MRI algorithm outline

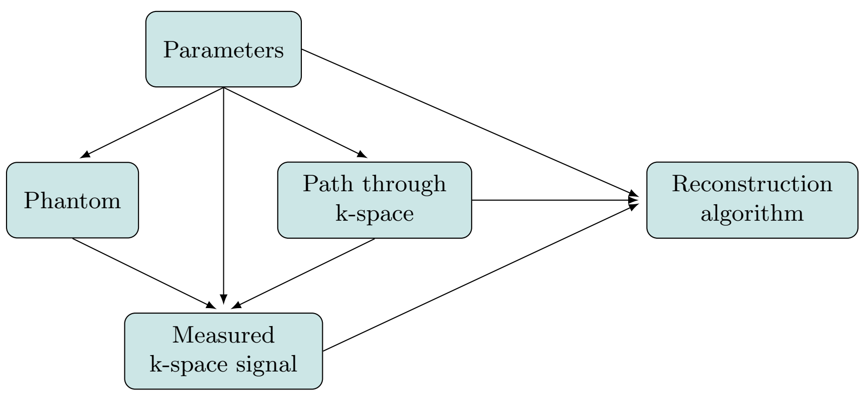
U. Chicago MRI Research Center

December 12, 2019

Correspondence:
tyo8teasley@uchicago.edu, *rina@uchicago.edu*

1 Code outline

There are five main pieces of the code, pictured in the following dependency diagram:



1.1 Parameters

The parameters describe the settings of the scanner and reconstruction, and are stored in the structures `AcqPars` and `ReconPars`, respectively. `AcqPars` has the following fields:

- The dimensions and resolution of the spatial domain are expressed in three ways (only two of which need to be specified):
  - `AcqPars.nx`, `AcqPars.ny`, `AcqPars.nz` are positive integers specifying the dimensions of the reconstructed images, and of the corresponding Cartesian grid in k-space where we will acquire measurements.
  - `AcqPars.FOV` is a 3-dimensional vector encoding the size of the field of view in the spatial domain, in millimeters.
  - `AcqPars.Resolution` is a 3-dimensional vector encoding the size of each voxel in the reconstruction, in millimeters, which is equal to (`PhysSize(1)/nx`)×(`PhysSize(2)/ny`)×(`PhysSize(3)/nz`).

  If any two are specified, we compute the third one; if all three are specified and do not agree, we overwrite `AcqPars.Resolution`.

- `AcqPars.stddev` is a `nx`×`ny`×`nz` array of nonnegative values specifying the standard deviation of the noise at each point in k-space
- `AcqPars.FlipAngle` is the flip angle of the scanner, with value in $[0, 2\pi]$
- `AcqPars.TR`, `AcqPars.TE` encode timing of the scanner (units = milliseconds). `TR` is the repetition time (milliseconds between each line acquisition) and `TE` is the echo time (milliseconds from center to edge of k-space and back along readout dimension)
- `AcqPars.randomseed` is the seed for the random number generator to produce the noise of the measurements in k-space.
- `AcqPars.acqtimeres` is the time resolution (in milliseconds) of the signal acquisition. When computing the measured k-space signal, we update the phantom at time resolution `acqtimeres` (and take a linear interpolation in between update times).
- The dimensions and resolution of the temporal domain are expressed in three ways (only two of which need to be specified):
  - `AcqPars.nscan`, the total number of passes through k-space
  - `AcqPars.totalscantime`, the duration of the scan (in milliseconds)
  - `AcqPars.onescantime`, the time duration of a single pass through k-space (in milliseconds), which is equal to `totalscantime`÷`nscan`

  If any two are specified, we compute the third one; if all three are specified and do not agree, we overwrite `onescantime`. (Note that `onescantime` should, for real data, be a function of `TR`, `TE`, and the dimensions, but in the code is allowed to take an arbitrary value.)

`ReconPars` has the following fields:

- The reconstruction algorithm is based on a linear system, which is solved via preconditioned conjugated gradient descent, using Matlab's `pcg` function. We use two convergence parameters:
  - `ReconPars.convergethresh` is a convergence threshold parameter, e.g., 10^{-8}.
  - `ReconPars.maxiter` is the maximum number of iterations allowed, e.g., 1000.
- `ReconPars.smoothing` is a smoothing parameter, which takes a small positive value, e.g., 10^{-5}. It's added to the eigenvalues of the smoothing-over-time matrix in order to ensure this matrix is invertible.
- `ReconPars.weights` is a `nx`×`ny`×`nz` of positive weights. A higher weight corresponds to a voxel where we expect the signal to be more smooth over time.
- The time resolution of the reconstruction is expressed by two parameters:
  - `ReconPars.nimage` is the number of images in the reconstruction
  - `ReconPars.recontimeres` is the time resolution of the reconstruction (in milliseconds), i.e., one image is produced every `recontimeres` milliseconds. It is equal to `scantime`÷`nimage`.

  Only one of these needs to be specified. If both are specified and do not agree, then we overwrite `recontimeres`. Note that the problem is underdetermined only when `recontimeres`<`onescantime` (equivalently, `nimage`>`nscan`), which is assumed in the implementation.

1.2 Phantom

The phantom is not constructed directly, but instead is specified via a function that can return the 3D phantom at any specified time, along with any parameters needed for calling this function.

- `PhantomPars` holds any parameters or objects needed for calling the phantom evaluation function. If no parameters are needed, `PhantomPars` is an empty variable.
- `PhantomEvalFn` is a function handle for evaluating the parameter at a time (between 0 and `totalscantime`). The times are measured in milliseconds, with zero being a reference time at the start of the scan. The function returns a real-valued array of dimension `nx`×`ny`×`nz` containing the signal at each point in the spatial grid, at time `time`.
  The function is called with the command `PhantomEvalFn(PhantomPars,AcqPars,time)`.
  - Workflow: in order to construct a particular phantom, we create a custom function:
    `function PhantomAtTime = MyPhantomFunction(PhantomPars,AcqPars,time)`
    stored in `MyPhantomFunction.m`. Then, we store its handle with the command:
    `PhantomEvalFn=@MyPhantomFunction;`
  - If the function `PhantomEvalFn` requires access to any helper functions, libraries, etc, this can be achieved by adding a path name to the parameter structure. For example, the first line of the `PhantomEvalFn` function definition might be
    `addpath(genpath(PhantomPars.filepathname))`
    where set `PhantomPars.filepathname` is defined as a string giving the necessary path.

1.3 Path through k-space

The path through k-space is specified by a `nx`×`ny`×`nz`×`nscan` array. The entries of this array specify the time (in milliseconds, since the start of the scan) when each point in k-space is reached, during each pass through k-space.

1.4 Measured signal

The measured signal is a `nx`×`ny`×`nz`×`nscan` complex-valued array called `KspaceSignal`, storing the noisy value measured at each point in k-space, during each pass of the scanner.

- The array `KspaceSignal` is created by the function `GenerateKspaceSignal`, of the format:
  `function KspaceSignal = GenerateKspaceSignal(PhantomEvalFn,PhantomPars,Path,AcqPars)`.
  To produce the k-space measurement `KspaceSignal(ix,iy,iz,iscan)`, the function performs the following steps:
  - This value in k-space is measured at time `Path(ix,iy,iz,iscan)`. Since the phantom is updated only once every `AcqPars.acqtimeres` milliseconds, find update times $t_{\mathrm{pre}}, t_{\mathrm{post}}$ such that
    $t_{\mathrm{pre}} \leq \texttt{Path(ix,iy,iz,iscan)} \leq t_{\mathrm{post}}$,
    i.e., the update times immediately before and after the measurement time. Find the value $0 \leq c \leq 1$ so that $\texttt{Path(ix,iy,iz,iscan)} = c \cdot t_{\mathrm{pre}} + (1-c) \cdot t_{\mathrm{post}}$.
  - Let `kspace_pre` be the Fourier transform of the phantom evaluated at time t_{pre}, and `kspace_post` at time t_{post}. Then, using a linear interpolation, the measured signal is:
    $c \cdot \texttt{kspace\_pre(ix,iy,iz)} + (1-c) \cdot \texttt{kspace\_post(ix,iy,iz)} + \text{noise}$,
    where the noise is a complex Gaussian with standard deviation `AcqPars.stddev(ix,iy,iz)`.

1.5 Reconstruction algorithm

The reconstructed signal is a `nx`×`ny`×`nz`×`ntime` complex-valued array called `EstSignal`, where the number of time points is `ntime` = `nscan`·`nhighres`.

- The array `EstSignal` is computed with the function `ReconAlg`, of the format:
  `function EstSignal = ReconAlg(KspaceSignal,Path,AcqPars,ReconPars).`

The reconstruction algorithm is described below.

2 Reconstruction algorithm

Next we give the details of the reconstruction algorithm implemented in the function `ReconAlg`. A glossary for translating between the algorithm and the code outline:

$n =$ `nx·ny·nz`, the total number of voxels in the discretized grid
$T =$ the number of timepoints in the reconstructed image, `nimage`
$X =$ the phantom produced by `PhantomEvalFn`
$Y =$ the measured k-space signal `KspaceSignal`
$\widehat{X} =$ the estimated signal `EstSignal` computed by the function `ReconAlg`
$W = \mathrm{diag}(w_1, \ldots, w_n)$ where $w_1, \ldots, w_n$ are specified by `weights`
$\lambda =$ the smoothing parameter `smoothing`
$\Omega =$ specifies which points in k-space are observed at which time, as determined by `Path`

2.1 Measurement model

Let $n = n_x \cdot n_y \cdot n_z$. At each time $t = 1, \ldots, T$, the signal is given by $X_t \in \mathbb{C}^n$, which contains the true $n_x \times n_y \times n_z$ discretized signal reshaped into a vector. We will write $X \in \mathbb{C}^{n\times T}$ to be the matrix with t-th column X_t.

Let $\Omega_t \subset \{1, \ldots, n\}$ index the points in k-space where we take measurements at time t, and let

$$\Omega = \cup_{t=1}^{T} \big(\Omega_t \times \{t\}\big) \subset \{1, \ldots, n\} \times \{1, \ldots, T\}$$

index the observed points in k-space at their observed times. While the number of timepoints T is higher than the total number of scans, it is still an approximation to the continuous-time measurements taken by the scanner. To assign each measurement in k-space to a time $t = 1, \ldots, T$, we simply divide the total scan time into T time bins of equal length, and then all measurements in k-space acquired during time bin t are assigned to Ω_t.

Next, let $Y \in \mathbb{C}^{n\times T}$ be a matrix containing our measured entries, with zeros elsewhere. Specifically, in the t-th column, we have

$$(Y_t)_{\Omega_t} = [\mathcal{F}X_t]_{\Omega_t} + \text{noise}, \quad (Y_t)_{(\Omega_t)^c} = 0,$$

where $\mathcal{F}$ is the 3D discrete Fourier transform (rearranged into a $n \times n$ matrix, so that it acts on vectorized 3D signals, i.e., $\mathcal{F}$ is a map from $\mathbb{C}^n$ to $\mathbb{C}^n$). The entries of the noise are independent complex normals with variance corresponding to the location in k-space (variance is highest near the center of k-space).

2.2 Smoothed optimization

We will solve a smoothed optimization problem:

$$\min \left\{ \langle W, XDX^* \rangle : (\mathcal{F}X)_\Omega = Y_\Omega \right\}, \tag{1}$$

where $D \in \mathbb{C}^{T\times T}$ and $W \in \mathbb{C}^{d\times d}$ are positive definite matrices chosen such that D induces smoothness over time while W controls the penalty over the spatial domain.

For the matrix W, we will use a diagonal matrix $W = \mathrm{diag}(w_1, \ldots, w_n)$, where $w_i > 0$ is the smoothness penalty acting on voxel i (larger w_i corresponds to enforcing more smoothness on that voxel's trajectory over time).

For the matrix D, we will use

$$D = \begin{pmatrix} 1 & -2 & 1 & 0 & \ldots & 0 & 0 & 0 \\ -2 & 5 & -4 & 1 & \ldots & 0 & 0 & 0 \\ 1 & -4 & 6 & -4 & \ldots & 0 & 0 & 0 \\ 0 & 1 & -4 & 6 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & 6 & -4 & 1 \\ 0 & 0 & 0 & 0 & \ldots & -4 & 5 & -2 \\ 0 & 0 & 0 & 0 & \ldots & 1 & -2 & 1 \end{pmatrix} + \lambda \mathbf{I}_T.$$

The first term penalizes the squared magnitude of the discretized second derivative, while $\lambda \mathbf{I}_T$ ensures that D is invertible (here $\mathbf{I}_T$ is the $T \times T$ identity matrix, while $\lambda > 0$ is some small smoothing parameter).

We can see that (1) is a simple optimization problem, minimizing a convex quadratic objective subject to some linear constraints.

2.2.1 Solving for the minimizer

With some rearranging, we can find a closed-form solution to (1). First we rearrange into vector form, and equivalently rewrite the optimization problem (1) as:

$$\min \left\{ \mathrm{vec}(X)^* (\overline{D} \otimes W) \mathrm{vec}(X) : \left[(\mathbf{I}_T \otimes \mathcal{F}) \cdot \mathrm{vec}(X) \right]_\Omega = \left[\mathrm{vec}(Y) \right]_\Omega \right\}, \tag{2}$$

where, abusing notation, we are now treating Ω as a subset of indices $\{1, \ldots, nT\}$ (i.e., the indices after vectorizing our $n \times T$ matrices). The solution is then given by

$$\mathrm{vec}(\widehat{X}) = \left[\overline{D}^{-1} \otimes (W^{-1} \mathcal{F}^*) \right]_{*,\Omega} \cdot \left(\left[\overline{D}^{-1} \otimes (\mathcal{F} W^{-1} \mathcal{F}^*) \right]_{\Omega,\Omega} \right)^{-1} \cdot \left[\mathrm{vec}(Y) \right]_\Omega. \tag{3}$$

In terms of computation cost, the main step is solving the $|\Omega| \times |\Omega|$ linear system. Note that, in the notation of our code outline, $|\Omega|$ is given by

$$|\Omega| = \mathtt{nx} \cdot \mathtt{ny} \cdot \mathtt{nz} \cdot \mathtt{nscan},$$

i.e., it's the total number of measurements acquired.

Now we verify that (3) solves the optimization problem (2). First we check feasibility:

$$\begin{aligned} \left[(\mathbf{I}_T \otimes \mathcal{F}) \cdot \mathrm{vec}(\widehat{X}) \right]_\Omega &= \left[\left[\overline{D}^{-1} \otimes (\mathcal{F} W^{-1} \mathcal{F}^*) \right]_{*,\Omega} \cdot \left(\left[\overline{D}^{-1} \otimes (\mathcal{F} W^{-1} \mathcal{F}^*) \right]_{\Omega,\Omega} \right)^{-1} \cdot \left[\mathrm{vec}(Y) \right]_\Omega \right]_\Omega \\ &= \left[\overline{D}^{-1} \otimes (\mathcal{F} W^{-1} \mathcal{F}^*) \right]_{\Omega,\Omega} \cdot \left(\left[\overline{D}^{-1} \otimes (\mathcal{F} W^{-1} \mathcal{F}^*) \right]_{\Omega,\Omega} \right)^{-1} \cdot \left[\mathrm{vec}(Y) \right]_\Omega \\ &= \left[\mathrm{vec}(Y) \right]_\Omega. \end{aligned}$$

Next we check first-order optimality conditions—we need to see that $(\overline{D} \otimes W)\mathrm{vec}(\widehat{X})$ (the gradient of the objective function) lies in the span of $[\mathbf{I}_T \otimes \mathcal{F}^*]_{*,\Omega}$ (the gradient of the constraints):

$$\begin{aligned} (\overline{D} \otimes W)\mathrm{vec}(\widehat{X}) &= (\overline{D} \otimes W) \cdot \left[\overline{D}^{-1} \otimes (W^{-1} \mathcal{F}^*) \right]_{*,\Omega} \cdot \left(\left[\overline{D}^{-1} \otimes (\mathcal{F} W^{-1} \mathcal{F}^*) \right]_{\Omega,\Omega} \right)^{-1} \cdot \left[\mathrm{vec}(Y) \right]_\Omega \\ &= \left[(\overline{D} \otimes W) \cdot (\overline{D}^{-1} \otimes (W^{-1} \mathcal{F}^*)) \right]_{*,\Omega} \cdot \left(\left[\overline{D}^{-1} \otimes (\mathcal{F} W^{-1} \mathcal{F}^*) \right]_{\Omega,\Omega} \right)^{-1} \cdot \left[\mathrm{vec}(Y) \right]_\Omega \\ &= \left[\mathbf{I}_T \otimes \mathcal{F}^* \right]_{*,\Omega} \cdot \left(\left[\overline{D}^{-1} \otimes (\mathcal{F} W^{-1} \mathcal{F}^*) \right]_{\Omega,\Omega} \right)^{-1} \cdot \left[\mathrm{vec}(Y) \right]_\Omega, \end{aligned}$$

as desired. This proves that (3) is optimal. (Since D and W are positive definite, the solution is also unique.)

2.2.2 Special case: constant weights

If $W = \text{diag}(w_1, \ldots, w_n)$ where the weights $w_1 = \cdots = w_n$ are constant across the n voxels of the image, this is a special case that can be solved much more efficiently. The solution is given by

$$\widehat{X} = \mathcal{F}^* \widetilde{Y},$$

where $\widetilde{Y}$ is the smoothed signal in k-space. Specifically, for each voxel $i = 1, \ldots, n$, let $\Omega_{(i)} \subset \{1, \ldots, T\}$ be the time points when voxel i is measured, and let $Y_{(i)}$ and $\widetilde{Y}_{(i)}$ denote the i-th row of the matrix Y, expressed as column vector (i.e., the measurements at voxel i, and zeros at the unmeasured time points). We define the smoothed k-space signal with rows $\widetilde{Y}_{(i)}$,

$$\widetilde{Y}_{(i)} = \left[\overline{D}^{-1}\right]_{*,\Omega_{(i)}} \cdot \left(\left[\overline{D}^{-1}\right]_{\Omega_{(i)},\Omega_{(i)}}\right)^{-1} \cdot \left[Y_{(i)}\right]_{\Omega_{(i)}}. \tag{4}$$

Note that

$$\left[\widetilde{Y}_{(i)}\right]_{\Omega_{(i)}} = \left[\overline{D}^{-1}\right]_{\Omega_{(i)},\Omega_{(i)}} \cdot \left(\left[\overline{D}^{-1}\right]_{\Omega_{(i)},\Omega_{(i)}}\right)^{-1} \cdot \left[Y_{(i)}\right]_{\Omega_{(i)}} = \left[Y_{(i)}\right]_{\Omega_{(i)}},$$

that is, the measured values at the observed voxels stay the same.

Now we verify that $\widehat{X}$ is the right solution. Referring back to equation (3) for the general case, note that in this special case we have $W^{-1} = w_1^{-1}\mathbf{I}_n$, and so $\mathcal{F}W^{-1}\mathcal{F}^* = w_1^{-1}\mathcal{F}\mathcal{F}^* = w_1^{-1}\mathbf{I}_n$. Therefore, the solution is given by

$$\text{vec}(\widehat{X}) = \left[\overline{D}^{-1} \otimes w_1^{-1}\mathcal{F}^*\right]_{*,\Omega} \cdot \left(\left[\overline{D}^{-1} \otimes w_1^{-1}\mathbf{I}_n\right]_{\Omega,\Omega}\right)^{-1} \cdot \left[\text{vec}(Y)\right]_{\Omega}.$$

This can be rearranged to

$$\widehat{X} = \mathcal{F}^*\widetilde{Y} \text{ where } \text{vec}(\widetilde{Y}) = \left[\overline{D}^{-1} \otimes \mathbf{I}_n\right]_{*,\Omega} \cdot \left(\left[\overline{D}^{-1} \otimes \mathbf{I}_n\right]_{\Omega,\Omega}\right)^{-1} \cdot \left[\text{vec}(Y)\right]_{\Omega},$$

and we can verify that this definition of $\widetilde{Y}$ is equivalent to the solution given above in (4).