Matlab Code for MRI Simulation and Reconstruction

Matthieu Guerquin-Kern

July 27, 2012

Contents

1 Introduction

1.1 Contents and Installation Instructions

This package contains a set of routines and functions for Matlab providing the tools to simulate MRI experiments and reconstruct images out of scanner data. Some functions have been written in C++. Most of them have fall-back counterparts written in Matlab language. To make sure that your machine takes advantage of the C++ code, run `make.m` in Matlab's command prompt. The package was developed and tested under Linux and MacOSX platforms with Matlab R2011b. Partial testing demonstrated that the code is mainly compatible with Octave 3.2. The package is completely untested under Microsoft's OS.

1.2 Origin of the code

Some parts of the code have not been written by the author of this package. Please consider acknowledging the respective authors if you use their code and publish results based on it. The following pieces of code are affected:

- MEX files for wavelet transform by Cédric Vonesch, 2008. Publicly distributed at: `http://bigwww.epfl.ch/algorithms/mltldeconvolution/`.
- MEX files for determining the points inside a polygon Bruno Luong, 2010. Publicly distributed at: `http://www.mathworks.com/matlabcentral/fileexchange/27840-2d-polygon-interior-detection/`.
- Matlab code for n-dimensional n permute k problem by Matt Fig, 2009. Publicly distributed at `http://www.mathworks.com/matlabcentral/fileexchange/11462-npermutek/`
- Matlab functions for the computation of MRI data with Bézier-defined phantoms by Laurent Lejeune under the author's supervision, 2010. See `http://bigwww.epfl.ch/teaching/projects/abstracts/lejeune/index2.html`.

In addition, Marcel Leutenegger's instructions, publicly distributed at `https://documents.epfl.ch/users/l/le/leuteneg/www/MATLABToolbox/ErrorFunction.html`, have been followed in the C++/MEX multi-threaded code that computes the error function of a complex variable.

Some parts of the code, have been written by the author of this package from 2007 to 2011 for the work presented in [1, 2, 3]. This code can obtained by the Biomedical Imaging Group members from the SVN: `https://username@svn.epfl.ch/svn/guerquin-sources/projects`. A specific case is the code for analytical simulations which is publicly distributed at `http://bigwww.epfl.ch/algorithms/mriphantom`.

The matlab and MEX files for gridding and NUFT computation have been coded by the author during spring 2012, inspired by [4] and a partial implementation publicly distributed at `http://web.eecs.umich.edu/~fessler/irt/irt/nufft/greengard/`.

The rest of the code was written by the author of this package in July 2012 for the Biomedical Imaging Group (EPFL, Lausanne, Switzerland) and the Institute for Biomedical Engineering (University and ETH, Zürich, Switzerland).

1.3 Origin of the MRI Scanner Data

The data were collected by Maximilian Häberlin, at ETH Zürich, on a 3T Achieva system (Philips Medical Systems, Best, The Netherlands) and used in [1]. A field camera with 12 probes was used to monitor the actual k-space trajectory [5]. An array of 8 head coils provided the measurements. Please contact Maximilian before publishing any result based on these data. You can currently contact him at `haeberlin@biomed.ee.ethz.ch`.

EPI data This dataset of the brain of a healthy volunteer was acquired with a gradient echo EPI sequence with T2* contrast.

The data was acquired with the following parameters: excitation slice thickness of 4mm, TE=35ms, TR=900ms, flip angle of 80 degrees, and trajectory composed of 13 interleaves, supporting a 200×200 reconstruction matrix with resolution 1.18mm×1.18mm. The oversampling ratio along the readout direction was 1.62.

Spiral data This dataset of the brain of a healthy volunteer was acquired with parameters TR = 1000 ms and TE = 30 ms. The excitation slice thickness was 3 mm with a flip angle of 30 degrees. The trajectory was designed for a FOV of 25 cm with a resolution of 1.5 mm. It was composed of 100 spiral interleaves. The distance between neighboring interleaves for the highest sampled frequencies defined a fraction of the Nyquist sampling density (R = 0.9).

2 Demonstration Scripts

DemoSimuAndRecon.m This script defines a parallel MRI experiment setting, with analytically defined phantoms (see (a)(b)(c) in Figure 1). Simulation can be performed in two different ways and the resulting data is corrupted by noise. This synthetic scanner data is prepared for reconstruction. Finally, several reconstruction methods are performed and the resulting reconstructions are compared to the reference image. Note that even in a very favorable pMRI setting, reconstruction will not be perfect because of the Gibb's phenomenon which reflects the mismatch between the continuous nature of the MRI physics and the discrete nature of the model used for reconstruction [3].

DemoBrainEPI.m This script loads real scanner data from an Echo Planar Imaging experiment (Cartesian k-space sampling) and precomputed receiving coil sensitivity maps. The full dataset is first processed and a reference image is reconstructed out of it (see (d) in Figure 1). Then, a reduced dataset is loaded and a more challenging reconstruction takes place. The resulting image is compared to the reference. Note that the processing of the data includes a modulation such that our convention that defines the origin at the upper left corner of the image is satisfied. Results using this dataset were presented in [1].

Note that undersampled k-space reconstructions from this dataset suffer from artifacts that might be caused by a non homogeneous static field.

DemoBrainSpiral.m This script loads real scanner data from a spiral MRI experiment. By default, precomputed receiving coil sensitivity maps are loaded but the code to compute them can be uncommented. The full dataset is first processed and a reference image is reconstructed out of it (see (e) in Figure 1). Then, a reduced dataset is loaded and a more challenging reconstruction takes place. The resulting image is compared to the reference. Note that the processing of the data includes a modulation such that our convention that defines the origin at the upper left corner of the image is satisfied. Results using this dataset were presented in [1].

3 Code for MRI simulation

This set of routines provides MRI simulation tools in 2D.

The data-formation model for the parallel magnetic resonance imaging data is

$$m(\mathbf{k}) = \int S(\mathbf{r})\rho(\mathbf{r})\mathrm{e}^{-2\mathrm{i}\pi\mathbf{k}\cdot\mathbf{r}}\mathrm{d}\mathbf{r},$$

where S is the receiving coil sensitivity map, k is the k-space position that evolves during the acquisition, and ρ is the signal that is to be imaged.

During the scan, a finite number M of k-space measurements is performed. With a receiving coil array of N_c channels, the corresponding measurements form a $MN_c \times 1$ vector. The corresponding 2D k-space positions are stored in a $M \times 2$ array k.

Note that both the object under investigation ρ and the coil sensitivity maps are continuous and complex-valued functions of space.

In general, the k-space trajectory is defined for a rectangular field of view (FOV) that includes the support of ρ, meaning that neighboring k-space samples are in average less distant than $1/FOV$. An other crucial parameter of k-space trajectories the highest k-space frequencies sampled that determine the finer resolution (or pixel size) achievable after reconstruction.

GenerateCartesianTraj.m Generates a Cartesian k-space trajectory.

Inputs:

- field of view in meters (can be 2×1 vector)
- resolution (Nyquist distance) in meters (can be 2×1 vector)
- undersampling factor along frequency encoding direction (normalized units, positive, may be lower than one)
- undersampling factor along phase encoding direction (normalized units, positive, may be lower than one)

Output:

- $M \times 2$ array of k-space trajectory points (in rad/meters)

GenerateRadialTraj.m Generates a radial k-space trajectory.

Inputs:

- field of view in meters
- resolution (Nyquist distance) in meters
- undersampling factor along frequency encoding direction (normalized units, positive, may be lower than one)
- undersampling factor in high frequencies(normalized units, positive, may be lower than one)

Output:

- $M \times 2$ array of k-space trajectory points (in rad/meters)

Figure 1: Analytical phantoms and real data reference images: (a) realistic analytical brain phantom, (b) simplified analytical brain phantom, (c) analytical Shepp-Logan phantom, (d) reference image of the *in vivo* brain EPI dataset, and (e) reference image of the *in vivo* brain spiral dataset.

GenerateSpiralTraj.m Generates a spiral k-space trajectory with a method adapted from [6] (the paper is a bit buggy).

Inputs:

- field of view in meters
- resolution (Nyquist distance) in meters
- undersampling factor along frequency encoding direction (normalized units, positive, may be lower than one)
- undersampling factor (normalized units, positive, may be lower than one)
- number of interleaves
- variable density factor (α in [6])
- maximum field gradient amplitude (in T/m)
- maximum field gradient slew rate (in T/m/s)
- resampling the trajectory (true or false)
- analysis option (true or false)

Output:

- $M \times 2$ array of k-space trajectory points (in rad/meters)

GenerateSensitivityMap.m Generates a set of 2D discrete sensitivity maps that are simulated with the Biot-Savart law. The coils are circular, with centers which are equidistant to the origin, and their axis are, by default, uniformly distributed radii. See the illustration of the setup in Figure 2.

Inputs:

- field of view in meters (can be 2×1 vector)
- resolution (pixel size) in meters (can be 2×1 vector)
- number of coils **OR** vector of the desired angles (in radians)
- radius of coils in meters
- distance from the coils centers to the origin in meters

Output:

- $N_x \times N_y \times N_c$ array of complex-valued sensitivity maps with N_x and N_y are the number of pixels in the two directions.

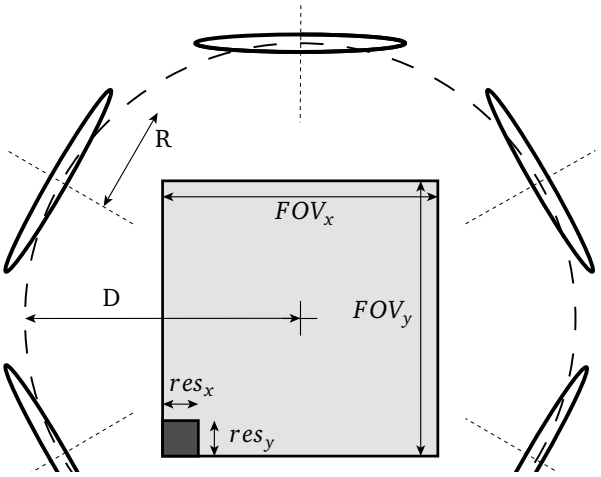

Figure 2: Coil array setup for the generation of sensitivity maps.

SensFitting.m Function that fits a given discrete sensitivity map by a 2D polynomial of a given degree or a linear combination of complex sinusoids. These models are defined in [3]. This function allows parametric continuous representations of the coil sensitivities.

Inputs:

- $N_x \times N_y$ complex-valued sensitivity map array
- string that is either 'polynomial' or 'sinusoidal'
- parameter of the model (polynomial degree or bandwith)
- support (optional): mask for the sensitivity fitting

Outputs:

- structure defining the continuous sensitivity profile
- normalized root mean square fitting error
- signal to error ratio of the fit
- maximal error of fit inside the support
- condition number of the matrix to be inverted (gives an indication of the accuracy of the results)

MRDataAnalytical.m Function that returns the MR data corresponding to the given phantom that is weighted by a sensitivity profile, for the given k-space samples. Using the analytical phantom computations described in [3].

Inputs:

- structure defining an analytical phantom. Predefined phantoms are returned by `DefineBrain.m` (see Figure 1 (a)), `DefineSimpleBrain.m` (see Figure 1 (b)), or `DefineSL.m` (see Figure 1 (c)).
- structure defining a parametrized sensitivity profile as returned by `SensFitting.m`
- $M \times 2$ array of k-space trajectory points (in rad/meters)

Output:

- simulated MRI measurements in a $M \times 1$ complex-valued vector

MRDataRasterized.m A script performing the simulation of an MRI experiment from rasterized image and sensitivities. The discrete non-uniform Fourier transform is performed using FFT and our own MEX implementation of Greengard's fast Gaussian gridding algorithm [4]. See Appendix A for more details. The same code is used for reconstruction. It is highly recommended to use a much finer resolution for simulation than for reconstruction in order to avoid *inverse crime* biases (see [3] for illustration).

Inputs:

- ground truth image (preferably at a much finer resolution than what is used for reconstruction), possibly already weighted by a sensitivity map at the same resolution
- $M \times 2$ array of k-space trajectory points (in rad/meters)
- field of view in meters (can be 2×1 vector)

Outputs:

- simulated MRI measurements in a $M \times 1$ complex-valued vector

SimulateNoise.m A script generating noise to corrupt MRI data. The noise power is computed relatively to the average power in the highest frequencies sampled.

Inputs:

- $M \times 1$ vector of k-space measurements
- $M \times 2$ array of k-space sampling points (in whatever unit)
- SNR for the highest frequencies
- analysis option (true of false)

Outputs:

- simulated noise ($M \times N_c$ complex-valued matrix)

4 Processing MRI Data Prior to Reconstruction

In this section, we present the tools that prepare MRI data to be used with the problem-agnostic reconstruction tools presented in next section.

The MRI inverse problem is posed as follows: $\mathbf{m} = \mathbf{E}\mathbf{x} + \mathbf{n}$, with $\mathbf{m}$ the $MN_c \times 1$ measurements vector, $\mathbf{E}$ the $MN_c \times N$ SENSE encoding matrix that possibly includes the coil sensitivities [7], $\mathbf{x}$ the $N_x N_y \times 1$ vector of the unknown pixel values of the image, and $\mathbf{n}$ a $MN_c \times 1$ noise perturbation vector.

TrajInGridUnits.m A function computing the k-space sampling positions in grid units. Optionally proposes a matrix size for reconstruction (experimental).

Inputs:

- $M \times 2$ array of k-space trajectory points (in rad/meters)
- FOV in meters (can be 2×1 vector)
- 2×1 vector of desired reconstruction matrix size (optional)

Outputs:

- $M \times 2$ array of k-space trajectory points in grid units
- proposed matrix size $[N_x, N_y]$ for reconstruction (2×1 vector)

EstimateCovarianceMatrix.m Estimates the cross-channel covariance matrix out of noise only data. Returns the More-Penrose pseudoinverse of this covariance matrix that is to be used to whiten the noise.

Inputs:

- noise-only measurements ($M' \times N_c$ complex-valued matrix)

Output:

- pseudoinverse of the noise covariance matrix: $N_c \times N_c$ Hermitian-symmetric, positive-definite, and complex-valued matrix.

Prepare4Recon.m This function prepares data for reconstruction. The idea is that the back-projected image $\mathbf{a} = \mathbf{E}^{\mathrm{H}}\mathbf{V}\mathbf{m}$, where $\mathbf{V}$ is the pseudoinverse of the noise correlation matrix, and the Hermitian symmetric and positive-definite matrix $\mathbf{A} = \mathbf{E}^{\mathrm{H}}\mathbf{V}\mathbf{E}$ are sufficient to perform reconstruction. Indeed, for reconstruction, one generally wants to find an image $\mathbf{x}$ that keeps the following quantity as small as possible:

$$||\mathbf{m} - \mathbf{E}\mathbf{x}||_{\mathbf{V}}^2 = ||\mathbf{m}||_{\mathbf{V}}^2 + ||\mathbf{x}||_{\mathbf{A}}^2 - 2\mathrm{Re}\langle \mathbf{a}, \mathbf{x} \rangle.$$

Inputs:

- $M \times 1$ vector of k-space measurements $\mathbf{m}$
- $M \times 2$ array of k-space sampling points (in grid units)
- $N_x \times N_y \times N_c$ sensitivity maps array
- support to be used for reconstruction

- $N_c \times N_c$ noise-whitening matrix

Outputs:

- back-projected measurements image **a** (complex-valued $N_x \times N_y$ matrix)
- function handle that performs the linear operation $\mathbf{y} = \mathbf{Ax}$
- real positive valued $N_x \times N_y$ matrix of the root sum of square sensitivities (used for preconditioning).

5 Generic Reconstruction Methods

The reconstruction methods presented in this section are designed to achieve image reconstruction from linear inverse problems. They rely on a image **a** and a matrix **A**, ensuring a problem-agnostic reconstruction as long as the reconstructed image **x** is required to maintain the quantity $||\mathbf{x}||_{\mathbf{A}}^2 - 2\text{Re}\langle \mathbf{a}, \mathbf{x} \rangle$ as small as possible.

CGRecon.m This function implements the Conjugate Gradient method [7] to solve the linear system $\mathbf{Ax} = \mathbf{a}$, with **A** a Hermitian symmetric and positive-definite matrix. A typical Tikhonov regularization for MRI would correspond to $\mathbf{a} = \mathbf{E}^{\text{H}}\mathbf{Vm}$ and $\mathbf{A} = \mathbf{E}^{\text{H}}\mathbf{EE} + \lambda\mathbf{I}$.

Optionally, a real-valued image representing a diagonal matrix **P** can be used for preconditioning such that the following system is solved $\mathbf{Mu} = \mathbf{b}$, with $\mathbf{M} = \mathbf{PAP}$, $\mathbf{x} = \mathbf{Pu}$ and $\mathbf{b} = \mathbf{Pa}$.

Inputs:

- back-projected measurements $N_x \times N_y$ image **a**
- function handle performing $\mathbf{y} = \mathbf{Ax}$
- $N_x \times N_y$ image used as a starting point
- number of iterations to be performed
- $N_x \times N_y$ real-valued array used for preconditioning

Outputs:

- reconstructed image ($N_x \times N_y$ complex-valued matrix)
- computation time vector
- residual vector

TVRecon.m This function implements Total Variation penalized reconstruction trying to solve the linear system $\mathbf{Ax} = \mathbf{a}$, with **A** a Hermitian symmetric and positive-definite matrix. For MRI reconstruction, one would typically impose $\mathbf{a} = \mathbf{E}^{\text{H}}\mathbf{Vm}$ and $\mathbf{A} = \mathbf{E}^{\text{H}}\mathbf{VE}$.

Optionally, a real-valued image representing a diagonal matrix **P** can be used for preconditioning such that the following system is solved $\mathbf{Mu} = \mathbf{b}$, with $\mathbf{M} = \mathbf{PAP}$, $\mathbf{x} = \mathbf{Pu}$ and $\mathbf{b} = \mathbf{Pa}$.

The Total Variation term considered is the ℓ_1 norm of the pixel-wise ℓ_2 norm of the image gradient (isotropic TV). The algorithm used to perform the reconstruction is the Iteratively Reweighted Least-Squares algorithm [8].

Inputs:

- back-projected measurements $N_x \times N_y$ image **a**
- function handle performing $\mathbf{y} = \mathbf{Ax}$
- regularization parameter
- $N_x \times N_y$ image used as a starting point
- number of iterations of IRLS
- number of iterations for internal CG
- $N_x \times N_y$ real-valued array used for preconditioning

Outputs:

- reconstructed image ($N_x \times N_y$ complex-valued matrix)
- computation time vector
- residual vector

6 Wavelet-based reconstruction

In this section we present the functions that implement the fast wavelet-based reconstruction described in [1]. As in the previous section, the reconstruction is designed to be problem-agnostic.

Two families of transforms are available: discrete wavelet transforms (used in JPEG2000) and block discrete cosine transforms (used in JPEG).

@BlockDCT Defining a block discrete cosine transform as an object that mimics the corresponding matrix using overloaded functions.

Input:

- size of the blocks

Output:

- BlockDCT object

@DWT Defining a discrete wavelet transform that mimics the corresponding matrix using overloaded functions.

Inputs:

- number of decomposition levels
- size of the images to be transformed (2×1 vector)
- use decimation (true or false). If decimated the transform is orthonormal. If undecimated, the transform is shift-invariant
- wavelet family ('haar', 'daub1', 'daub2', 'daub4', 'sym4', 'spline1', 'vspline2', 'spline3', 'spline4', 'spline5', 'spline6', 'sym8', 'sinc', 'shannon', 'v97', '9/7', or 'espline')

Output:

- DWT object

PowerIteration.m Computes the greatest absolute eigenvalue α of a square matrix $\mathbf{M}$ such that for all $\mathbf{x}$, $||\mathbf{Mx}||_2 \leq \alpha||\mathbf{x}||_2$. If $\mathbf{M}$ is positive definite and expressed as $\mathbf{M} = \mathbf{A}^{\mathrm{H}}\mathbf{A}$, we have, for all $\mathbf{x}$, $||\mathbf{Ax}||^2 \leq \alpha||\mathbf{x}||^2$, and α can be seen as the Lipschitz constant of the function $f(\mathbf{x}) = \mathbf{Ax}$.

Inputs:

- a matrix or function handle performing $\mathbf{y} = \mathbf{Mx}$
- an initial non-all-zero vector or image

Outputs:

- the greatest absolute eigenvalue
- the associated eigen-vector/image

PowerIterationWav.m Computes the vector of wavelet subband bounds $\boldsymbol{\alpha}$ of a square matrix $\mathbf{M}$ such that for all set of wavelet coefficients $\mathbf{w}$, $||\mathbf{Mw}||_2 <= ||\mathrm{diag}(\boldsymbol{\alpha})\mathbf{w}||_2$. See [1] for more details.

Inputs:

- a function handle performing $\mathbf{w}' = \mathbf{Mw}$
- an initial non-all-zero set of wavelet coefficients

Outputs:

- the vector of wavelet subband bounds

ReconWavFISTA.m Performs wavelet-regularized reconstruction [1] trying to solve the linear system $\mathbf{Ax} = \mathbf{a}$, with $\mathbf{A}$ a Hermitian symmetric and positive-definite matrix. For MRI reconstruction, one would typically impose $\mathbf{a} = \mathbf{E}^{\mathrm{H}}\mathbf{Vm}$ and $\mathbf{A} = \mathbf{E}^{\mathrm{H}}\mathbf{VE}$.

The DWT or DCT object W provided for regularization should perform an orthogonal transform.

See the classes DWT, DCT and WAVELET.

Inputs:

- back-projected measurements $N_x \times N_y$ image $\mathbf{a}$
- function handle performing $\mathbf{y} = \mathbf{Ax}$
- regularization parameter
- DWT or DCT object
- scalar that is larger than the greatest eigenvalue of $\mathbf{A}$ (FISTA), **OR** vector of scalars for each wavelet-subband (FWISTA [1])
- $N_x \times N_y$ image used as a starting point
- number of iterations
- use random shifting technique (true or false)

Outputs:

- reconstructed image ($N_x \times N_y$ complex-valued matrix)
- time vector
- residual vector

7 Test Scripts

Several test scripts are provided to check the different modules present in this package. They are included in the folder `test/`. The function `PerformTests.m` runs the called test or all the tests (default with no argument).

TestGeneratingTraj.m Generating the different kinds of trajectories and scaling them to grid units for visualization.

TestGeneratingSens.m Computing the sensitivity maps for an array of receiving coils. Fitting the complex maps with two parametric models and comparing the fitting errors.

TestNUFTaccuracy.m Testing the accuracy of several implementations of the matrix-vector multiplications $\mathbf{m} = \mathbf{Ex}$, $\mathbf{y} = \mathbf{E}^{\mathrm{H}}\mathbf{m}$, and $\mathbf{z} = \mathbf{Mx}$, where $\mathbf{E}$ is a particular non uniform discrete Fourier transform matrix, $\mathbf{E}^{\mathrm{H}}$ is its Hermitian transpose and $\mathbf{M} = \mathbf{E}^{\mathrm{H}}\mathbf{E}$ (with a specific implementation). The reference implementation does not use gridding. We check the consistency of the operations using the property

$$||\mathbf{Ex}||^2 = \langle \mathbf{E}^{\mathrm{H}}\mathbf{Ex}, \mathbf{x} \rangle = \langle \mathbf{Mx}, \mathbf{x} \rangle \geq 0.$$

TestNUFTspeed.m Testing the speed of several implementations of the matrix-vector multiplications $\mathbf{m} = \mathbf{Ex}$, $\mathbf{y} = \mathbf{E}^{\mathrm{H}}\mathbf{m}$, and $\mathbf{z} = \mathbf{Mx}$, where $\mathbf{F}$ is a particular non uniform discrete Fourier transform matrix, $\mathbf{E}^{\mathrm{H}}$ is its Hermitian transpose and $\mathbf{M} = \mathbf{E}^{\mathrm{H}}\mathbf{E}$ (with a specific implementation).

TestNoise.m A random correlation matrix is made. A noise draw is correlated accordingly. The evaluation of the correlation matrix out of the noise data is checked.

TestSimulation.m In a typical single channel MRI setting, the consistency of two different MRI data simulation methods is checked in both k-space and reconstructed image domain.

TestDWT.m Checking the implementation of the DWT and WAVELET classes. They mimic discrete wavelet transform matrices and, respectively, vectors of wavelet coefficients thanks to overloaded functions.

TestBlockDCT.m Checking the implementation of the BlockDCT class. It mimics the block discrete cosine transform matrices thanks to overloaded functions.

8 Under the Hood

In general, the functions that don't need to be accessed directly by the end-user of this package have been hidden in the `private/` folder.

The scripts implementing the different validation tests are included in the `test/` folder.

The folders `@DWT/`, `@BlockDCT/`, and `@wavelet/` implement the classes for discrete wavelet transform objects, block discrete

cosine transform objects, and wavelet coefficients objects, respectively. They contain the classes definitions and the overloaded operations making these objects behave like matrices and vectors. The functions implementing wavelets related operations, are included in the folder `waveletstuff/` that must be included in your Matlab path in order to enable wavelet-based reconstructions.

The folder `scannerdata/`, if it is included in this package, contains the real scanner data. Given the size of the data, this folder will probably be distributed separately from the rest of the package.

A The Gaussian Gridding Method

A.1 Principle

In this section, we describe the principle of Gaussian gridding in 1D, inspired by [4]. Note that our conventions slightly differ.

The two operations to be performed are the sums

$$m_j = \hat{x}(2\pi k_j/N) = \sum_{n=0}^{N-1} x_n \mathrm{e}^{-2\mathrm{i}\pi n k_j/N} \tag{1}$$

and its adjoint operation

$$y_n = \sum_j m_j \mathrm{e}^{2\mathrm{i}\pi n k_j/N} \tag{2}$$

for a set of arbitrary k-space points $k_j \in [-N/2, N/2[$ and for sequences x_n and y_n of N samples ($0 \le n < N$).

When considering N samples m_j, the summations (1) and (2) have, *a priori*, a complexity $\mathcal{O}(N^2)$. Gridding achieves the computations with complexity $\mathcal{O}(N \log N)$, similar to what is possible when the k_j lie on a regular grid using FFT.

Equation (2) is interpreted as the set of the Fourier coefficients of the 2π-periodic function $f(t) = 2\pi \sum_j m_j \delta(t - 2\pi k_j)$ corresponding to $0 \le n < N$.

The principle of gridding is to smooth out $f(t)$ using a convolution kernel. In our case, it is the Gaussian function $g_\tau(t) = \sum_i \mathrm{e}^{(t-2\pi i)^2/(4\tau)}$ and we end up with the function

$$f_\tau(t) = f * g_\tau(t) = \sum_j m_j g_\tau(t - 2\pi k_j)$$

which is 2π-periodic as well. The Fourier coefficients of f_τ for $0 \le n < N$ are given by

$$c_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f_\tau(t) \mathrm{e}^{-\mathrm{i}nt} \mathrm{d}t = y_n \sqrt{\frac{\tau}{\pi}} \mathrm{e}^{-\tau n^2}. \tag{3}$$

Since f_τ is smoothed by the Gaussian kernel, the Fourier coefficients c_n decay rapidly. For $0 \le n < N$ and provided that $M \gg N$, the M-periodized version $\sum_{p\in\mathbb{Z}} c_{n+pM}$ approximates c_n well. Thus, one obtains a good approximation of the coefficients c_n using the DFT coefficients of a finely discretized version of f_τ

$$c_n \approx \frac{1}{M} \sum_{p=0}^{M-1} f_\tau(2\pi p/M) \mathrm{e}^{-2\mathrm{i}\pi n p/M}, \text{ for } M \gg N. \tag{4}$$

Finally, the computation of (2) is performed with the steps

1. Compute the samples $C_p = \sum_j m_j g_\tau(2\pi(p/M - k_j))$ of f_τ,
2. Compute the inverse FFT of these coefficients,
3. Get c_n as the first N coefficients,
4. Set $y_n = c_n \sqrt{\frac{\pi}{\tau}} \mathrm{e}^{\tau n^2}$.

The adjoint operation, corresponding to the summation (1) is naturally performed with the steps:

1. Set $c_n = x_n \sqrt{\frac{\pi}{\tau}} \mathrm{e}^{\tau n^2}$,
2. Zero-pad the coefficients c_n to length M,
3. Perform the FFT to obtain coefficients C_p,
4. Set $m_j = \sum_{p=0}^{M-1} C_p g_\tau(2\pi(p/M - k_j))$.

A specificity of Greengard's Gaussian gridding is to consider centered discrete coordinates, namely $-\frac{N}{2}\rfloor \le n \le \frac{N+1}{2}\rfloor$. For a given set of parameters, this choice yields an improved accuracy. These computations are related to the algorithm presented above because

$$m_j = \hat{x}(2\pi k_j/N) = \sum_{n'=-\frac{N}{2}}^{\frac{N+1}{2}\rfloor} \left(x_n \mathrm{e}^{-2\mathrm{i}\pi \frac{N}{2}\rfloor k_j/N} \right) \mathrm{e}^{-2\mathrm{i}\pi n' k_j/N} \tag{5}$$

and

$$y_{n'} = \mathrm{e}^{2\mathrm{i}\pi \frac{N}{2}\rfloor} \left(\sum_j m_j \mathrm{e}^{2\mathrm{i}\pi n' k_j/N} \right), \text{ with } -\frac{N}{2}\rfloor \le n' \le \frac{N+1}{2}\rfloor. \tag{6}$$

A.2 Choice of the parameters

In practice, for each k-space sample, the Gaussian kernel convolution is limited to the nearest points, controlled by M_{sp}, the number of neighbors in each direction. In [4], it is reported that the choice $M = 4N$, $M_{sp} = 12$, and $\tau = \pi M_{sp}/(M(M - N/2))$, leads to an accuracy of about 1e-14. We suggest these settings as the default for accurate computations. For a faster version of gridding, we suggest $M = 3N$, $M_{sp} = 8$, and $\tau = \pi M_{sp}/(M(M - N/2))$.

References

[1] M. Guerquin-Kern, M. Häberlin, K. P. Pruessmann, and M. Unser, "A fast wavelet-based reconstruction method for magnetic resonance imaging," *IEEE Transactions on Medical Imaging*, vol. 30, no. 9, pp. 1649–1660, September 2011.

[2] M. Guerquin-Kern, J.-C. Baritaux, and M. Unser, "Efficient image reconstruction under sparsity constraints with application to MRI and bioluminescence tomography," in *Proceedings of the Thirty-Sixth IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'11)*, Prague, Czech Republic, May 22-27 2011, pp. 5760–5763.

[3] M. Guerquin-Kern, L. Lejeune, K. P. Pruessmann, and M. Unser, "Realistic analytical phantoms for parallel magnetic resonance imaging," *IEEE Transactions on Medical Imaging*, vol. 31, no. 3, pp. 626–636, March 2012.

[4] L. Greengard and J.-Y. Lee, "Accelerating the nonuniform fast Fourier transform," *SIAM Review*, vol. 46, no. 3, pp. 443-454, 2004.

[5] C. Barmet, N. De Zanche, B. J. Wilm, and K. P. Pruessmann, "A transmit/receive system for magnetic field monitoring of *in vivo* MRI," *Magnetic Resonance in Medicine*, vol. 62, no. 1, pp. 269–276, July 2009.

[6] D.-H. Kim, E. Adalsteinsson, and D. M. Spielman, "Simple analytic variable density spiral design," *Magnetic Resonance in Medicine*, vol. 50, no. 1, pp. 214–219, 2003.

[7] K. P. Pruessmann, M. Weiger, P. Börnert, and P. Boesiger, "Advances in sensitivity encoding with arbitrary k-space trajectories," *Magnetic Resonance in Medicine*, vol. 46, no. 4, pp. 638–651, 2001.

[8] B. Wohlberg and P. Rodríguez, "An iteratively reweighted norm algorithm for minimization of total variation functionals," *IEEE Signal Processing Letters*, vol. 14, no. 12, pp. 948–951, Dec. 2007.