter set, suitable for GROG along the phase encode direction. use hdr->grog = 2 for an additional set of parameters suitable for GROG along the readout direction.
radius	width, (0.0 .. 1.0], along the readout line to use in calibration

5.10.2.9 int grappa_for_cine (short *Nx*, short *N_acq*, short *Ny*, int *Ny_proc*, short *Nz_proc*, short *Nf*, short *ncoils*, short *fill_fac*, short *fr1*, int *first_line*, float *pF*, KCOORD * *kline_c*, float * *samp_dens*, float * *weight_pF*, COMPLEX * *buf2*, COMPLEX * *b1*, COMPLEX * *data*, COMPLEX * *full*)

A function to call GRAPPA from cine_unf

Parameters

Nx	Size of input dataset, along kx (readout)
N_acq	k locs acquired / frame (in ky-kz plane)
Ny	Size of input dataset, along ky (phase)
Ny_proc	Ny used for processing, in vdsense
Nz_proc	Nz used for processing
Nf	number of frames (either time or phase)

ncoils	Number of coils
fill_fac	fill factor, to accomodate non-int samp pts
fr1	whether 1st frame considered even (0) or odd (1)
first_line	First y line kept, when cropping recon FOV
pF	partial-Fourier (1=fully sampled, 0.5=50% sampled)
kline_c	array of acquired phase encode lines
b1	pointer to coil sensitivity map data
data	pointer to input data
full	pointer to reconstructed data

5.10.2.10 int grappa_recon (dataheader * *hdr,* COMPLEX * *O,* int *YKS,* int * *indx0,* int *XKS,* int * *indx3,* int * *bins,* COMPLEX * *gparm,* float *radius*)

reconstruct the missing data using GRAPPA.

called by `grappa`

Parameters

hdr	structure describing the data
O	the (zero-padded) input data
YKS	size of kernel along phase encode direction
indx0	phase encode offsets for kernel
XKS	size of kernel along read-out direction
indx3	sample offsets along readout
bins	an array dscribing the data along the phase encoding dimension. 0 : empty ; 1 : raw data ; 2 : ACS line
gparm	pointer to the GRAPPA reconstruction parameters
radius	size of reconstruction radius along freq-encode dimension

5.10.2.11 int grog_generate_eigd (GRAPPA_PARMS ** *gparm,* int *ncoil,* int *nslc*)

Derives the eigen-vectors and eigen-values needed by GROG from the input GRAPPA parameters.

These are stored in the .eig_d and .eig_v pointers of the GRAPPA_OBJ, over-writing whatever may have been there previously.

Parameters

gparm	input array of GRAPPA parameters, suitable for GROG (set gparm->grog = 1 or 2 prior to generating gparm coefficients)
ncoil	number of coils assocated with the input array
nslc	number of slices assocated with the input array

5.10.2.12 int grog_grid_2Ddata (DATA_OBJ * *smpl_x,* DATA_OBJ * *smpl_y,* DATA_OBJ * *kdata,* DATA_OBJ * *src,* DATA_OBJ * *new_x,* DATA_OBJ * *new_y,* DATA_OBJ * *kdata_out,* GRAPPA_PARMS * *gparms*)

A function to shift k-space data locations, using GROG.

the input data is a triplet array, [smpl_x, smpl_y, kdata] that is transformed to [new_x, new_y, kdata_out]. The first 5 elements are specified, while the 6th is computed.

Parameters

smpl_x	a 1D (double / "dblr") array of sample point locations, along the x (horizontal) axis
smpl_y	a 1D (double / "dblr") array of sample point locations, along the y (vertical) axis
kdata	a 1D (complex double / "dblc") array of sampled k-space data, at each {smpl_x,smpl_y} point in the previous two arrays
src	a 1D (short int / "intr") array of indices, indicating which kdata point should be used for each output target listed in {new_x,new_y}
new_x	a 1D (double / "dblr") array of destination grid locations, along x
new_y	a 1D (double / "dblr") array of destination grid locations, along y
kdata_out	a 1D (complex double / "dblc") array of computed k-space data, at the points specified by {new_x,new_y}
gparms	a pointer to the GROG parameters

5.10.2.13 int import_grappa_coef (GRAPPA_PARMS *** *gparm_out,* DATA_OBJ * *gimport*)

Converts a DATA_OBJECT, with an approriately defined and consistent metadata field, into an array of grappa parameters suitable for subsequent GRAPPA reconstruction calls. Each location in the array is assumed to be for a specific slice of data, and the grappa kernel is presumed to be the same size (resulting in the same number of grappa coefficients) for each slice.

Parameters

gparm_out	a pointer to an (empty) array of GRAPPA parameters. This function will replace the data pointer with newly allocated memory.
gimport	the input data object

5.10.2.14 int inv_sens_matrix (short *ncoils,* short *npts,* float *thresh,* double ** *a,* double * *w,* double ** *v,* double * *wgt_roe,* double * *wgt,* double ** *S,* double ** *S_buf,* double ** *S_inv,* double ** *S_inv_buf*)

Estimate the inverse of the sensitivity matrix, for each pixel.

The threshold paramter is described in Madore B. "UNFOLD-SENSE: A parallel MRI method with self-calibration and artifact suppression." *Magn Reson Med* 2004; 52(2):310–320.

At the expense of artifact content (presumed to be suppressed with UNFOLD later in the processing), a matrix with less noise (i.e. greater regularization) for the i^{th} entry will be obtained. If the npts vectors representing how each one of the overlapped points is seen by the coils were orthogonal, optimum SNR (i.e. the Roemer case) would be achieved. Since the solution obtained by matrix inversion would amplify noise beyond the threshold, fool the recon to believe these vectors are more orthogonal than they really are, so that it finds smaller weights. The vector for the i^{th} component, the one under consideration, is left unchanged, while the npts-1 others are made more orthogonal to it. Each one of these npts-1 vectors has a component parallel, and one orthogonal to the i^{th} (unit) vector uv[]. Vectors are made more orthogonal to uv[] by keeping only a fraction of the parallel component (and leaving the orthogonal one untouched). The goal of the iterative process is to find the right value that brings noise amplification down to a factor of thresh.

The implication is that the reconstructed image will have less noise than a standard psuedo-inverse, at the expense of greater artifact. It is assumed that these artifacts will then be eliminated using UNFOLD.

If UNFOLD is not being used, then one can use a very large value for thresh to suppress these iterations.

Parameters

ncoils	Number of receiver coils.

npts	Number of non-degenerate points (npts <= n_s, the acceleration seen by SENSE).
thresh	Noise threshold.
a	Matrix to invert for the SENSE recon of npts pixels. (of size 2∗ncoils x 2∗accel
w	Singular values, in the SENSE recon. (of size 2∗accel)
v	Required for backsubstitution, in the SENSE recon. (of size (2∗accel)
wgt_roe	Weights as in Roemer et al.
wgt	Weights with present method.
S	Sensitivity matrix, copy of "a".
S_buf	Modified version of S.
S_inv	Inverse of the sensitivity matrix. (2∗accel x 2∗ncoils)
S_inv_buf	Unfinished version of S_inv.

5.10.2.15 int rlsgrappa (COMPLEX ∗ *I,* int ∗ *PhaseLines,* dataheader ∗ *hdr,* COMPLEX ∗ *O,* int ∗ *Nlines,* int ∗∗ *OutPhaseLines,* int *YKS,* float *radius,* int *ndk,* int ∗ *dks_in,* float *lambda*)

An implementation of “RLS-GRAPPA: Reconstructing parallel MRI data with adaptive filters,” by W S Hoge, F Gallego, Z Xiao, and D H Brooks, to appear in *Proc 2008 IEEE Intl Symp on Biomedical Imaging*, May 2008.

Processing is done in hybrid-space, ky-x, coil-by-coil with GRAPPA reconstruction coefficients calculated simlutaneously with missing data reconstruction using an adapative RLS filter. Currently only ’RLS-along-x’ is supported.

Parameters

I	input data
PhaseLines	input phase index list
hdr	header associated with input data
O	buffer to hold output data. This should be Nvertical x xres x Ncoils, i.e. zero padded
Nlines	number of non-zero output lines
OutPhaseLines	list of output non-zero lines (will be malloced)
YKS	grappa-kernel size, y-dim (2 or 4)
radius	size of reconstruction radius along freq-encode dimension. In some cases (e.g. coil sensitivity estimation) the full read-out length is not needed
ndk	number of Delta k’s to compute. If zero, then the dks list is computed automatically.
dks_in	the list of Delta k’s to compute. Set to NULL if ndk = 0.
lambda	the rls forgetting factor

5.10.2.16 void self_ref_b1_basic (COMPLEX ∗ *kspace_data,* float *b1snr_thr,* short *Ny,* short *Nx,* short *Nz,* short *ncoils,* short *Nf,* fil_fft_mngr ∗ *mngr,* COMPLEX ∗ *b1*)

Calculate B1 sensitivity maps from raw kspace data.

This is a very basic function: it applies only a 1/2 sinusoidal filter only along ky; assumes kspace is fully sampled with the DC coordinate in the center of k-space. An fftshift is during processing to ensure that the estimated coil sensitivities have smooth phase.

Parameters

kspace_data	kspace data of size (ky, kx, z, coils, t).
b1snr_thr	SNR threshold for sensitivity maps.
Ny	Dimension along the y/ky direction.
Nx	Dimension along the x/kx direction.

Nz	Dimension along the z/kz direction.
ncoils	Number of coil elements in array.
Nf	Number of frames in kspace_data.
mngr	FFT manager
b1	B1 sensitivity maps. Size: nx-ny-nz-ncoils

5.10.2.17 void self_ref_b1_for_cine (COMPLEX * *kspace_data,* float *sigmak,* float *b1snr_thr,* short *N_acq,* short *Nx,* short *Ny,* short *Ny_proc,* short *Nz_proc,* short *ncoils,* short *Nf,* short *fr1,* short *fill_fac,* short *first_line,* float * *filt_b1,* KCOORD * *kline_o,* KCOORD * *kline_e,* float * *samp_dens,* fil_fft_mngr * *mngr,* COMPLEX * *bufsp,* COMPLEX * *b1*)

Calculate B1 sensitivity maps from the dynamic data itself (i.e. self- referenced). It uses a combination of the strategies as part of GRAPPA (k- space center sampled more densely) and TSENSE (UNFOLD-based approach).

Parameters

kspace_data	Re-ordered data of size (Nx, N_acq, Nz, nph, ncoils).
sigmak	Noise level in kspace_data, as determined by eval_knoise_cine.
b1snr_thr	SNR threshold for sensitivity maps.
N_acq	Number of sampled lines per frame.
Nx	Dimension along the x/kx direction.
Ny	Dimension along the y/ky direction.
Ny_proc	Ny if sampling density $>$ 1 at ky$\sim$=0.
Nz_proc	Dimension along the z/kz direction.
ncoils	Number of coil elements in array.
Nf	Number of frames in kspace_data.
fr1	1st frame considered even (0) or odd (1)
fill_fac	Factor to make ky lines integer.
first_line	First line for cropping (0 = off).
filt_b1	Filter to select ky-kz region for B1 sensitivity mapping.
kline_o	Sampling function, odd frames.
kline_e	Sampling function, even frames.
samp_dens	Sampling density, in ky-kz plane.
mngr	FFT manager
bufsp	Memory space, to work in.
b1	calculated B1 sensitivity maps.

5.10.2.18 int self_ref_b1_via_geyser (COMPLEX * *datain,* dataheader * *hdr,* int * *PhaseLines,* COMPLEX * *ptrW,* fil_fft_mngr * *mngr,* int *m*)

Compute the coil sensitivity estimate for one frame of data, using GRAPPA as an intermediary to fill in any gaps (of $2\Delta k$) in the low-frequency coordinate range of the acquired k-space data.

For reference, see Hoge WS and Brooks DH, "Using GRAPPA to improve auto-calibrated coil sensitivity estimation for the SENSE family of parallel imaging reconstruction algorithms," *Magn. Reson. Med.*, Magn Reson Med, 2008; 60(2):462-467.

Parameters

datain	input k-space data (size: P-N-L)
hdr	a header to describe the data and associated output matrix size
PhaseLines	a list of acquired phase encode indices, of length P
ptrW	A buffer to store output coil sensitivity maps. (size: M-N-L)
mngr	a pointer to an FFT manager
m	binary flag: use a mask? default: 0

5.10.2.19 int spacerip_for_cine (short *Nx*, short *N_o*, short *N_e*, short *N_c*, short *Ny*, int *Ny_proc*, short *Nz_proc*, short *Nf*, short *ncoils*, float *pF*, KCOORD * *kline_o*, KCOORD * *kline_e*, KCOORD * *kline_c*, COMPLEX * *b1*, COMPLEX * *buf*, COMPLEX * *data*, COMPLEX * *spacerip*, short *recon_flag*)

An implementation of "Sensitivity profiles from an array of coils for encoding and reconstruction in parallel (SPACE RIP)." W. E. Kyriakos, et. al. *Magn Reson Med.* 44(2):301-308, August 2000.

SPACE RIP can be used to reconstruct data sampled non-uniformly on a rectilinear sampling grid.

Parameters

Nx	Size of input dataset, along kx (readout)
N_o	k locs acquired in odd ky-kz frames
N_e	k locs acquired in even ky-kz frames
N_c	k locs acquired in combined frames
Ny	Full matrix size along ky (phase)
Ny_proc	Ny used for processing, in vdsense
Nz_proc	Nz used for processing
Nf	number of frames (either time or phase)
ncoils	Number of coils
pF	partial-Fourier (1=fully sampled, 0.5=50% sampled)
kline_o	lines acquired on 'odd' frames
kline_e	lines acquired on 'even' frames
kline_c	lines acquired on combined frames
b1	pointer to sensitivity map data
buf	pointer to available memory block
data	pointer to input data
spacerip	pointer to reconstructed data
recon_flag	0 for low t freqs, 1 for high t freqs

5.10.2.20 int sprip3d_conv_pts2y (int *nph*, int * *phz_y*, int * *phz_z*, int *M*, int *N*, int * *Y*)

calculate the array index values for the specified acquired phase encode points. i.e. maps (phz_y,phz_z) = (2,1) of a (MxN =) 10 x 8 image to y = 12

y[i] = phz_z[i] * M + phz_y[i]

```
  01234567
0 ........
1 ........
2 .x......
3 ........
4 ........
5 ........
6 ........
7 ........
8 ........
9 ........
```

Note: the C convention (which is followed here) is to start counting at zero, which differs from Matlab, which starts counting vector elements at one.

Parameters

nph	number of acquired phase encode points
phz_y	y locs of each phase encode point
phz_z	z locs of each phase encode point
M	total span of the FOV, along y
N	total span of the FOV, along z
Y	calcuated list of array index values

5.10.2.21 void sprip3d_lsqr_init (int *nph,* int *M,* int *N,* int *L*)

setup the memory allocation and FFT configuration for the LSQR reconstruction algorithm of 3D imaging data, sprip3d-_lsqrsolv().

Parameters

nph	number of acquired phase encode lines
M	total span of the phase-encode dimension
N	total span of the phase-encode dimension
L	number of coils

5.10.2.22 void sprip3d_lsqr_post ( )

tear-down the memory allocation and FFT configuration used during the LSQR reconstructions of 3D data.

5.10.2.23 void sprip3d_lsqrsolv (int * *Y,* int *nph,* int *m,* int *n,* int *L,* COMPLEX * *W,* COMPLEX * *Sk,* COMPLEX ** *I,* float * *inlams,* int *lamsize,* int *maxits,* double * *xnormout,* double * *Rout,* int *tid*)

Solves the inverse problem associated with one slice of the 3D SPACE RIP reconstruction algorithm, utilizing fast matrix vector products available via FFT operations *and* embedded Krylov techniques to quickly find solutions associated with multiple regularization values.

Parameters

Y	phase encode index list
nph	number of phase encodes
m	size of full phase encode span along y
n	size of full phase encode span along z
L	number of coils
W	(M,N,L) coil sensitivity data
Sk	(nph,L) acquired data
I	pointer to reconstructed image data. If this value points to an existing memory block, a single image is returned, corresponding to the highest regularization parameter value with no aliasing in the image. If a NULL pointer is passed, then all reconstructions (one for each lambda) is returned. After copying the most desirable image, free the buffer to prevent memory leaks.

inlams	list of lambda's
lamsize	number of lambda's in the lams list
maxits	max number of iterations
xnormout	returns the norm of solution, for each lambda
Rout	returns the norm of residual error, for each lambda
tid	unused

5.10.2.24 void sprip_cgsolv (int * *Y,* int *nph,* int *n,* int *L,* COMPLEX * *Wx0,* COMPLEX * *u,* COMPLEX * *x,* COMPLEX *lambda,* int *k_max,* int *tid*)

Solves the inverse problem associated with one column of the SPACE RIP reconstruction algorithm, utilizing fast matrix vector products available via FFT operations

Parameters

Y	phase encode index list
nph	number of phase encodes
n	size of full phase encode span
L	number of coils
Wx0	(n,L) coil sensitivity data
u	(nph,L) acquired data
x	(length n) reconstructed image data
lambda	DLS regularization setting
k_max	max number of iterations
tid	thread id

5.10.2.25 void sprip_cgsr_init (int *nph,* int *N,* int *L,* int *threadnum*)

Set up memory allocation blocks and FFT plans needed for `cgsolv`

Parameters

nph	number of acquired phase encode lines
N	total span of the phase-encode dimension
L	number of coils
threadnum	number of threads to prepare for

5.10.2.26 void sprip_cgsr_post ( )

Tear-down the allocated memory blocks and FFT mananger used with `cgsolv`

5.10.2.27 void sprip_cgsr_recon (float * *phz_encode_list,* int *P,* int *N,* int *M,* int *L,* COMPLEX * *W,* COMPLEX * *Sk,* double *tau,* int *maxits,* COMPLEX * *rho,* int *tmax*)

Reconstruct an image using the Space-RIP paradigm, solving the linear system via the method of conjugate gradients

Parameters

phz_encode_list	the phase encodes used (length P vector) range = [0, N-1], $<$ with the center of k-space at N/2
P	number of phase encodes used
N	number of reconstructed samples along phase encode direction

M	number of samples along readout direction
L	number of coils
W	coil sensitivity estimate in spatial domain of size (N,M,L)
Sk	acquired data in k-space domain of size (P,M,L)
tau	the regularization parameter
maxits	max number of iterations
rho	the reconstructed image (size = M-by-N)
tmax	max number of threads to use. valid range: [0..M] If 0, then default number is used. set to non-zero if memory workspace is pre-allocated.

5.10.2.28 void sprip_lsqr_init (int *nph,* int *N,* int *L,* int *threadnum,* int *lamsize,* int *maxits*)

setup the memory allocation and FFT manager for the LSQR reconstruction algorithm, `lsqrsolv`.

Parameters

nph	number of acquired phase encode lines
N	total span of the phase-encode dimension
L	number of coils
threadnum	number of threads to prepare for
lamsize	maximum number of lambda's
maxits	maximum number of iterations

5.10.2.29 void sprip_lsqr_post ( )

tear-down the memory allocation and FFT manager used during the LSQR reconstructions.

5.10.2.30 void sprip_lsqr_recon (float * *phz_encode_list,* int *P,* int *N,* int *M,* int *L,* COMPLEX * *W,* COMPLEX * *Sk,* float * *lams,* int *numlams,* COMPLEX * *D,* int *maxits,* COMPLEX ** *rho,* double * *xout,* double * *rout,* int *tmax,* float * *usedlams*)

reconstruct an image using the SPACE-RIP paradigm, solving the linear system using the LSQR-Hybrid algorithm. This algorithm embeds the regularization parameter search within each iteration, efficiently producing multiple solutions for each regularization paramater (lambda) value.

Efficiently solves:

$$\min_x \{ \|Ax - b\|_2 + \lambda \|Dx\|_2 \}$$

for multiple vales of λ, where D is a diagonal regularization operator matrix.

Parameters

phz_encode_list	the phase encodes used (length P vector) range = [0, N-1], with the center of k-space at N/2
P	number of phase encodes used
N	number of reconstructed samples along phase encode direction
M	number of samples along readout direction
L	number of coils
W	coil sensitivity estimate in spatial domain of size (N, M, L)
Sk	acquired data in k-space domain of size (P, M, L, nimgs)
lams	array of regularization parameter. values (if *lams == NULL, use the default of 10^{-4} to 10^1 on a logarithmic scale.)

numlams	number of lambda values in the array (max=50). if zero, then default=20 is used
D	a length N vector for varied regularization along the reconstructed dimension. Set to NULL to use the identity matrix.
maxits	the maximum number of iterations
rho	the reconstructed image(s). If given a pointer, then the algorithm will return one image of size M-by-N. if (∗rho==NULL), it will return an image for each of the regularization values. To avoid memory leaks, clear the returned buffer after the most desirable image is copied.
xout	returns the estimated norm of the solution for each lambda
rout	returns the norm of residual error, for each lambda
tmax	maximum number of threads to use
usedlams	an array of size M, used to return the regularization value determined by the LSQR-Hybrid algorithm for each solved problem.

5.10.2.31 void sprip_lsqrsolv (int ∗ *Y,* int *nph,* int *N,* int *L,* COMPLEX ∗ *Wx0,* COMPLEX ∗ *Sk,* COMPLEX ∗∗ *I,* float ∗ *lams,* int *lamsize,* COMPLEX ∗ *D,* int *maxits,* double ∗ *xnormout,* double ∗ *Rout,* int *tid,* float ∗ *lout*)

Solves the inverse problem associated with one column of the SPACE RIP reconstruction algorithm, utilizing fast matrix vector products available via FFT operations *and* embedded Krylov techniques to quickly find solutions associated with multiple regularization values.

Parameters

Y	phase encode index list
nph	number of phase encodes
N	size of full phase encode span
L	number of coils
Wx0	(N,L) coil sensitivity data
Sk	(nph,L) acquired data
I	reconstructed image data
lams	list of lambda's
lamsize	number of lambda's in the lams list
D	a vector to use in non-I regularization. Set to NULL if not needed/desired.
maxits	max number of iterations
xnormout	returns the norm of solution, for each lambda
Rout	returns the norm of residual error, for each lambda
tid	thread id

5.10.2.32 void vdsense (short *Nx,* short *N_acq,* short *Ny,* short *Ny_proc,* short *Nz_proc,* short *Nf,* short *ncoils,* short *fill_fac,* short *fr1,* short *first_line,* float *pF,* fil_fft_mngr ∗ *mngr,* KCOORD ∗ *kline_o,* KCOORD ∗ *kline_e,* float ∗ *samp_dens,* float ∗ *weight_pF,* COMPLEX ∗ *buf,* DATATYPE ∗ *Uarray,* COMPLEX ∗ *data,* COMPLEX ∗ *vdsen_ima*)

The reconstruction strategy implemented by Variable density SENSE (VD-SENSE) is to apply a Cartesian SENSE reconstruction to data that is not necessarily uniformly sub-sampled. The processing proceeds in three steps.

First, the coil sensitivity estimates are used to construct reconstruction (or un-mixing or un-aliasing) operators, based on the highest acceleration factor employed in the sampling scheme. Second, the raw data is zero padded and transformed to the spatial domain. Finally, the reconstruction operators are applied to the spatial domain representation of the acquired image data, to form the un-aliased image.

Parameters

Nx	Size in x direction
N_acq	number of acquired samples along y
Ny	Size in y direction
Ny_proc	Ny used for processing
Nz_proc	Nz used for processing (number of slices)
Nf	Size in time direction
ncoils	number of coils
fill_fac	fill factor, to accomodate non-int samp pts
fr1	whether 1st frame considered even (0) or odd (1)
first_line	First y line kept, when cropping recon FOV
pF	partial-Fourier (1=fully sampled, 0.5=50% sampled)
mngr	a pointer to an FFT manager
Uarray	Contains an inverted SENSE matrix U for each pixel, to perform the unaliasing.

5.10.2.33 void vdsense_apply (short *Nx,* short *Ny,* short *Nz,* short *Nf,* short *ncoils,* COMPLEX * *sen_before,* DATATYPE * *Uarray,* COMPLEX * *sen_after*)

Apply variable-density SENSE.

Parameters

Nx	Size in x direction.
Ny	Size in y direction.
Nz	Size in z direction.
Nf	Size time direction.
ncoils	Number of receiver coils.
sen_before	Data, before vdsense. (size: Nx*Ny*Nz*Nf*ncoils*sizeof(COMPLEX))
Uarray	Contains an inverted matrix U for each pixel, to unalias.
sen_after	Data, after vdsense. (size: Nx*Ny*Nz*Nf*sizeof(COMPLEX))

5.10.2.34 void vdsense_prep (short *Nx,* short *Ny,* short *Nz,* short *ncoils,* float *thresh,* int *accel,* XCOORD * *alias,* COMPLEX * *b1,* DATATYPE * *Uarray*)

Invert sensitivity matrices, in preparation for a vdsense reconstruction.

See inv_sens_matrix() for a description of the *thresh* parameter.

Parameters

Nx	Size in x direction.
Ny	Size in y direction.
Nz	Size in z direction.
ncoils	Number of receiver coils.
thresh	Threshold for noise tolerance.
accel	the maximum delta k in the sampling pattern
alias	Description of the aliasing PSF (where aliasing comes from).
b1	B1 sensitivity maps. (size: Nx*Ny*Nz*sizeof(COMPLEX))
Uarray	Array where all the inverted matrices will be stored. (size: Nx*Ny*Nz*ncoils*sizeof(COMPLEX))

5.11 Filtering Functions

Functions to apply spatial and temporal filters to k-space data.

Functions

- void apodization (SCAN_INFO hdr, int Nx, float ∗fx, int Ny, float ∗fy, int Nz, float ∗fz, KCOORD ∗kline_o, KCOORD ∗kline_e, COMPLEX ∗data)
- void apply_ramp (COMPLEX ∗data, short Nx, short Ny, short nph, short ncoils, float rampx, float rampy)
- void fermi (short N, float trw, float ∗f)

5.11.1 Detailed Description

Functions to apply spatial and temporal filters to k-space data.

5.11.2 Function Documentation

5.11.2.1 void apodization (SCAN_INFO *hdr,* int *Nx,* float ∗ *fx,* int *Ny,* float ∗ *fy,* int *Nz,* float ∗ *fz,* KCOORD ∗ *kline_o,* KCOORD ∗ *kline_e,* COMPLEX ∗ *data*)

Apply an apodization filter, to reduce ringing.

Parameters

hdr	scan information
Nx	Number of points along kx.
fx	Filter function along kx.
Ny	Number of points along ky.
fy	Filter function along ky.
Nz	Number of points along kz.
fz	Filter function along kz.
kline_o	Sampling function, odd time frames.
kline_e	Sampling function, even time frames.
data	k-space matrix to filter

5.11.2.2 void apply_ramp (COMPLEX ∗ *data,* short *Nx,* short *Ny,* short *nph,* short *ncoils,* float *rampx,* float *rampy*)

Apply a phase ramp to kspace data, to shift the image in the spatial domain.

Parameters

data	Data to apply a phase ramp to.
Nx	Size in kx direction.
Ny	Size in ky direction.
nph	Size time direction.
ncoils	Number of receiver coils.
rampx	Phase increment from x pixel to next.
rampy	Phase increment from y pixel to next.

5.11.2.3 void fermi (short *N,* float *trw,* float ∗ *f*)

Generate a fermi filter.

Parameters

N	Number of points.
trw	Width of transition region where the filter passes from 1% to 99%, expressed as a fraction of the full bandwidth.

5.12 Matrix-Vector Utility Functions

Functions

- COMPLEX inner_product (COMPLEX *a, COMPLEX *b, int n)
- void matrix_transpose (COMPLEX *A, int m, int n)
- void matrix_vector_product (COMPLEX *A, COMPLEX *b, int m, int n, COMPLEX *res)
- void matrix_vector_product_threaded (COMPLEX *A, COMPLEX *b, int m, int n, COMPLEX *res, int numthread)

5.12.1 Detailed Description

A collection of matrix-vector utilities for use with complex-valued data structures

5.12.2 Function Documentation

5.12.2.1 COMPLEX inner_product (COMPLEX * *a*, COMPLEX * *b*, int *n*)

Calculate the inner product, $\sum_{i=1}^{n} a_i^* b_i$, of two complex vectors

Parameters

a	pointer to first vector
b	pointer to second vector
n	length of the vectors

5.12.2.2 void matrix_transpose (COMPLEX * *A*, int *m*, int *n*)

Perform $A = A^T$ where (m,n) is the size of the *input* matrix

Parameters

A	pointer to input data array
m	number of rows in the data
n	number of columns in the data

5.12.2.3 void matrix_vector_product (COMPLEX * *A*, COMPLEX * *b*, int *m*, int *n*, COMPLEX * *res*)

Compute a matrix vector product, $x = Ab$, where $x_i = \sum_j A_{i,j} b_j$.

Parameters

A	a pointer to the complex matrix (*m* rows, *n* columns)
b	a pointer to the complex vector (*n* entrys)
m	number of rows in A
n	number of columns in A
res	a pointer to the result, a complex vector, $x = Ab$, (*m* entrys)

5.12.2.4 void matrix_vector_product_threaded (COMPLEX * *A*, COMPLEX * *b*, int *m*, int *n*, COMPLEX * *res*, int *numthread*)

A threaded version of matrix_vector_product

Parameters

A	a pointer to the complex matrix (*m* rows, *n* columns)
b	a pointer to the complex vector (*n* entrys)
m	number of rows in A
n	number of columns in A
res	a pointer to the result, a complex vector, $x = A'b$, (*m* entrys)
numthread	default=16

5.13 File I/O

Functions to read and write image and data files.

Functions

- void copy_data_obj (DATA_OBJ ∗dest, DATA_OBJ ∗src)
- void display_params (SCAN_INFO hdr)
- void dump_out (char ∗name, int nc_out, int nf_out, int nz_out, int ny_out, int nx_out, short temp, COMPLEX ∗bin)
- void free_data_obj (DATA_OBJ ∗data_obj)
- void init_data_obj (DATA_OBJ ∗data)
- int read_nd_data (char ∗filename, char ∗∗ptr, char ∗frmt, int ∗size, int ∗szI)
- int read_nd_data2 (char ∗filename, DATA_OBJ ∗dat)
- int write_nd_data (char ∗filename, void ∗bin, char ∗frmt, int size, int ∗sz)
- int write_nd_data2 (char ∗filename, DATA_OBJ ∗dat)

5.13.1 Detailed Description

Functions to read and write image and data files. The library provides a number of functions to pass data back and forth between C and Matlab.

One method is the .nd data format, which is a straight forward way to save real or complex multi-dimensional data. The .nd file format is:

- 4-byte int giving the total number of bytes in the file (filesize)
- 4-byte char string 'nddf', to specify the ND data format
- 4-byte int giving the number of dimensions in the array. (size)
- array of 4-byte ints, of length (size), giving number of samples in each dimension
- 4-byte char string declaring the data type. The first 3 chars declare the data type, with the last char for real or complex.

  The seven possibilities are:

  - (short, 16b) integers, real or complex : 'intr' or 'intc'
  - floats, real or complex : 'fltr' or 'fltc'
  - doubles, real or complex : 'dblr' or 'dblc'
  - (8b) character data (strings) : 'char'
- followed by 'the data'

Version 2 allows meta-data describing the data to also be stored.

- 4-byte int describing the length of the meta-data block, in bytes [8 + sizeof(char)∗[number of metadata bytes]],
- 4-byte char string 'meta',
- the meta-data.

Alternatively, one can use the dump_out function to write memory contents to disk. This format writes the raw binary data in one file, and then writes an associated text file describing the format of the data.

5.13.2 Function Documentation

5.13.2.1 void copy_data_obj (DATA_OBJ * *dest,* DATA_OBJ * *src*)

clone a copy of a data object

if dest is a clean object, then the memory will be allocated.

if not, (i.e. dest has pre-allocated data memory) the copy will proceed if (dest->init >= src->init).

5.13.2.2 void display_params (SCAN_INFO *hdr*)

Display some of the scan parameters contained in the SCAN_INFO header

5.13.2.3 void dump_out (char * *name,* int *nc_out,* int *nf_out,* int *nz_out,* int *ny_out,* int *nx_out,* short *temp,* COMPLEX * *bin*)

Write data to file, for debugging. The fuection output two files, one file is a binary format with the data. The second, with suffix '_dims' records the data structure (number of coils, frames, Nz, Ny, Nx, etc.). Data can be read in to Matlab using read_dump .

Parameters

name	output filename
nc_out	number of coils
nf_out	number of temporal frames
nz_out	number of slices
ny_out	size along phase-encode dimension
nx_out	size along read-out dimension
temp	(unused)
bin	the data to write to disk

5.13.2.4 void free_data_obj (DATA_OBJ * *data_obj*)

release all allocated memory associated with the input data object.

- sets the data_obj->type to NULL
- frees the data_obj->data
- clears the data_obj->init value (to '0')
- frees the data_obj->metadata, if non-NULL

5.13.2.5 void init_data_obj (DATA_OBJ * *data*)

initialize the data points and array values for an empty data object

5.13.2.6 int read_nd_data (char * *filename,* char ** *ptr,* char * *frmt,* int * *size,* int * *szl*)

import/read an N-D data array file (output by savend)

Usage:

```
COMPLEX *data;
char *tmpp;
int n, cnt, sz[4];
char  dtype[5] = {0,0,0,0,0};

read_nd_data( "inputdata.nd", &tmpp, dtype, &n, sz );
```

```
if ( strcmp(dtype,"dblc")==0 ) {
  data = (COMPLEX *) tmpp;
}
printf("read_nd_data read an %s %d-dimensional array of size ",dtype,n);
for ( cnt=0; cnt<n; cnt++ ) { printf(" %d",sz[cnt]);} printf(".\n");
```

Returns

Returns a 0 if read is successful, a non-zero value otherwise. Error messages are reported to STDERR.

Parameters

filename	Output filename.
ptr	pointer to the data buffer to read data to.
frmt	pointer the file format char field.
size	pointer the returned number of dimensions in the data.
szl	pointer to an array that holds the number of elements in dimension.

5.13.2.7 int read_nd_data2 (char * *filename*, DATA_OBJ * *dat*)

import/read an N-D data array file (output by savend)

Usage:

```
DATA_OBJ img;

init_data_obj( &img );
read_nd_data2( "inputdata.nd", &img );

printf("read_nd_data2 read an %s %d-dimensional array of size ",dtype,n);
for ( cnt=0; cnt<n; cnt++ ) { printf(" %d",sz[cnt]);} printf(".\n");
```

Returns

Returns a 0 if read is successful, a non-zero value otherwise. Error messages are reported to STDERR.

This version (version 2) can read meta-data stored at the end of the file.

Parameters

filename	Output filename.
dat	pointer to the data object to populate.

5.13.2.8 int write_nd_data (char * *filename*, void * *bin*, char * *frmt*, int *size*, int * *sz*)

write an N-D data array to disk suitable for readnd

Usage:

```
int sz[4];
COMPLEX *ImageData;

sz[0] = (int)hdr.xres;
sz[1] = (int)hdr.yres;
sz[2] = (int)hdr.CinePhases;
sz[3] = (int)hdr.Ncoils;
write_nd_data( "I.dat", ImageData, "dblc", 4, sz );
```

Returns

Returns a 0 if read is successful, a non-zero value otherwise. Error messages are reported to STDERR.

Parameters

filename	input data filename
bin	a pointer to the output data
frmt	a format descriptor of the data type: 'char', 'intr', 'intc', 'fltr', 'fltc', 'dblr', or 'dblc'.
size	the number of dimensions spanned by the data
sz	the number of elements in each dimension

5.13.2.9 int write_nd_data2 (char * *filename*, DATA_OBJ * *dat*)

export/write an N-D data array file (to be read by readnd) along with any associated meta-data

Usage:

```
DATA_OBJ img;

init_data_obj( &img );

// some processing, saving the data to img.data

write_nd_data2( "img_data.nd", &img );
```

This version (version 2) will append meta-data stored in the img.metadata field of the object

Parameters

filename	Output filename.
dat	pointer to the data object to write.

5.14 EPI specific functions

Provides common corrections needed for EPI data.

Functions

- int calc_vrgf_data_matrix (DATA_OBJ ∗vrgf, int input_res, int output_res, float time_attack, float time_flat, float time_decay, float time_sample_delay, float time_sampling_window)
- int correct_rampsamp (DATA_OBJ ∗k, DATA_OBJ ∗vrgf, void ∗scratchbuf)
- int ngc_apply_phase_shift (DATA_OBJ ∗k, DATA_OBJ ∗ramp, fil_fft_mngr ∗mngr, NGC_MEM ∗ngc_mem)
- int ngc_compute_phase_shift (DATA_OBJ ∗ngc_even, DATA_OBJ ∗ngc_odd, DATA_OBJ ∗ramp, fil_fft_mngr ∗mngr, NGC_MEM ∗ngc_mem)
- int ngc_mem_init (int ∗sz, int nEl, NGC_MEM ∗ngc_mem)
- int ngc_mem_quit (NGC_MEM ∗ngc_mem)

5.14.1 Detailed Description

Provides common corrections needed for EPI data. These functions include that ability to grid data acquired on EPI readout gradient ramps and to correct for offsets between even and odd EPI readout lines.

See also:

- EPI Ghost Ellimination via Spatial and Temporal Encoding (GESTE)
- Vendor Specific Functions: GE
- Vendor Specific Functions: Siemens

5.14.2 Function Documentation

5.14.2.1 int calc_vrgf_data_matrix (DATA_OBJ ∗ *vrgf,* int *input_res,* int *output_res,* float *time_attack,* float *time_flat,* float *time_decay,* float *time_sample_delay,* float *time_sampling_window*)

This function computes the matrix to regrid EPI data that is sampled on the readout gradient attack and decay slopes.

Parameters

vrgf	the output regridding matrix
input_res	number of input data on readout
output_res	number of data points on the readout line after gridding
time_attack	the 'ramp up' time for the gradient (in msec)
time_flat	the 'flat top' time for the gradient (in msec)
time_decay	the 'ramp down' time for the gradient (in msec)
time_sample_-delay	the time (in msec) between the start of the readout gradient and the start of the sampling window
time_sampling_-window	length of time (in msec) the sampling window is 'on'

5.14.2.2 int correct_rampsamp (DATA_OBJ ∗ *k,* DATA_OBJ ∗ *vrgf,* void ∗ *scratchbuf*)

a function to regrid EPI data sampled on gradient ramps, using the provided gridding object

Parameters

k	the input ramp-sampled data (of type "fltc" or "dblc")
vrgf	the ramp-sampling gridding matrix. Needs to be of type "fltr", with vrgf.sz[0] == k.sz[0]
scratchbuf	A pointer to some workspace memory. Needs to be as large as the (dblc) input data memory size. If NULL, the temporary scratch space will be internally allocated.

5.14.2.3 int ngc_apply_phase_shift (DATA_OBJ * *k,* DATA_OBJ * *ramp,* fil_fft_mngr * *mngr,* NGC_MEM * *ngc_mem*)

Applies a pre-estimated phase correction to correct the k-space shift between the odd/even lines of EPI data.

Parameters

k	input k-space data, of size [Nx Ny Ncoil]
ramp	input phase correction data, of length [ncoil]
mngr	and FIL FFT manager
ngc_mem	an pre-initialized NGC object to hold the memory buffers

5.14.2.4 int ngc_compute_phase_shift (DATA_OBJ * *ngc_even,* DATA_OBJ * *ngc_odd,* DATA_OBJ * *ramp,* fil_fft_mngr * *mngr,* NGC_MEM * *ngc_mem*)

Computes the estimated shift (for each coil of data) based on cross-correlation between the input even and odd EPI readout lines.

Input data type can be either "dblc" of "fltc".

Parameters

ngc_even	input set of even (+Gx) EPI phase reference lines. Should be 2D, with size [Nx Ncoils]
ngc_odd	input set of odd (-Gx) EPI phase reference lines. Should be 2D, with size [Nx Ncoils]
ramp	an object to store the output estimated phase correction for each coil. Should be a "fltr" object of length [Ncoils]. If ramp->data is set to NULL, data object will be initialized internally.
mngr	an FIL FFT manager
ngc_mem	an pre-initialized NGC object to hold the memory buffers

5.14.2.5 int ngc_mem_init (int * *sz,* int *nEl,* NGC_MEM * *ngc_mem*)

Initializes the memory structures needed for Nyquist ghost correction when using the 3-line-at-start-of-echo-train method.

Parameters

sz	a 2D int array holding the Nx and ncoil sizes of the data used to estimate the phase correction values
nEl	the number of complex data value elements (Nx * Ny * ncoil) in the EPI data to be corrected
ngc_mem	the NGC memory object to hold the initialized memory

5.14.2.6 int ngc_mem_quit (NGC_MEM * *ngc_mem*)

Free the NGC memory

5.15 Utility Functions

Functions

- void calc_samp_dens (KCOORD ∗kline_o, KCOORD ∗kline_e, short N_acq, short Nz_proc, int fill_fac, short Ny, float ∗samp_dens)
- float eval_knoise (SCAN_INFO hdr, COMPLEX ∗kspace_data, KCOORD ∗kline_o, COMPLEX ∗bufspace)
- void extract_ksubset (short ncoils, short nf, short nx, KCOORD ∗kline_full, KCOORD ∗kline_sub, short nl_full, short nl_sub, COMPLEX ∗fullmat, COMPLEX ∗submat)
- void fil_version ()
- int imnum_out (short next_im, short z_anat, short Nf, short ncoils, short dump_b1maps)
- void kstretch (short Nx, short Ny, short Nz_proc, short N_acq, short fill_fac, short ifr, short fr1, KCOORD ∗kline_o, KCOORD ∗kline_e, COMPLEX ∗data, COMPLEX ∗data_str)
- void select_kcenter (short firsty, short lasty, short Ny, short firstz, short lastz, short Nz, float ∗filt)
- int slice_number (int z, int Nsl, int z_first, char ∗application)
- void weigh_kykz (short ncoils, short nf, short nz, short ny, short nx, float ∗weight, COMPLEX ∗data)

5.15.1 Detailed Description

5.15.2 Function Documentation

5.15.2.1 void calc_samp_dens (KCOORD ∗ *kline_o,* KCOORD ∗ *kline_e,* short *N_acq,* short *Nz_proc,* int *fill_fac,* short *Ny,* float ∗ *samp_dens*)

Calculate the sampling density.

Parameters

kline_o	Sampling function, odd time frames.
kline_e	Sampling function, even time frames.
N_acq	Number of acquired lines per frame, i.e., length of kline_o and _e.
Nz_proc	Size along z, in processing.
fill_fac	Factor to make ky lines integer.
Ny	Matrix size along y, actual dataset.
samp_dens	Sampling density, size = Nz_proc∗fill_fac∗Ny.

5.15.2.2 float eval_knoise (SCAN_INFO *hdr,* COMPLEX ∗ *kspace_data,* KCOORD ∗ *kline_o,* COMPLEX ∗ *bufspace*)

Evaluate the standard deviation of the noise on the k-space data. The returned value is used to set the regularization threshold in vdsense. Get an estimate of the noise level. Look only at the corners of the kx-ky space (don't look at the central 20% part on both axes, which may contain a lot of signal). Subtract the time average to remove most of the remaining signal. Odd and even time frames are different, so use different time averages. The remaining data should be mostly noise. Find its standard deviation.

Parameters

hdr	Contains scan info.
kspace_data	k-space data (size ncoils∗Nf∗N_acq∗Nx)
kline_o	Sampling function, odd time frames.

bufspace	Memory space to work in. (size:ncoils∗Nf∗N_acq∗Nx)

5.15.2.3 void extract_ksubset (short *ncoils*, short *nf*, short *nx*, KCOORD ∗ *kline_full*, KCOORD ∗ *kline_sub*, short *nl_full*, short *nl_sub*, COMPLEX ∗ *fullmat*, COMPLEX ∗ *submat*)

From a matrix with k-space lines listed in kline_full, generate a subset matrix featuring only the lines listed in kline_sub.

Parameters

ncoils	Number of coils.
nf	Number of time frames.
nx	Size along x or kx.
kline_full	List of ky/kz for lines in fullmat
kline_sub	List of ky/kz for lines in submat
nl_full	Number of lines in kline_full.
nl_sub	Number of lines in kline_sub.
fullmat	Full matrix to subsample.
submat	Subsampled matrix.

5.15.2.4 void fil_version ( )

Output the current version and compile time of the library.

5.15.2.5 int imnum_out (short *next_im*, short *z_anat*, short *Nf*, short *ncoils*, short *dump_b1maps*)

Calculates the image number for the next image to be written out.

Returns

```
if (dump_b1maps == 1),
   next_im+z_anat*(Nf+ncoils);
else,
   next_im+z_anat*Nf;
```

Parameters

next_im	1st image number to use for output.
z_anat	Slice number, in anatomical order.
Nf	Number of time frames.
ncoils	number of receiver coils.
dump_b1maps	Flag to display the B1 maps.

5.15.2.6 void kstretch (short *Nx*, short *Ny*, short *Nz_proc*, short *N_acq*, short *fill_fac*, short *ifr*, short *fr1*, KCOORD ∗ *kline_o*, KCOORD ∗ *kline_e*, COMPLEX ∗ *data*, COMPLEX ∗ *data_str*)

zero-pads the k-space data set, placing sub-sampled data in the full-sized data matrix with zeros at unsampled locations.

Take the compact representation of k-space, with a list of N_acq lines, each one with Nx real and imaginary values, and 'stretch' it into its normal shape, a cube having Nx by (fill_fac∗Ny) by Nz_proc complex elements. (Note that due to sparse sampling and filling, fill_fac∗Ny∗Nz_proc is typically much greater than N_acq).

Parameters

Nx	Size in kx direction.
Ny	Size in ky direction.
Nz_proc	Size in kz direction.
N_acq	Number of acquired ky-kz lines.
fill_fac	Factor by which raw data must be stretched so that all the (fractional) ky lines become integers.
ifr	Time frame under consideration.
fr1	Whether 1st frame is considered even (0) or odd (1).
kline_o	Sampling function, odd time frames.
kline_e	Sampling function, even time frames.
data	Data to uncompress.
data_str	Uncompressed version of data.

5.15.2.7 void select_kcenter (short *firsty*, short *lasty*, short *Ny*, short *firstz*, short *lastz*, short *Nz*, float * *filt*)

Generate a 2D Gaussian filter to extract a central region in the ky-kz plane.

Parameters

firsty	1st ky line in selected region.
lasty	Last ky line in selected region.
Ny	Size of filter along ky.
firstz	1st kz line in selected region.
lastz	Last kz line in selected region.
Nz	Size of filter along kz.
filt	Calculated filter.

5.15.2.8 int slice_number (int *z*, int *Nsl*, int *z_first*, char * *application*)

Convert the order in which slies are acquired into the anatomical order in which the output should be written.

Parameters

z	Slice number, according to acquisition.
Nsl	Number of slices
application	Specific MR application under consideration.

5.15.2.9 void weigh_kykz (short *ncoils*, short *nf*, short *nz*, short *ny*, short *nx*, float * *weight*, COMPLEX * *data*)

Apply a correction in the ky-kz plane, e.g., to correct for sampling density.

data[kx][ky] = data[kx][ky] / weight[ky];

Parameters

ncoils	Number of coils.
nf	Number of time frames.
nz	Size along kz.
ny	Size along ky.
nx	Size along x or kx.

weight	Correction to apply in ky-kz plane.
data	Data to be corrected.

5.16 EPI Ghost Ellimination via Spatial and Temporal Encoding (GESTE)

Functions

- int geste_calibrate__all_slices (DATA_OBJ ∗combo, float ∗phase_list, GESTE_MEM ∗mem)
- void geste_init (GESTE_MEM ∗mem_obj, DATA_OBJ ∗data_frame)
- int geste_interleave_frames (DATA_OBJ ∗even_frame, DATA_OBJ ∗odd_frame, DATA_OBJ ∗combo, int mode, GESTE_MEM ∗mem)
- void geste_quit (GESTE_MEM ∗mem_obj)
- int geste_recon__all_slices (DATA_OBJ ∗data_frame, float ∗phase_list, DATA_OBJ ∗img, GESTE_MEM ∗mem, int polarity)
- int pull_readout_data (int ndim, int ∗sz, COMPLEX ∗P, int indx_in, COMPLEX ∗Pfull, int indx_out)
- int run_geste (DATA_OBJ ∗even_frame, DATA_OBJ ∗odd_frame, DATA_OBJ ∗raw_frame, float ∗phase_list, DATA_OBJ ∗img, GESTE_MEM ∗mem)

5.16.1 Detailed Description

This category of functions is designed to remove Nyquist ghosts from EPI images. It is a C-code implementation of the method described in [Q].

The strategy is to use temporal encoding in a series of EPI acquisitions, so that each line of k-space is acquired using readout gradients of alternating polarity over the course of the image series.

If one interleaves data from either all the positive or all the negative readouts in two frames that were acquired in this fashion, the gradient errors associated with each readout polarity will be consistent. This reduces Nyquist ghosts substantially. The authors of PLACE [R] showed that when these images are combined coherrently, the Nyquist ghosting can be further reduced due to ghost cancelation. These images result in lower temporal resolution, however, due to the need for data interleaving.

In GESTE, the Nyquist-ghost-free PLACE image is used to calibrate parallel imaging reconstruction coefficients. These coefficients are then used in the original data to reconstruct two images at each time frame: one associated with negative readout polarity gradients, and one associated with the positive readout polarity gradients. When these images are combined coherently, residual pMRI artifacts will cancel, yeilding images with superior ghost suppression while maintaining the original temporal resolution.

For an example on how to call the GESTE functions, consult the geste.c Matlab MEX file.

Q. Hoge WS, Tan H, and Kraft RA, "Robust Elimination of EPI Nyquist Ghosts via Spatial and Temporal Encoding." *Magn Reson Med*, 2010; 64(6):1781-1791. DOI

R. "Correction for geometric distortion and N/2 ghosting in EPI by phase labeling for additional coordinate encoding (PLACE)." Qing-San Xiang and Frank Q. Ye. *Magn Reson Med*, 57(4):731-741, 2007. DOI

5.16.2 Function Documentation

5.16.2.1 int geste_calibrate__all_slices (DATA_OBJ ∗ *combo,* float ∗ *phase_list,* GESTE_MEM ∗ *mem*)

Using data combined from two interleaved frames, calculate the GRAPPA reconstruction parameters for all slices.

Parameters

combo	the calibration data frame. 4D, size: kx-ky-z-coil
phase_list	(unused) a list of the sampled phase encode lines.
mem	an initialized GESTE object of scratch memory buffers

5.16.2.2 void geste_init (GESTE_MEM * *mem_obj,* DATA_OBJ * *data_frame*)

Initialize the memory buffers required by GESTE. For the data_frame, only the size of the supported data needs to be declared prior to calling the init function.

Parameters

mem_obj	GESTE object of sratch memory buffers
data_frame	a pointer to a 4D data object (sized kx-ky-slices-coil) to be processed by GESTE (for image size information)

5.16.2.3 int geste_interleave_frames (DATA_OBJ * *even_frame,* DATA_OBJ * *odd_frame,* DATA_OBJ * *combo,* int *mode,* GESTE_MEM * *mem*)

The first step of GESTE (comparable to PLACE):

A interleave input kspace of alternating readout polarity.

B phase align the interleaved images in the image domain,

C and then add the two frames

D if (mode==0), convert the output back in to k-space domain

Parameters

even_frame	frame1: even lines are positive readout k-space
odd_frame	frame2: odd lines are positive readout k-space
combo	combined data
mode	if mode=0: revert the output data back to k-space before returning
mem	an initialized memory object

5.16.2.4 void geste_quit (GESTE_MEM * *mem_obj*)

Clear and release the GESTE memory buffers.

An array of objects holding GRAPPA reconstruction paramemters can be cleared as well. If not needed, set mem_obj->nslices to '0' and mem_obj->gparm to 'NULL'.

Parameters

mem_obj	GESTE object of sratch memory buffers

5.16.2.5 int geste_recon_all_slices (DATA_OBJ * *data_frame,* float * *phase_list,* DATA_OBJ * *img,* GESTE_MEM * *mem,* int *polarity*)

GESTE reconstruction of EPI data frames using previously calculated GRAPPA parameters.

Parameters

data_frame	The raw input EPI data: ndim should equal '4', sized: kx-ky-z-coil type: double-complex ('dblc')
phase_list	(UNUSED). List of phase encode locations
img	A preallocated data array to store the reconstructed image. sized: x-y-z-coil
mem	An initialized memory buffer object
polarity	declare whether first k-space line is positive (1, align to even lines) or negative (0, align to odd lines).

5.16.2.6 int pull_readout_data (int *ndim*, int * *sz*, COMPLEX * *P*, int *indx_in*, COMPLEX * *Pfull*, int *indx_out*)

This auxillary function is used to extract all readout data of the same polarity from an EPI data set.

P, and Pfull should be the same size, with dimensions ordered as: kx-ky-z-coil

It assumes that every other line of k-space is sampled at the same polarity.

Parameters

ndim	number of dimensions in data set
sz	size array, describing each dimension
P	the source data
indx_in	the first index of source data to read
Pfull	the destination buffer
indx_out	the first index for the data destination

5.16.2.7 int run_geste (DATA_OBJ * *even_frame*, DATA_OBJ * *odd_frame*, DATA_OBJ * *raw_frame*, float * *phase_list*, DATA_OBJ * *img*, GESTE_MEM * *mem*)

An example GESTE reconstruction function, geared towards an fMRI application.

Send two raw images in, get one reconstructed image back.

Parameters

even_frame	raw input EPI data frame. Even lines have positive readout data
odd_frame	raw input EPI data frame. Odd lines have positive readout data
raw_frame	raw input EPI data frame, the frame to reconstruct. Could/should be equal to either even_frame or odd_frame
phase_list	(UNUSED). List of phase encode locations
img	reconstructed image, from raw_frame data
mem	memory buffer object.

5.17 Image Phase Alignment Functions

Functions

- int align_image_phase (COMPLEX ∗P, COMPLEX ∗N, int nz, int ∗sz, float ∗∗phzL, COMPLEX ∗∗phzC, int tmax, short int mode)
- int calc_phase_correlation (Phase_Correlation_Args ∗arg)
- void fit_line_to_phase (COMPLEX ∗v, float ∗mask, int N, float ∗ramp)
- int unwrap_phase (double ∗b, float ∗mask, int N)

5.17.1 Detailed Description

This category of functions is designed to support image phase alignment, to idenitify and remove phase differences introduced by a shifts in kspace betwen two images,

This is useful in Nyquist ghost removal [O] to align data acquired on positive and negative readout gradients in EPI imaging, and in rigid image registration [P].

O. Hoge WS, Tan H, and Kraft RA, "A method to remove Nyquist ghosts from echo planar imaging (EPI) using UNFOLD." in *Proc ISMRM 17th Scientific Meeting*, pg 571, 2009.

P. Hoge WS, "A subspace identification extension to the phase correlation method," *IEEE Trans. Medical Imaging*, 22(2):277-280, Feb. 2003.

5.17.2 Function Documentation

5.17.2.1 int align_image_phase (COMPLEX ∗ *P,* COMPLEX ∗ *N,* int *nz,* int ∗ *sz,* float ∗∗ *phzL,* COMPLEX ∗∗ *phzC,* int *tmax,* short int *mode*)

Align the phase of two input images. This function will modify P and N so that when added coherently, the addition is constructive.

All dimensions beyond the first two will be corrected.

Parameters

P	image associated with pos readout data
N	image associated with neg readout data
nz	number of input image dimensions
sz	array of sizes for each dimension. Note, P and N need to be identical in size
phzL	UNUSED
phzC	UNUSED
tmax	maximum number of threads
mode	if mode=0, the phase shift is split equally to both P and N. if mode=1, the phase shift is applied only to N

5.17.2.2 int calc_phase_correlation (Phase_Correlation_Args ∗ *arg*)

Calculates the phase correlation (normalized cross correlation) between two input matrices

C = conj(A).∗B ./ abs(A.∗conj(A))

So that the function is threadable, the input must be a pointer to the structure:

```
typedef struct{
```

```
  COMPLEX *A;                   // input 1
  COMPLEX *B;                   // input 2
  int Nx;                       // number of rows in A,B
  int Ny;                       // number of cols in A,B
  COMPLEX *C;                   // output
  short int mode;               // 0: split phase correction between A,B; 1: apply phase correction t
} Phase_Correlation_Args;
```

5.17.2.3 void fit_line_to_phase (COMPLEX * *v*, float * *mask*, int *N*, float * *ramp*)

Solves a simple LMS problem to find the slope of a line fit to the (presumed) unwrapped phase of the input vector.

Parameters

v	the input vector
mask	a windowing vector of the same length, to provide a weighted LMS fit if needed.
N	length of v
ramp	the estimate of the line slope.

5.17.2.4 int unwrap_phase (double * *b*, float * *mask*, int *N*)

A 1D phase unwrapping algorithm.

Parameters

b	input vector of phase (angle) values
mask	optional binary weighting vector
N	length of input vectors