BART Documentation

Release 0.3.01

BART Dev

December 11, 2016

Contents

Contents:

CHAPTER 1

Introduction

1.1 Overview

BART is a free and open-source image-reconstruction framework for Computational Magnetic Resonance Imaging. It consists of a programming library and a toolbox of command-line programs.

This documentation is meant to be a guide for BART usage and development. Since this is hand-written rather than machine generated, some code snippets might be outdated by the time you read them. However, the general idea should remain the same.

If you have any questions about the implementation details or find errors in this documentation, please send them to our mailing list https://lists.eecs.berkeley.edu/sympa/info/mrirecon.

1.2 List of Features

- Basic features
  - support for Linux, Mac OS X, and Windows (with Cygwin)
  - multi-dimensional operations on arrays
  - fast non-uniform Fourier Transform (nuFFT)
  - multi-dimensional (divergence-free) wavelet transform
  - parallel computation on multiple cores and with Graphical Processing Units (GPU)
- Iterative methods
  - Conjugate Gradients (CG)
  - (Fast) Iterative Soft-Thresholding Algorithm (ISTA and FISTA)
  - Alternating Direction Method of Multipliers (ADMM)
  - Iteratively Regularized Gauss-Newton Method (IRGNM)
- Calibration methods for parallel MRI
  - direct calibration from k-space center
  - Walsh's method
  - ESPIRiT
- Reconstruction methods for MRI

  - iterative parallel imaging reconstruction: POCSENSE, SENSE
  - compressed sensing and parallel imaging
  - non-linear inverse reconstruction: NLINV (blind multi-channel deconvolution)
  - calibration-less parallel imaging: SAKE (structured low-rank matrix completion)
- Regularization (in arbitrary dimensions)
  - Tikhonov
  - total variation
  - l1-wavelet
  - (multi-scale) low-rank

1.3 Resources for BART

There are many resources for getting started with BART.

For tutorial-style code demos, please visit the github repository:

https://github.com/mikgroup/bart-workshop.

The material was presented at the 2016 ISMRM Workshop on Data Sampling & Image Reconstruction as a software demo.

Examples of using BART in MATLAB is available at:

https://github.com/mikgroup/espirit-matlab-examples

Please direct all questions or comments to our public mailing list:

mrirecon@lists.eecs.berkeley.edu

https://lists.eecs.berkeley.edu/sympa/info/mrirecon

Note: This list has a public archive! Please do not send any confidential information.

Updates and further information can be found here:

http://mrirecon.github.io/bart/

1.4 License

The tools in BART implement various reconstruction algorithms for Magnetic Resonance Imaging. The software is intended for research use only and NOT FOR DIAGNOSTIC USE. It comes without any warranty (see LICENSE for details).

Please cite the corresponding articles when using these tools. Some references can be found at the end of this file. The source code might provide more detailed references, e.g. for specific iterative algorithms.

CHAPTER 2

Installation

2.1 Prerequisites

BART requires the GCC compiler, the FFTW library, the BLAS/LAPACK library and optionally CUDA for NVIDIA GPU computing. (see *Optional: Turn on GPU acceleration*)

The software can be used in combination with Matlab, python or octave.

There is limited support for reading Cartesian data encoded with the ISMRM Raw Data format when linking with the ISMRMRD library (version 0.5.2) (http://ismrmrd.sourceforge.net/).

In the following, the symbol `$` indicates a shell prompt. Please do not type `$` when entering commands.

2.1.1 Linux

BART should run on any recent Linux distribution.

To install the required libraries on Debian and Ubuntu run:

```
$ sudo apt-get install gcc make libfftw3-dev liblapacke-dev
```

2.1.2 Mac OS X

Xcode command line tools are required and can be installed by running:

```
$ xcode-select --install
```

Mac installation with Macports

Follow the instruction at https://www.macports.org/install.php to install Macports.

To install the required libraries, run:

```
$ sudo port install fftw-3-single gcc6 openblas
```

Mac installation with Homebrew

To install Homebrew (http://brew.sh), run:

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

To install BART, run

```
$ brew install homebrew/science/bart
```

2.1.3 Windows

You can use BART on Windows using Cygwin:

https://www.cygwin.com/

Install Cygwin and select the following packages:

```
Devel: gcc, make
Math: fftw3, fftw3-doc, libfftw3-devel, libfftw3_3
Math: liblapack-devel, liblapack-doc, liblapack0
```

Then use the cygwin shell to compile BART as described below.

An alternative to using Cygwin is a virtual machine with Linux.

2.2 Download and Compilation

If you are a git user, you can simply clone our public repository:

```
$ git clone https://github.com/mrirecon/bart
```

Otherwise, please download the latest version as a zip file from Github:

http://github.com/mrirecon/bart/releases/latest

and unpack it somewhere on your computer.

Open a terminal window and enter the bart directory (the top-level directory with the Makefile in it). To build the reconstruction tools type:

```
$ make
```

If you have installed the ISMRMRD library version 0.5.2, you can also build the ISMRM raw data import tool:

```
$ make ismrmrd
```

2.3 Optional: Turn on GPU acceleration

To turn on GPU acceleration using CUDA, you can toggle the CUDA flag in the Makefile:

```
CUDA=1
```

CHAPTER 3

Data Format

BART's data format consists of a pair of files: one header file (.hdr) and one raw complex float data file (.cfl).

The header file (.hdr) is a simple text readable file that describes the dimensions of the data. Any line that starts with # is regarded as comments. For example, a valid data header for a 192-by-128 dataset is:

```
# Dimensions
192 128 1 1 1 1
```

The raw data file (.cfl) is a binary file containing a single contiguous block of array data of dimensions described in the header stored in column-major order (first index is sequential). The raw data file is complex float (32 bit real + 32 bit imaginary, IEEE 747 binary32 little-endian).

MATLAB functions to read and write our data files may be found in the folder matlab (readcfl.m and writecfl.m). Python functions to read and write our data files may be found in folder‘python <https://github.com/mrirecon/bart/tree/master/python>‘_ (cfl.py).

3.1 Cartesian Datasets

For Cartesian MRI data and images, the dimensions are usually assigned in the following order:

Dimension	Usage
0	readout
1	phase-encoding dimension 1
2	phase-encoding dimension 2
3	receive channels
4	ESPIRiT maps

(more dimensions are defined in src/misc/mri.h)

Undersampled data is stored with zeros in the unsampled positions.

3.2 Non-Cartesian Datasets

Non-Cartesian datasets are stored in two separate BART datafiles: one kspace coordinates dataset and one non-Cartesian samples dataset . Each datafile consists of a header and a raw data file.

The k-space coordinates for each sample are stored along dimension 0 which must have size equal to three. Dimension 1 stores the samples along a single readout windows while dimension 2 may be used to differentiate between different lines (e.g. radial spokes). Channel (3) and map (4) dimensions must not be used (i.e. have size one), while other dimensions can be used as for Cartesian data.

The first five dimensions for the kspace coordinates are usually assigned in the following order:

Dimension	Usage
0	number of dimension (set to 3)
1	readout dimension
2	number of TRs
3	not used (set to 1)
4	not used (set to 1)

The unit of measurement for kspace trajectory is 1/FOV. For example, a valid kspace trajectory for a 192-by-128 image should be scaled such that the x-coordinates cover the range of -192/2 to 192/2 and y-coordinates cover the range of -128/2 to 128/2.

Non-Cartesian samples are stored in a similar way as trajectories except that dimension 0 is not used (i.e. has size one). The channel dimension can be used for different receiver coils as usual.

The first five dimensions for the kspace samples are usually assigned in the following order:

Dimension	Usage
0	not used (set to 1)
1	readout dimension
2	number of TRs
3	number of coils
4	not used (set to 1)

CHAPTER 4

Tool Interfaces

BART tools can be access through multiple interfaces: command-line, Matlab, and Python. All interfaces accept the same arguments.

4.1 Command-line

4.2 Matlab

4.3 Python

CHAPTER 5

Tools

This section focuses on a few main reconstruction tools in BART and details their usage and design.

5.1 ecalib

The ecalib tool estimates coil sensitivies using the ESPIRiT calibration method. It takes a Cartesian kspace data with fully-sampled calibration region as input and outputs ESPIRiT sensitivity maps with the same image dimensions as the input kspace. ecalib usage can be as simple as follows:

```
bart ecalib kspace sensitivity
```

The calibration region does not have to be centered in the kspace array. ecalib estimates the center of kspace by looking at the maximum kspace amplitude and extracts the surrounding region as the calibration region.

Non-Cartesian kspace data must first be gridded onto a Cartesian grid and passed to ecalib. For details about gridding, please see the nufft section.

5.2 pics

The pics tool performs general parallel imaging and compressed sensing reconstruction. For Cartesian imaging, it takes kspace data and sensitivity maps as inputs and outputs an image dataset. By default, pics performs a SENSE reconstruction using conjugate gradient and can be used as simple as:

```
bart pics kspace sensitivity image
```

For non-Cartesian imaging, the trajectory file must be supplied with the flag `-t`. For example:

```
bart pics -t trajectory kspace sensitivity image
```

5.3 nufft

The nufft tool performs non-uniform fast Fourier transform operations.

CHAPTER 6

Viewers

There are multiple ways of viewing BART datasets and images. If you are using the Matlab interface, A Matlab-based image viewer which works well with BART is arrayShow by Tilman Sumpf.

If you have Python installed, you can view BART datasets using the bartview.py script in the folder python to view the images. The simplest way to use it is to install python through Anaconda. To use the script to view a dataset img (with files img.cfl and img.hdr), type the following in the command line:

```
$ python bartview.py img
```

CHAPTER 7

Developer API

This section describes the API available for reconstruction researchers to develop their custom reconstruction methods. The three main abstractions are:

7.1 Iterative Algorithms

Implementation of the operators is separated from the implementation of the iterative algorithm itself and passed as a function pointer. A similar stategy called supermarionation has been described in Murphy et al.

7.1.1 iter interfaces

There are three iter interfaces that allow simplified access of the algorithms: iter, iter2, and iter3. The iter interfaces use the *linop interface* and *operator_p interface*.

The iter interface considers the problem $\min_x \frac{1}{2}\|Ax - y\|_2^2 + g(x)$. It requires $A^\top A$, $A^\top y$, and prox_g as inputs and outputs x. The iter interface supports cg, ist, fista and admm. prox_g has to be NULL for cg.

The iter2 interface considers the problem $\min_x \frac{1}{2}\|Ax - y\|_2^2 + \sum_i f_i(G_i x)$. It requires $A^\top A$, $A^\top y$, prox_{f_i}, and G_i linops as inputs and outputs x. The iter2 interface supports cg, ist, fista and admm. prox_{f_i} has to be NULL for cg. For ist and fista, the number of prox_{f_i} has to be one.

The iter3 interface contains the rest of the algorithms without a unifying interface.

7.1.2 cg

cg implements the conjugate gradient method. It solves for:

$$\min_x \frac{1}{2}\|Ax - y\|_2^2 + \lambda\|x\|_2^2$$

- Parameters:

  Regularization λ and number of iterations

- Input:

  $A^\top A$, and $A^\top y$

- Ouput:

  x

- Iteration steps:

$$\begin{aligned}
\alpha_k &= \frac{r_k^\top r_k}{p_k^\top (A^\top A + \lambda I) p_k} \\
x_{k+1} &= x_k + \alpha_k p_k \\
r_{k+1} &= r_k - \alpha_k (A^\top A + \lambda I) p_k \\
\beta_k &= \frac{r_{k+1}^\top r_{k+1}}{p_k^\top p_k} \\
p_{k+1} &= r_{k+1} + \beta_k p_k
\end{aligned}$$

7.1.3 ist

ist implements the iterative soft threshold method, also known as the proximal gradient method. It solves for:

$$\min_x \frac{1}{2}\|Ax - y\|_2^2 + g(x)$$

where $g(x)$ is simple, that is the proximal of *g(x)* can be computed easily.

- Parameters:

  Step-size τ and number of iterations

- Input:

  $A^\top A$, $A^\top y$, and prox_g

- Ouput:

  x

- Iteration steps:

$$x_{k+1} = \text{prox}_{\tau g}\left(x_k + \tau(A^\top y - A^\top A x_k)\right)$$

7.1.4 fista

fista implements the fast iterative soft threshold method, also known as the accelerated proximal gradient method. It solves for:

$$\min_x \frac{1}{2}\|Ax - y\|_2^2 + g(x)$$

where $g(x)$ is simple, that is the proximal of *g(x)* can be computed easily.

- Parameters:

  Step-size τ and number of iterations

- Input:

  $A^\top A$, $A^\top y$, and prox_g

- Ouput:

  x

- Iteration steps:

$$
\begin{aligned}
x_k &= \text{prox}_{\tau g}\left(z_k + \tau(A^\top z_k - A^\top A z_k)\right) \\
t_{k+1} &= \frac{1 + \sqrt{1 + 4t_k^2}}{2} \\
z_{k+1} &= x_k + \left(\frac{t_k - 1}{t_{k+1}}\right)(x_k - x_{k-1})
\end{aligned}
$$

7.1.5 admm

admm implements the alternating direction method of multipliers. It solves for:

$$\min_x \frac{1}{2}\|Ax - y\|_2^2 + \sum_i f_i(G_i x - b_i)$$

where f_i s are simple, that is the proximal of each *f_i* can be computed easily.

- Parameters:

  Convergence parameter ρ, and number of iterations

- Input:

  $A^\top A$, $A^\top y$, prox_{f_i}, G_i, $G_i^\top$, $G_i^\top G_i$, and b_i

- Ouput:

  x

- Iteration steps:

$$
\begin{aligned}
x &= \left(A^\top A + \rho \sum_i G_i^\top G_i\right)^{-1} \left(A^\top y + \rho \sum_i G_i^\top (z_i - u_i + b_i)\right) \\
z_i &= \text{prox}_{f_i/\rho}(G_i x + u_i - b_i) \\
u_i &= G_i x + u_i - z
\end{aligned}
$$

7.2 Linear Operators

7.2.1 linop interface

The linop interface provides an abstraction for linear operators. For each linop A, it supports forward operation $x \to A(x)$, adjoint operation $x \to A^\top(x)$ and normal operation $x \to A^\top A(x)$.

7.2.2 linop_chain

The linop_chain function is arguably the most powerful feature of the linop interface. Given two linops A and B, the composite linear operator $A(B)$ can be created by doing:

```
AB = linop_chain(A, B)
```

This automatically chains the forward operation, adjoint operation, and normal operation.

7.2.3 fft

fft implements the fast Fourier transform using the FFTW library for CPU and cudaFFT for GPU.

7.2.4 nufft

7.2.5 sense

7.2.6 wavelet

7.2.7 finite_diff

7.3 Proximal Operators

7.3.1 operator_p interface

operator_p is BART's standard interface for proximal operators. It is based on operator but with the constraint of having one input/output pair and one parameter.

CHAPTER 8

Software Implementation

8.1 Memory-mapped Input/Output

Input and output is performed using memory-mapped files. This allows efficient access to small parts of a large file, e.g. a slice of a 3D data set.

The use of memory-mapped input/output allows simple processing of extremely large data sets. While all data can be accessed simply, either directly with points or using the functions of the library, only the parts of the data which are actually accessed are loaded into memory.

It is also possible to access data during long-running computations or debugging stops.

8.2 Multi-Dimensional Arrays

Multi-dimensional arrays are part of almost every framework for image reconstruction. A set of functions is provided for basic addressing/indexing, mathematical operations, and some transforms. It offers simple but powerful interfaces for many operations on multi-dimensional arrays, e.g. to access slices of an array or to apply an FFT along selected dimensions.

8.3 GPU Acceleration

Most operations can be transparently accelerated using GPUs. Data can be allocated on the GPU using:

The generic copy function `md_copy` can be used to copy data from on to the GPU. Basic operations, FFT, ... are implemented for the CPU and GPU. The library keeps track of all allocated memory regions and automatically uses the right function.

CHAPTER 9

Unit Tests

Procedures to add unit tests for particular file:

- add test under bart/src/utests
- include “utest.h” in testfile
- add “UTARGET += test_file” under the appropriate rule files under “bart/src/rules”