Magnetic Resonance Imaging xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Magnetic Resonance Imaging

journal homepage: www.elsevier.com/locate/mri

Original contribution

snapMRF: GPU-accelerated magnetic resonance fingerprinting dictionary generation and matching using extended phase graphs

Dong Wang[a],[*], Jason Ostenson[b], David S. Smith (Ph.D.)[b]

[a] School of Science, Nanjing University of Science and Technology, Nanjing, Jiangsu, China
[b] Institute of Imaging Science, Vanderbilt University Medical Center, Nashville, TN, USA

A R T I C L E I N F O

Keywords:
Magnetic resonance fingerprinting
Graphics processing units
Extended phase graph
Quantitative MRI
Relaxometry
Non-Cartesian

A B S T R A C T

Purpose:: Magnetic resonance fingerprinting (MRF) is a state-of-the-art quantitative MRI technique with a computationally demanding reconstruction process, the accuracy of which depends on the accuracy of the signal model employed. Having a fast, validated, open-source MRF reconstruction would improve the dependability and accuracy of clinical applications of MRF.
Methods:: We parallelized both dictionary generation and signal matching on the GPU by splitting the simulation and matching of dictionary atoms across threads. Signal generation was modeled using both Bloch equation simulation and the extended phase graph (EPG) formalism. Unit tests were implemented to ensure correctness. The new package, snapMRF, was tested with a calibration phantom and an in vivo brain.
Results:: Compared with other online open-source packages, dictionary generation was accelerated by 10–1000 × and signal matching by 10–100 ×. On a calibration phantom, T_1 and T_2 values were measured with relative errors that were nearly identical to those from existing packages when using the same sequence and dictionary configuration, but errors were much lower when using variable sequences that snapMRF supports but that competitors do not.
Conclusion:: Our open-source package snapMRF was significantly faster and retrieved accurate parameters, possibly enabling real-time parameter map generation for small dictionaries. Further refinements to the acquisition scheme and dictionary setup could improve quantitative accuracy.

1. Introduction

Magnetic resonance fingerprinting (MRF) has shown great promise for accelerating quantitative magnetic resonance imaging (MRI) [1-3]. MRF data acquisition and reconstruction involve four steps: (1) data acquisition, (2) image reconstruction, (3) signal simulation, and (4) pattern matching. While the primary limitation on the speed of data acquisition is the scanner hardware, the latter two steps are subject to severe computational bottlenecks.

A key step of the reconstruction process for MRF data is the generation of a dictionary of simulated signals. The accuracy of MRF depends critically on the accuracy of the dictionary of simulated signals. The original implementation of MRF [1] used a simple Bloch simulation of a single isochromat. This approximation is very fast, but becomes inaccurate in the presence of inhomogeneities in the main magnetic field across the voxel that cause dephasing. In order to take this gradient dephasing into account, the Bloch model can be extended by averaging over an ensemble of isochromats, but this is computationally expensive and still an approximation.

The extended phase graph (EPG) model [4] is an alternative approach that has been used previously in FISP (fast image with steady precession) signal simulations [5]. The EPG method describes the spin system as several discrete configuration states using the Fourier transform. This provides more accurate signal evolution compared to Bloch simulation when the spin system is affected by inhomogeneous magnetic fields, such as the gradient crusher field introduced by unbalanced SSFP. However, EPG is computationally difficult because it must track the signal history for ~100 or more discrete states for each simulated signal. In fact, as will be shown later, it takes hours to generate a dictionary of the same size as that used in Ref. [1] using the EPG method without parallelization on our hardware described below, so the time cost for increased signal model accuracy is large.

Besides the signal simulation, the fourth step of matching the measured signal with the most similar dictionary atom is also time consuming. Ma et al. [1] used a template matching approach that computed the complex inner product of the normalized voxel signal

[*] Corresponding author.
E-mail addresses: 311112253@njust.edu.cn (D. Wang), jason.ostenson@vanderbilt.edu (J. Ostenson), david.smith@vumc.org (D.S. Smith).

https://doi.org/10.1016/j.mri.2019.11.015
Received 15 June 2019; Accepted 10 November 2019

with each atom in the dictionary. Specifically, let $X = \mathbf{x}_i \in \mathbb{C}^L, i \in \{1,2,\ldots,N\}$, be the complex fingerprinting data with N voxels and L time points and $D = \{\mathbf{d}_k\} \in \mathbb{C}^L, k \in \{1,2,\ldots,P\}$, be the predefined dictionary with P unit-normalized atoms. If we define $\langle\cdot\rangle$ as the complex inner product, then the best matching dictionary atom is then computed by

$$\hat{k}_i = \underset{k}{\operatorname{argmax}} |\langle \mathbf{d}_k, \mathbf{x}_i \rangle|. \tag{1}$$

Note the proton density at voxel i can be simultaneously derived using the maximum inner product that was found:

$$\hat{\rho}_i = |\langle \mathbf{d}_{\hat{k}_i}, \mathbf{x}_i \rangle|. \tag{2}$$

The computational complexity of matching grows as $O(PNL^2)$ with the dictionary size P, the number of time points L, and the number of voxels N. This scaling is intolerable when large dictionaries are needed in rapid imaging techniques. For example, in Ref. [1], using MATLAB (MathWorks, Natick, MA) to generate a dictionary containing 563,783 entries with 1000 time points each took 399 s and matching required about 3 minutes for 128 × 128 pixels on their hardware.

Besides Ma et al.'s MATLAB code contained in the supplementary file of Ref. [1], there are other open-source MRF packages available. [6] published a MATLAB package called EPG-X, which is capable of modeling different kinds of MRI sequences using EPG. However, their code is implemented on the CPU and is slower, especially in dictionary generation. EPG-X also cannot handle sequences with variable T_E or T_R, which are commonly used in MRF, and does not handle B_1^+ effects. [7] released a C++ package called PnP-MRF that simulates the whole procedure of MRF. PnP-MRF does not implement the EPG method and is written for the CPU and is not parallel, except in its use of SIMD instructions for vector math. Also, users have to hard code the values of T_1 and T_2 for the dictionary directly into the source, which is not user-friendly, and does not handle B_1^+ corrections.

Several mathematical methods have been used to accelerate dictionary matching. One strategy is to compress the dimension of the dictionary in the temporal or parameter direction [8, 9]. In Ref. [8], McGivney et al. used singular value decomposition (SVD) to compress the dictionary in the time domain, providing a low-rank approximation that achieved a speed-up of the matching by a factor of 3.4–4.8. However, SVDs are time consuming, especially when the dimension is large. In Ref. [9], Cauley et al. introduced a fast group matching algorithm that divided MRF dictionaries into several clustered groups and used group principal component analysis to compress the dictionary in both temporal and parameter dimensions. Both methods, however, did not accelerate dictionary generation, which often takes longer than matching. Also, these approaches are approximations that could introduce additional parameter error.

Another category of acceleration uses a neural network to learn a non-linear function to generate parameter maps from the voxel signals [10]. This direct inference is much faster than exhaustive pattern matching on the whole dictionary, so a great speed up can be realized. The drawback of this method is that it depends critically on the exact sequence used for data collection. If anything at all is changed about the sequence, such as any particular flip angle, T_E, or T_R in the series, a new dictionary must be generated and the network retrained. An ideal solution would be both nearly instantaneous and robust to the details of acquisition. This is possible with the standard, exhaustive dictionary matching if properly parallelized.

Graphics processing units (GPUs) are now commonly used to speed up parallel calculations in many areas of imaging science (e.g. [11-13]), often enabling a 10–100× reduction in computation time. However, to the best of our knowledge, no one has used GPUs to generate MRF dictionaries and perform template matching. Also relevant here is the growing concern over the lack of reproducibility in scientific research [14-17]. Thus, to these ends, we present snapMRF, an open-source, validated MRF reconstruction package that parallelizes dictionary generation and template matching entirely on the GPU and implements a battery of unit tests to ensure reliability and accuracy of the generated signal. snapMRF is capable of simulating different kinds of MRI sequences using both single-isochromat Bloch equation and EPG simulations, as well as implementing template matching. Also snapMRF is as fast as neural network inference without the dependence on the sequence or requiring a training step.

The paper is organized as follows. Section 2 discusses the snapMRF design, data collection, and experimental design. Section 3 presents the results of time coding, parameter accuracy measurements on a standard quantitative MRI phantom and image quality evaluation on in vivo brain data. Section 4 gives the discussion and conclusions. The GPU kernels and unit tests are described in Appendices A and B, respectively.

2. Methods

2.1. snapMRF code overview

The algorithm that implements the fully parallel EPG signal modeling and pattern matching in snapMRF has five stages, broken into two input, two computation, and one output. Note that variables prefixed with a `*` are arrays.

- **Read MRF pulse sequence** from a CSV file to host array `*h_mrf` and copy to device array `*d_mrf`. Arrays contain four vectors: flip angle (`*d_FA`) and RF phase (`*d_phi`) in degrees, and repetition time (`*d_TR`) and echo time (`*d_TE`) in milliseconds. The number of time points are saved in `nreps`.
- **Read dictionary configuration** (T_1, T_2, B_0, B_1^+) from command line using `parse_length()` and `parse_params()` and save in host arrays `*h_t1`, `*h_t2`, `*h_b0`, and `*h_b1` respectively. Then for later convenience `*h_t1`, `*h_t2`, `*h_b0` and `*h_b1` are combined in a new host variable `*h_params` using `trans_params()` and then copied to the GPU array `*d_params`. The values of T_1, T_2, B_0 and B_1^+ are saved in T_1-major order, with the index of T_1 changing the fastest, then T_2, then B_0, and B_1^+ the slowest. The numbers of T_1, T_2, B_0 and B_1^+ are stored in the variables `l_t1`, `l_t2`, `l_b0` and `l_b1` respectively and the number of atoms in the dictionary is stored in `natoms`, which is computed using the `compute_natoms()`. Situations when $T_1 \leq T_2$ are removed.
- **Generate dictionary atoms** directly on the GPU using `MRF_dict()` and save in the device array `*d_atoms`. For each B_1^+ and each time point, the kernels are parallelized with respect to the index of the atoms. In other words, the signal for all atoms are calculated at the first time point only, then time is advanced by updating the transition matrix. After the RF pulse effects are stepped forward, the next time point is computed for all atoms. This reduces evaluation of the transition matrix, which contains costly transcendental functions, to once per time point. The atoms are ordered as in the parameter array `*d_params`.
- **Match image voxels to dictionary.** Read the gridded, undersampled fingerprinting images from a RawArray [18] file and save in `*d_img`. The function `MRF_match()` computes the maps in two steps. First, for each sub-dictionary that has a different B_1^+ (`l_b1` sub-dictionaries), compute the maps and save them in `*d_MAPS` in the order of B_1^+. For large dictionaries, GPU memory can be limited, so matching is split into even groups of voxels using `compute_nsplits()` that can fit into the remaining GPU memory. `MRF_minimatch()` then computes the maps for each group of voxels. To optimize matching speed, the measured signal is reshaped into a matrix, and the complex inner products of the atoms with the signals are computed by multiplying the conjugate of the dictionary by the signal matrix. This allows the use of the highly optimized cuBLAS matrix multiply function. The product matrix is then reduced via a custom kernel `generate_maps()` that finds the maxima and copies the parameters associated with these maxima

from `*d_params` into the output parameter maps. The second step uses `merge_maps()` and B_1^+ correction to generate the final maps. This kernel is parallelized by voxel. For each voxel, it takes the B_1^+ in `*d_b1` that is closest to the measured B_1^+ and selects the T_1, T_2, and B_0 associated with this B_1^+ in `*d_MAPS` as the final parameters. The closest match B_1^+ is included in the parameter maps as well. The final maps are stored in `*d_maps` in the loop order of T_1, T_2, off-resonance, B_1^+, and proton density.

- **Save results.** Copy the atoms and maps from GPU back to CPU and save to RawArray files. Free all allocated memory.

Time consuming transfers of data between CPU and GPU happen only twice, namely copying the RF pulse and fingerprinting images to the GPU at the beginning and saving the dictionary and parameter maps to the CPU at the end. All other steps are computed directly on the GPU.

2.2. Code timing and parameter accuracy

For code timing and parameter accuracy measurements (see below), an MRI system phantom [19] (HPD, Boulder, Colorado) was imaged coronally at 3.0 T (Philips Ingenia, Philips Healthcare, The Netherlands) using a 32-channel head coil. The phantom was imaged at a single slice through a series of contrast spheres with different $MnCl_2$ concentrations, giving different T_1 and T_2 values. The phantom temperature was the same as that during the reference measurements. The k-space data were acquired with a numerically calculated [20] uniform spiral with undersampling factor of 32. The FOV, in-plane and through-plane resolutions, and spiral acquisition time were 240 mm × 240 mm, 1 mm × 1 mm, 10 mm, and 4.9 ms, respectively.

The raw MRF data were gridded and coil-combined using the Berkeley Advanced Reconstruction Toolkit (BART; [21]). Sensitivity maps were estimated using eSPIRIT in BART. Sampled density corrections were determined using Ref. [22]. The final reconstructed data dimensions of the undersampled images were 1000 or 1500 × 240 × 240 depending on the MRF sequences below.

The MRF sequences used for the phantom were adapted from the first reported MRF unbalanced SSFP sequence [5]. The sequence used adiabatic inversion with an inversion time (T_I) of 40 ms, excitation with a sinc-gauss pulse and a time-bandwidth product of 10 to minimize B_1^+ heterogeneity in the slice profile [23]. Two variations of this sequence were used, one with a variable flip angle and one with variable flip angle *and* variable T_R. For the fixed T_R sequence, the T_E was fixed at 4.65 ms and the T_R was fixed at 16 ms [24]. For the variable T_R sequence, the T_R and flip angle were varied, with T_R variations $\in$ [16,18.7] ms and flip angle $\in$ [0,60] deg, similar to Jiang et al. The T_E was fixed at 3.5 ms. The scan duration was 17.5 s (1000 repetitions) for the variable T_R sequence and 24.0 s (1500 repetitions) for the fixed T_R sequence.

For timing comparisons, T_1 values were set to be between 100 ms and 3000 ms and T_2 to be between 20 ms and 2000 ms. To test the running time with respect to the number of atoms `natoms` in the dictionary, we selected 10 different increments for T_1 and T_2, resulting in 10 different numbers of atoms. We compared the run time with both EPG-X [6] and PnP-MRF [7] and plotted the time curves.

For parameter accuracy evaluation, we compared the performance of snapMRF and EPG-X in generating T_1 and T_2 maps. Since EPG-X can only deal with fixed T_R sequences, we used the fixed T_R sequence for direct comparison. To show the influence of variable T_R and B_1^+ effects on the final T_1 and T_2 maps, we also ran snapMRF with the variable T_R sequence.

For the experiments with no B_1^+ effects included, T_1 values were chosen to be between 50 ms and 2500 ms with an increment of 5 ms, and T_2 values were chosen to be between 5 ms and 600 ms with an increment of 2.5 ms, resulting in a final dictionary size of 105,028. For the case with B_1^+ effects included, T_1 values were chosen to be between 50 ms and 2500 ms with an increment of 12.5 ms, T_2 values were chosen to be between 5 ms and 600 ms with an increment of 5 ms, and B_1^+ values were chosen to be between 0.8 and 1.2 with an increment of 0.1, resulting in a final dictionary size of 105,655. Note that in both cases, the number of atoms was almost the same. The accuracy of the parameter maps was assessed with the relative error as follows:

$$\mathrm{err} = \frac{\|\mathbf{T}_\mathrm{m} - \mathbf{T}_\mathrm{gt}\|_2}{\|\mathbf{T}_\mathrm{gt}\|_2} \times 100\%, \tag{3}$$

where $\mathbf{T}_\mathrm{m}$ is the vector of corresponding parameters for the vials as reconstructed by snapMRF or EPG-X, and $\mathbf{T}_\mathrm{gt}$ is the vector of ground truth parameters.

2.3. Image quality

For image quality evaluation, a single volunteer was imaged at 3.0 T (Philips Ingenia, Philips Healthcare, The Netherlands) in a transverse slice in the brain using a 32-channel head coil after informed consent and with approval of the institutional review board. The MRF sequences used in the brain were the same as that for the phantom data. The k-space data were acquired with a numerically calculated [20] uniform spiral with undersampling factor of 32. The FOV, in-plane and through-plane resolution, and spiral acquisition time were 240 mm × 240 mm, 1 mm × 1 mm, 5 mm, and 5.1 ms, respectively. The same reconstruction method was used as for the phantom data. The reconstructed data dimensions were 1000 or 1500 × 240 × 240.

In this experiment, we visually compared the parameter maps generated using snapMRF and EPG-X. For the experiments with no B_1^+ effects included, T_1 values were chosen to be between 100 ms and 4000 ms with an increment of 10 ms, and T_2 values were chosen to be between 20 ms and 2000 ms with an increment of 5.5 ms, resulting in a final dictionary size of 108,056 atoms. For the case with B_1^+ effects included, T_1 values were chosen to be between 100 ms and 4000 ms with an increment of 20 ms, T_2 values were chosen to be between 20 ms and 2000 ms with an increment of 14.5 ms, and B_1^+ values were chosen to be between 0.8 and 1.2 with an increment of 0.1, resulting in a final dictionary size of 102,830 atoms. Note that the number of atoms was deliberately chosen to be similar in both cases.

All codes were run on a dual 10-core Intel Xeon E5-2630 2.20 GHz with 256 GB RAM with an Nvidia TITAN V GPU.

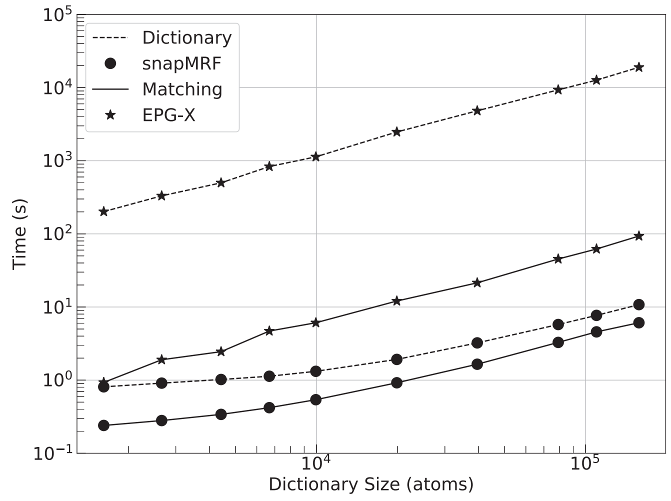
3. Results

3.1. Code timing

Fig. 1 shows the run time comparison with EPG-X (Top) and PnP-MRF (Bottom) in both dictionary generation and matching, plotted on a logarithmic scale. As can be seen, all the time curves grow linearly with respect to the number of atoms, as expected, with snapMRF requiring consistently 10–100 × less time than EPG-X and PnP-MRF. Specifically, snapMRF outperformed EPG-X by 60–1700× in dictionary generation (Top, dotted lines) and by about 2–20× in matching (Top, solid lines), and outperformed PnP-MRF by 2–100× in dictionary generation (Bottom, dotted lines), and by about 60–500× in matching (Bottom, solid lines). Thus overall, dictionary generation was accelerated by 10–1000× and template matching by 10–100×. Note that snapMRF performs much faster than the other two packages especially when the dictionary is large.

3.2. Parameter accuracy

Tables 1 and 2 present the estimated T_1 and T_2 values and the corresponding relative errors in the phantom data respectively. The running time of dictionary generation and template matching is shown in Table 3. Note that for all the sequences, snapMRF can generate 100,000 atoms and perform matching in less than 20 s, much faster



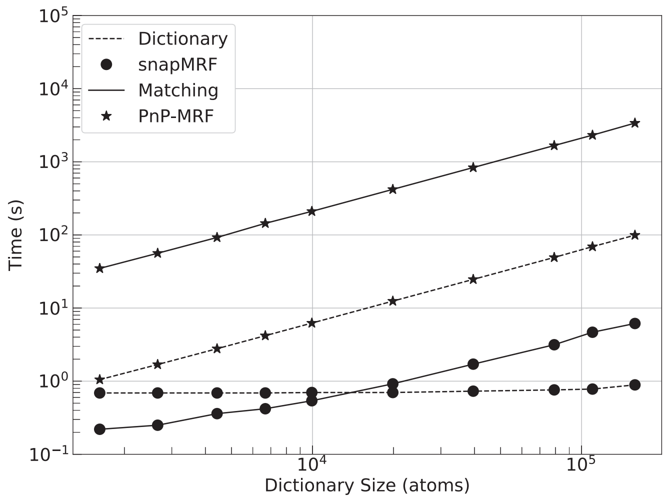


Fig. 1. Run time comparison with EPG-X (Top) and PnP-MRF (Bottom) in both dictionary generation and matching. Note the log scale. Time increases linearly with the dictionary size, showing efficient parallelization. For this example, with 240 × 240 image voxels, matching took much less time than dictionary generation.

Table 1
T_1 accuracy comparison between snapMRF and EPG-X on phantom data. True T_1 is the ground truth of T_1 based on the phantom manufacturer's documentation. snapMRF was quite accurate, even using an unoptimized sequence and dictionary.

True T_1 (ms)	EPG-X fixT_R	snapMRF fixT_R	snapMRF varT_R	snapMRF varT_R + B_1^+
90.9	128.5	127.7	111.5	94.2
126.9	155.0	155.0	146.5	127.9
176.6	173.1	173.1	172.3	153.8
244.2	280.4	280.4	265.0	225.0
336.5	342.7	342.7	326.5	319.2
458.4	471.2	471.2	471.9	468.3
608.6	602.7	601.2	625.0	622.1
801.7	771.5	770.4	818.5	813.5
1044.0	945.0	943.1	1032.3	1026.0
1332.0	1262.7	1263.1	1310.8	1306.7
1604.0	1568.8	1568.1	1607.3	1593.3
1907.0	1861.2	1861.9	1854.2	1828.8
2173.0	2043.1	2043.1	2091.9	2094.2
2480.0	2366.5	2366.2	2434.6	2416.3
err (%)	4.9	5.0	2.6	3.0

Table 2
T_2 accuracy comparison between snapMRF and EPG-X on phantom data. True T_2 column is the ground truth of T_1. snapMRF was quite accurate, even using an unoptimized sequence and dictionary.

True T_2 (ms)	EPG-X fixT_R	snapMRF fixT_R	snapMRF varT_R	snapMRF varT_R + B_1^+
5.6	6.9	6.9	9.4	12.3
7.9	11.5	11.2	10.0	13.5
11.2	13.3	13.3	11.2	13.5
15.8	13.7	13.5	11.3	20.4
22.6	21.2	21.2	23.3	30.0
32.0	32.3	32.3	38.3	47.3
46.4	45.0	44.8	50.4	60.0
64.1	64.2	64.2	70.2	83.8
96.9	84.6	84.4	90.8	104.6
133.3	144.0	143.8	146.9	170.8
190.9	175.4	175.4	185.2	213.8
278.1	266.5	266.5	255.4	290.0
403.5	323.3	323.5	343.7	407.7
581.3	474.0	474.2	453.5	531.5
Err (%)	16.9	16.9	17.9	9.3

Table 3
Running time comparison between snapMRF and EPG-X on both dictionary generation and template matching. Note that snapMRF is much faster than EPG-X.

Running time (s)	EPG-X fixT_R	snapMRF fixT_R	snapMRF varT_R	snapMRF varT_R + B_1^+
phantom/dict	17,797.1	11.0	7.4	9.4
phantom/match	137.1	6.0	4.1	4.9
brain/dict	18,629.8	11.3	7.6	8.7
brain/match	143.6	6.1	4.2	4.6

than EPG-X.

We can see from Tables 1 and 2 that for a fixed T_R sequence, snapMRF and EPG-X performed nearly identically, which is as expected because both used the EPG model. For variable T_R sequence, the error for T_1 went down to 2.6% while the error for T_2 remained the same. When B_1^+ effects were included, T_1 error remained low and T_2 error reduced to 9.3%. The results indicate that including both variable T_R and B_1^+ effects may improve the accuracy of parameter maps.

Fig. 2 shows the maps of T_1, T_2 and proton density separately reconstructed using snapMRF and EPG-X. To get a better view of the regions of interest, a binary mask was generated using proton density map by using Otsu's method to select the signal from the background and then computing the convex hull of that mask to fill in gaps. The final mask was applied to all the maps to mask out regions of low signal. Furthermore, a circular mask centered in the middle of the map was applied to the T_1 map and the T_2 map to highlight the regions of interest. For display, the T_1 map was clamped to the range 0–2500 ms, and the T_2 map was clamped to 0–800 ms.

As is shown in Fig. 2, for fixed a T_R sequence, snapMRF and EPG-X generated visually the same parameters, which again illustrates that the EPG model is implemented correctly in snapMRF. For variable T_R, the T_1 map was improved while the T_2 map remained the same. When B_1^+ effects were included, the T_2 was also improved.

3.3. Image quality

Fig. 3 shows an in vivo brain example using snapMRF to see if snapMRF could generate clean in vivo parameter maps. A signal mask was generated using the same method as in the phantom case and applied to all maps to remove parameter estimates from regions of low signal. For display, the T_2 map was clamped to a range of 0–300 ms, and the proton density map was clamped to 0–0.25. The running time is shown in Table 3.

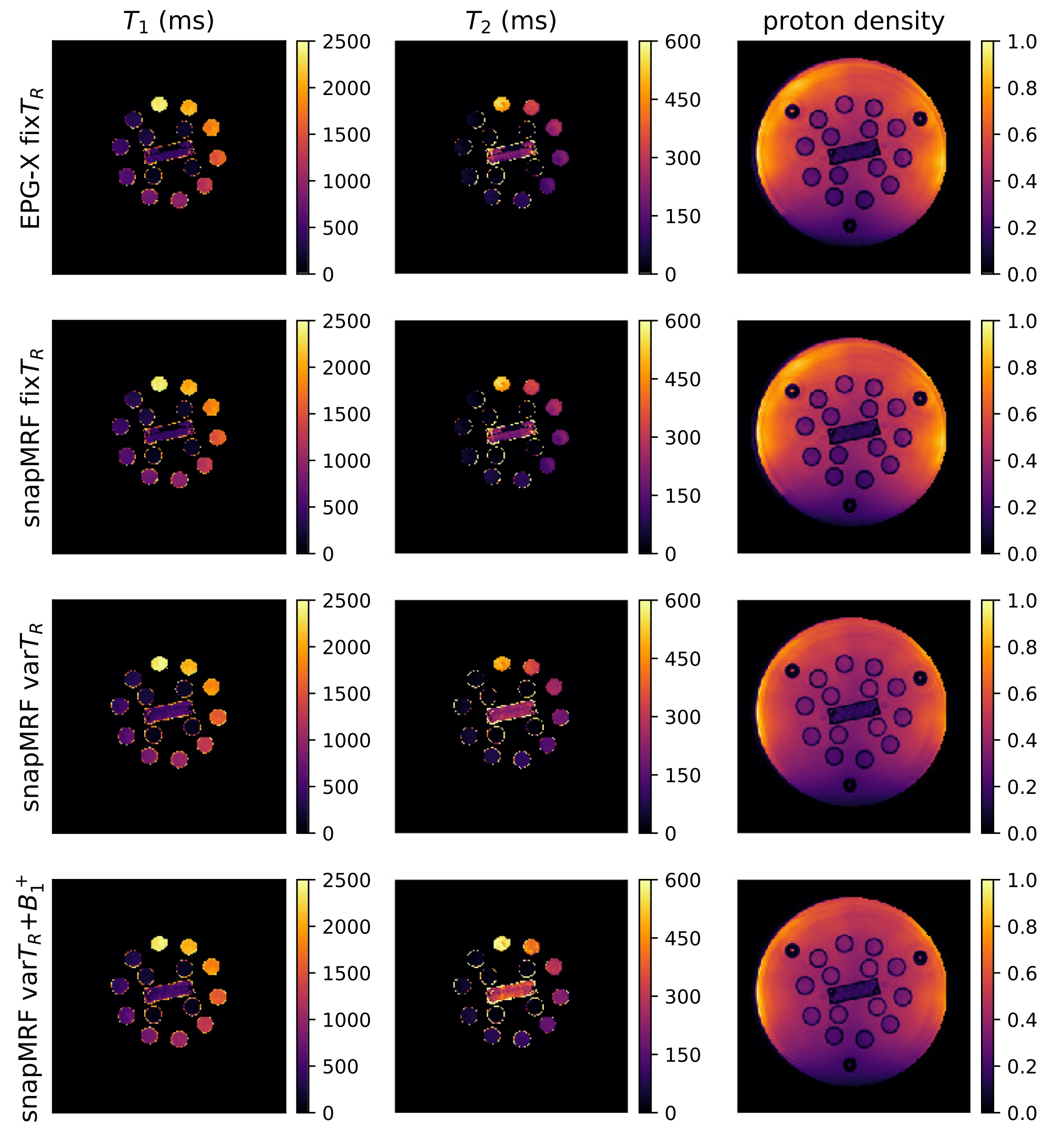


Fig. 2. Parameter accuracy comparison between snapMRF and EPG-X on phantom data. Top row: Parameter maps generated by EPG-X using fixed T_R sequence. From left to right: T_1, T_2, and proton density, respectively. The second through the last rows are the maps generated by snapMRF using a fixed T_R sequence, a variable T_R sequence and a variable T_R sequence with B_1^+ calibration, respectively. Quantitative estimates of the T_1 and T_2 values are contained in Table 1 and Table 2. The running time is shown in Table 3. In all the T_1 and T_2 maps, the background water filling the phantom was masked out for better visibility of the sample vials.

We can see from Fig. 3 that, in all cases, parameter maps were generated with good quality. Again for the fixed T_R sequence, snapMRF and EPG-X generated visually identical maps. For the variable T_R sequence, T_1 maps were smoother and proton density maps were darker than for the fixed T_R case. When B_1^+ effects were included, the T_2 map became smoother, especially in the gray matter.

4. Discussion and conclusion

In this paper, we presented a new package called snapMRF, which performs MRF dictionary generation and signal matching entirely on the GPU. snapMRF is capable of accurately generating parameter maps in real-time. Compared with other online open-source packages, dictionary generation using snapMRF was 100–1000× faster and signal matching was 10–100× faster. Additionally, snapMRF can handle more complex pulse sequences flexibly, including variable T_R, T_E, and phase (besides just flip angle), and it can match both off-resonance and transmit B_1 inhomogeneities, if desired.

In Tables 1 and 2, snapMRF and EPG-X differed very slightly on the fixed T_R sequence. We believe this may be a result of the fact that snapMRF uses single precision floats while EPG-X uses double precision.

In conclusion, snapMRF is as fast as neural network inference (i.e., timescales of seconds) without the dependence on the sequence details. Furthermore, if one still desires to use a neural network, the present work can be used for dictionary generation. The enhanced speed of snapMRF could potentially be used to decrease calculation time for multiple locations experiencing different B_1^+ amplitudes. This is applicable to variations in applied radiofrequency across the image, as well as variations along the slice profile [23].

An advantage of EPG is that it is easy to incorporate other effects, such as chemical shift, magnetization transfer [25], motion, diffusion, slice profile, etc., which are essential in advanced imaging techniques. In this work, snapMRF was used to model and fit B_1^+ effects in addition to relaxation parameters using a variable T_R MRF sequence with fixed T_E. Future work could extend snapMRF to include additional effects such as those mentioned above.

All the experiments above were run on a single GPU card, but snapMRF could be further accelerated with asynchronous workload splitting across multiple GPUs, such as used in Ref. [13]. In theory,

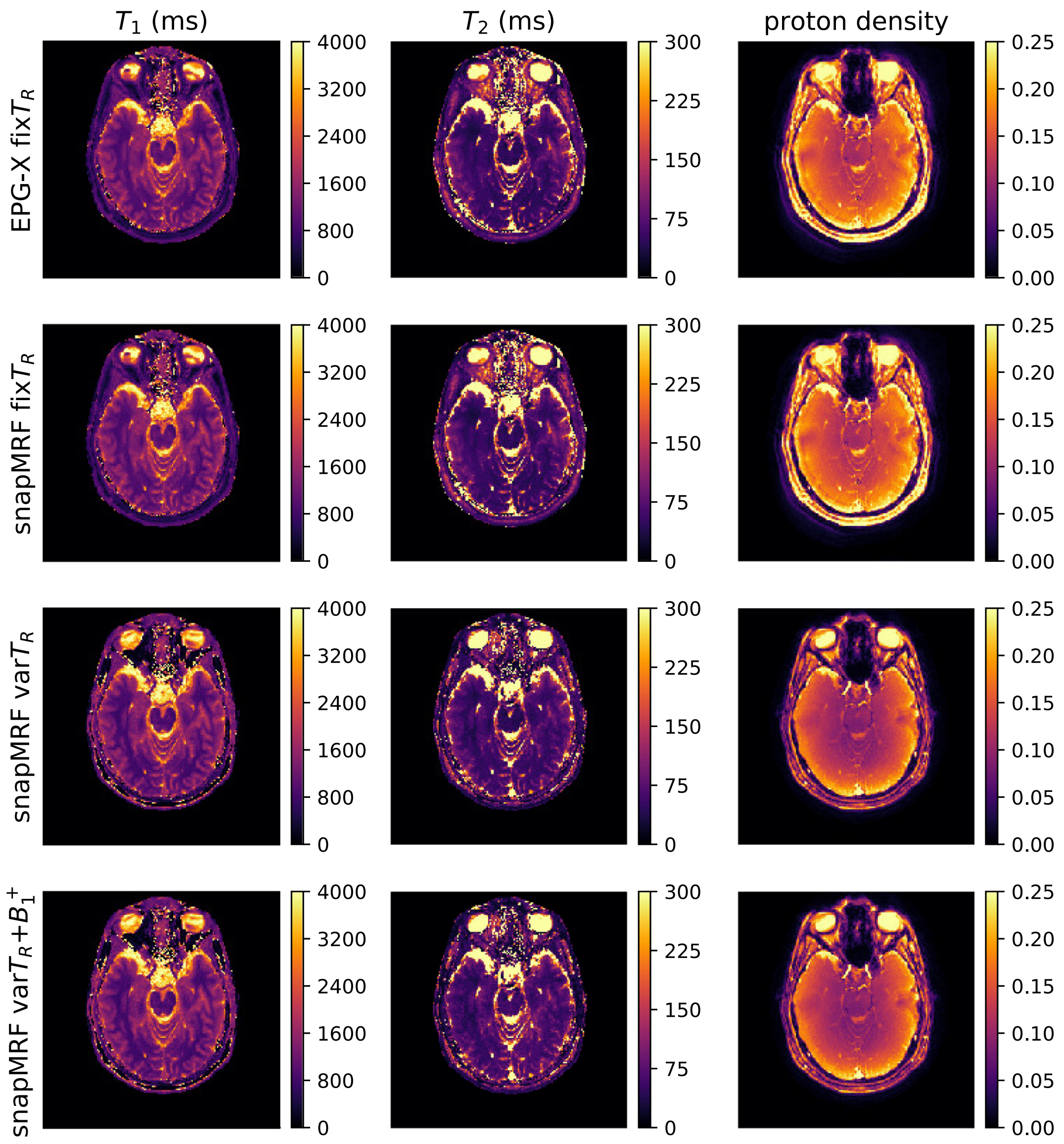


Fig. 3. Comparing snapMRF and EPG-X on an in vivo brain. Top row: Parameter maps generated by EPG-X using a fixed T_R sequence. From left to right: T_1, T_2, and proton density, respectively. The second through the last rows are the maps generated by snapMRF using a fixed T_R sequence, a variable T_R sequence and a variable T_R sequence with B_1^+ calibration, respectively. The running time is shown in Table 3. Note that all cases generated parameter maps with high quality.

further acceleration proportional to the number of GPU cards could be achieved because signal generation is parallelized with respect to atoms and matching is performed for each voxel independently.

The version of snapMRF used here can be obtained by cloning the snapMRF repository [26] and checking out the tag `v0.0.1`.

Acknowledgments

We gratefully acknowledge funding from NIH R01 DK105371,NIH R01 EB017230, andNIH K25 CA176219.

Appendix A. Description of CUDA kernels

In Nvidia's Compute Unified Device Architecture (CUDA), a kernel is a special function that executes in parallel on the GPU, with one instance of the function called per GPU thread. It is then natural to divide up algorithms according to data processing steps, so that each parallelizable step becomes one kernel. We now describe the kernels contained in snapMRF.

`init_rf_pulse()` initializes the magnetization with state $[0,0,1]^T$ by default and applies an inversion decay and initial hard pulse, as specified by the flip angle `alpha` and phase `phi` in degrees and inversion time `TI` in ms. The input `natoms` is the number of atoms, `nstates` is the number of states in the state matrix, and `*d_w` is the output configuration state matrix.

`fill_transition_matrix()` computes the elements of the transition matrix `*d_T_m` with flip angles α and RF pulse phases Φ which are saved in the input array `*d_mrf`. The transition matrix `*d_T_m` corresponds to the T operator in the EPG algorithm, and `index` is the time index of the echo.

`apply_rf_pulse()` multiplies the transition matrix `*d_T_m` and the state matrix `*d_w` and saves the result in the state matrix `*d_w`. The single-precision, complex matrix multiplication is conducted using the built-in cuBLAS function `cublasCgemm()`.

`shift_phase()` applies an in-place phase shift for B_0 to the transverse components of the state matrix `*d_w`.

`save_atom()` saves the most recent echo intensities from the state matrix `*d_w` to the dictionary `*d_atoms`. The input `nreps` is the number of times points of the sequence, and `index` is the time index of the echo.

`decay_signal()` applies T_1 and T_2 decay to the state matrix `*d_w` within a certain time interval and saves the result back in `*d_w`. Four valid values of `type` for the time intervals are available: T_E (`type=1`), $T_R - T_E$ (`type=2`), $T_E/2$ (`type=3`), and T_R (`type=4`). Here T_E and T_R are contained in the pointer `*d_mrf` while T_1 and T_2 are contained in `*d_params`. This kernel corresponds to the E operator in the EPG algorithm.

`dephase_gradients()` applies dephasing to the state matrix `*d_w` caused by the intra-voxel gradient and saves the shifted state to `*d_w`. This kernel corresponds to the shift operator S in the EPG algorithm.

Appendix B. Unit tests

Unit tests are simple test cases that can be validated to infer coded correctness and are useful for increasing the reproducibility of computational research. The normal result of unit tests is for them to all pass. If one or more fail, that can signal a code bug or algorithm mistake. Seven unit tests based on simple pulse sequences with analytic solutions and one test based on dictionary matching were performed in snapMRF.

All the unit tests are contained in the source file `test.cu`. Table A.1 shows the parameters used for each echo. For all the sequences, $B_0 = 0$ for simplicity. For more information about the echoes, please refer to Refs. [4] and [27].

Table A.1
Parameters used for unit tests. Six different sequences were chosen in the unit test, each with a different set of parameters. The symbol "n/a" means that the parameter is not applicable for that test. The notation rand(a,b) means the parameter was chosen randomly between a and b.

Echo Type	α (deg)	Φ (deg)	T_1 (ms)	T_2 (ms)	T_E (ms)	T_R (ms)
SE	$\frac{\pi}{2} \to \pi$	$\frac{\pi}{2} \to 0$	600	100	25	1000
SR	$\frac{\pi}{3} \to \cdots$	$\frac{\pi}{2} \to \cdots$	600	100	1	500
FSE	$\frac{\pi}{2} \to \frac{2\pi}{3} \to \cdots$	$\frac{\pi}{2} \to 0 \to \cdots$	n/a	n/a	0	0
FSE + relax	$\frac{\pi}{2} \to \pi \to \cdots$	$\frac{\pi}{2} \to 0 \to \cdots$	600	100	25	50
FISP	$\frac{\pi}{6} \to \cdots$	$0 \to \cdots$	1000	100	5	10
SSFP	rand(0,π/3)	$0 \to \cdots$	100, 200	20, 40	rand(3.5, 7.5)	16

Unit test 1: Spin echo

In this subsection, we show in detail how to calculate the magnetization response using EPG. The calculation for other echoes is similar to the spin echo except that the corresponding parameters and operators used are different.

First, we tested a simple spin echo sequence, which consists of a 90-deg pulse ($\alpha = \pi/2$) about the y-axis ($\Phi = \pi/2$) followed by a 180-deg refocusing pulse about the x-axis ($\Phi = 0$) at T_E. Here, $T_1 = 600$ ms, $T_2 = 100$ ms, $T_E = 25$ ms and $T_R = 1000$ ms. The corresponding host function in snapMRF is `epg_se()`. Note here that the meaning of T_E is somewhat different than in traditional spin echo terminology. In traditional spin echo, the echo time is the time between the initial 90-deg pulse and the readout. snapMRF, however, is optimized for simple representation of steady-state sequences, so the echo time must be defined as the time between the 180-deg inversion and the readout. To convert to standard spin-echo T_E you can simply double the T_E here. The same situation also applies for unit test 4.

First, we have the transition matrices:

$$\mathrm{T}_y(90^\circ) = \begin{bmatrix} 0.5 & -0.5 & 1 \\ -0.5 & 0.5 & 1 \\ -0.5 & -0.5 & 0 \end{bmatrix} \tag{B.1}$$

and

$$\mathrm{T}_x(180^\circ) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}. \tag{B.2}$$

Starting with the equilibrium magnetization, the initial state matrix (`init_rf_pulse()`) is

$$\Omega(t < 0) = [0, 0, 1]^T. \tag{B.3}$$

At $t = 0$, the 90-degree pulse (`apply_rf_pulse()`) changes the state matrix to be

$$\Omega(t = 0) = T_y(90^\circ)\Omega(t < 0) = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}. \tag{B.4}$$

Next, all the states are shifted up by 1 (`dephase_gradients()`) followed by relaxation for T_E (`decay_signal()`):

$$\begin{aligned}\Omega(t=T_E-) &= ES\Omega(t=0)\\ &= \mathrm{E}\begin{bmatrix}0&1\\0&0\\0&0\end{bmatrix}\\ &= \begin{bmatrix}0 & e^{-T_E/T_2}\\ 0 & 0\\ 1-e^{-T_E/T_1} & 0\end{bmatrix}\end{aligned} \quad \text{(B.5)}$$

Then the 180-degree refocusing pulse is applied:

$$\begin{aligned}\Omega(t=T_E+) &= T_x(180°)\Omega(t=T_E-)\\ &= \begin{bmatrix}0 & 0\\ 0 & e^{-T_E/T_2}\\ e^{-T_E/T_1}-1 & 0\end{bmatrix}\end{aligned} \quad \text{(B.6)}$$

Again all states are shifted up by 1, followed by relaxation for T_E to read the spin echo:

$$\begin{aligned}\Omega(t=T_E) &= ES\Omega(t=T_E+)\\ &= \mathrm{E}\begin{bmatrix}e^{-T_E/T_2} & 0 & 0\\ e^{-T_E/T_2} & 0 & 0\\ e^{-T_E/T_1}-1 & 0 & 0\end{bmatrix}\\ &= \begin{bmatrix}\left(e^{-T_E/T_2}\right)^2 & 0 & 0\\ \left(e^{-T_E/T_2}\right)^2 & 0 & 0\\ \left(e^{-T_E/T_1}-1\right)^2 & 0 & 0\end{bmatrix}\end{aligned} \quad \text{(B.7)}$$

Thus the echo intensity is $(e^{-T_E/T_2})^2 = [\exp(25/100)]^2 = 0.6065$.

Unit test 2: Saturation recovery

This sequence consists of 60-deg excitation pulses about the y-axis ($\Phi = \pi/2$) equally spaced with interval T_R. Here $T_1 = 600$ ms, $T_2 = 100$ ms, $T_E = 1$ ms and $T_R = 500$ ms. The corresponding host function in snapMRF is `epg_sr()`.

Similar to the spin echo, we also start with equilibrium magnetization of $[0,0,1]^T$, apply RF pulse, decay for time interval T_E, read the echo, and then decay for time interval $T_R - T_E$. This is repeated ten times, which corresponds to ten successive pulses, and the following signal values result: [0.857, 0.674, 0.631, 0.622, 0.620, 0.620, 0.620, 0.620, 0.620, 0.620]. We can see that after several pulses the echo intensity reaches a steady state of 0.620.

Unit test 3: Fast spin echo without relaxation

The third sequence we tested was a turbo spin echo sequence with constant 120-deg refocusing flip angles, neglecting relaxation effects. To obey the CPMG condition, the 90-deg RF pulse has a phase of $\pi/2$, whereas all 120-deg refocusing pulses have a phase of 0. Since relaxation effects were omitted, T_1 and T_2 were infinity in theory. In the code, however, that was not possible, so T_E and T_R were set to 0, and T_1 and T_2 were set randomly, and were not used. The corresponding host function in snapMRF is `epg_tse()`.

This sequence corresponds to repeating a spin echo several times, but neglecting relaxation effects. As a result, only the transition operator T and dephasing operator S were needed in this case. Here we only calculated the first three echoes, but the calculation can continue as the echo intensity gradually reaches the pseudo steady state. The final echo intensities are [0.750, 0.938, 0.844].

Unit test 4: Fast spin echo with relaxation

The fourth sequence was also a fast spin echo sequence except that relaxation effects were considered. It consists one 90-degree excitation pulse followed by eight 180-degree refocusing pulses (ETL = 8). Here $T_1 = 600$ ms, $T_2 = 100$ ms, $T_E = 25$ ms and $T_R = 50$ ms. The corresponding host function in snapMRF is `epg_tse()`.

The calculation is similar to the last case except that the relaxation operator E has to be used. The final echo intensities were [0.607, 0.368, 0.223, 0.235, 0.082, 0.0500, 0.0302, 0.0183, 0.0111, 0.0068].

Unit test 5: Rapid gradient echo

The fifth sequence we tested was an F_0-type SSFP sequence (commonly known as FISP) which has gradient spoiling but no RF spoiling. It has a constant flip angle of 30 deg around the x-axis ($\Phi = 0$) with equally spaced T_R. Here $T_1 = 1000$ ms, $T_2 = 100$ ms, $T_E = 5$ ms and $T_R = 10$ ms. The corresponding host function in snapMRF is `epg_fisp()`.

Here, the calculations for the transition matrix T and the relaxation operator E were like the former examples, but the shift operator S was tricky. If we followed the EPG method by rote to simulate this sequence, all states would get shifted by -1 and 1 before the readout. But since the net effect of the gradient before readout is nil on the configuration states, applying the shift operator S twice is pointless [4]. Thus here the shift operator S was used only once per T_R, and the echo intensity was computed (assuming $T_2 \gg T_E$ in this case) as

$$\Omega(t=T_E) = e^{-T_E/T_2}\Omega(t=0). \quad \text{(B.8)}$$

The final results were $[-0.4756i, -0.4125i, -0.3358i]$, where $i = \sqrt{-1}$.

Unit tests 6 and 7: Inversion-recovery balanced steady-state free precession

The last sequence-based unit tests we performed were an inversion-recovery balanced steady-state free procession (IR-bSSFP) sequence using both Bloch simulation and EPG. The difference between this sequence and the one used in Unit Test 5 is the addition of an inversion recovery period at the beginning. For both models, we tested a dictionary containing four atoms with five time points each. The corresponding host functions are `epg_ssfp()` and `roa_ssfp()` respectively.

Unit test 8: Dictionary self-matching

An additional unit test for matching was performed using the dictionary generated by EPG. Ten different voxels were selected randomly from the dictionary and matched to the dictionary. The corresponding host function is `MRF_match()`. The matched atoms were validated to ensure that the matched parameters were correct.

Reference

[1] Ma D, Gulani V, Seiberlich N, Liu K, Sunshine J, Duerk J L. Magnetic resonance fingerprinting. Nature 2013;495(7440):187–92. https://doi.org/10.1038/nature11971.

[2] European Society of Radiology (ESR). Magnetic resonance fingerprinting — a promising new approach to obtain standardized imaging biomarkers from MRI. Insights into Imaging 2015;6(2):163–5. https://doi.org/10.1007/s13244-015-0403-3.

[3] Bipin Mehta B, Coppo S, McGivney D, Hamilton J, Chen Y, Jiang Y. Magnetic resonance fingerprinting: a technical review. Magn Reson Med 2019;81(1):25–46. https://doi.org/10.1002/mrm.27403.

[4] Weigel M. Extended phase graphs: dephasing, RF pulses, and echoes–pure and simple. J Magn Reson Imaging 2015;41(2):266–95. https://doi.org/10.1002/jmri.24619.

[5] Jiang Y, Ma D, Seiberlich N, Gulani V, Griswold M A. MR fingerprinting using fast imaging with steady state precession (FISP) with spiral readout. Magn Reson Med 2015;74(6):1621–31. https://doi.org/10.1002/mrm.25559.

[6] Malik S J, Teixeira R P A, Hajnal J V. Extended phase graph formalism for systems with magnetization transfer and exchange: EPG-X: extended phase graphs with exchange. Magn Reson Med 2018;80(2):767–79. https://doi.org/10.1002/mrm.27040.

[7] Cloos M, Knoll F, Zhao T, Block K, Bruno M, Wiggins G C. Multiparametric imaging with heterogeneous radiofrequency fields. Nat Commun 2016;7:12445. https://doi.org/10.1038/ncomms12445.

[8] McGivney D F, Pierre E, Ma D, Jiang Y, Saybasili H, Gulani V. SVD compression for magnetic resonance fingerprinting in the time domain. IEEE Trans Med Imaging 2014;33(12):2311–22. https://doi.org/10.1109/TMI.2014.2337321.

[9] Cauley StephenF, Kawin Setsompop, Ma Dan, Yun Jiang, Ye Huihui, Elfar Adalsteinsson, Wald L L. Fast group matching for MR fingerprinting reconstruction. Magn Reson Med 2015;74(2):523–8. https://doi.org/10.1002/mrm.25439.

[10] Cohen O, Zhu B, Rosen M S. MR fingerprinting deep reconstruction network (DRONE). Magn Reson Med 2018;80(3):885–94. https://doi.org/10.1002/mrm.27198.

[11] Sørensen T S, Schaeffter T, Noe K Ø, Hansen M S. Accelerating the nonequispaced fast Fourier transform on commodity graphics hardware. IEEE Trans Med Imaging 2008;27(4):538–47. https://doi.org/10.1109/TMI.2007.909834.

[12] Knoll F, Unger M, Diwoky C, Clason C, Pock T, Stollberger R. Fast reduction of undersampling artifacts in radial MR angiography with 3D total variation on graphics hardware. Magn Reson Mater Phys, Biol Med 2010;23(2):103–14. https://doi.org/10.1007/s10334-010-0207-x.

[13] Smith D S, Sengupta S, Smith S A, Welch E B. Trajectory optimized NUFFT: faster non-Cartesian MRI reconstruction through prior knowledge and parallel architectures. Magn Reson Med 2019;81(3):2064–71. https://doi.org/10.1002/mrm.27497.

[14] Vasilevsky N A, Brush M H, Paddock H, Ponting L, Tripathy S J, LaRocca G M. On the reproducibility of science: unique identification of research resources in the biomedical literature. PeerJ 2013;1:e148. https://doi.org/10.7717/peerj.148.

[15] Collins F S, Tabak L A. Policy: NIH plans to enhance reproducibility. Nature 2014;505(7485):612–3. https://doi.org/10.1038/505612a.

[16] Open Science Collaboration. Estimating the reproducibility of psychological science. Science 2015;349(6251). https://doi.org/10.1126/science.aac4716.

[17] Begley C G, Ioannidis J P. Reproducibility in science. Circ Res 2015;116(1):116–26. https://doi.org/10.1161/CIRCRESAHA.114.303819.

[18] Smith D S, RawArray file format reference implementation : http://github.com/davidssmith/ra 2018. [Online; accessed 12-May-2018].

[19] Keenan K E, Stupic K F, Boss M A, Russek S E, Chenevert T L, Prasad P V. Comparison of T1 measurement using ISMRM/NIST system phantom In: ISMRM 24th Annual Meeting. 2016. p. p.3290.

[20] Pipe J G, Zwart N R. Spiral trajectory design: a flexible numerical algorithm and base analytical equations. Magn Reson Med 2014;71(1):278–85. https://doi.org/10.1002/mrm.24675.

[21] Uecker M, Tamir J I. Berkeley advanced reconstruction toolbox: version 0.4.01; 2017. 10.5281/zenodo.592960.

[22] Zwart N R, Johnson K O, Pipe J G. Efficient sample density estimation by combining gridding and an optimized kernel. Magn Reson Med 2012;67(3):701–10. https://doi.org/10.1002/mrm.23041.

[23] Ma D, Coppo S, Yong Chen, McGivney D F, Jiang Y, Pahwa S. Slice profile and B1 corrections in 2D magnetic resonance fingerprinting. Magn Reson Med 2017;78(5):1781–9. https://doi.org/10.1002/mrm.26580.

[24] Ostenson J, Damon B M, Welch E B. Mr fingerprinting with simultaneous T1, T2, and fat signal fraction estimation with integrated b0 correction reduces bias in water T1 and T2 estimates. Magn Reson Imaging 2019;60:7–19. https://doi.org/10.1016/j.mri.2019.03.017.

[25] Hamilton J I, Griswold M A, Seiberlich N. Mr fingerprinting with chemical exchange (MRF-X) to quantify subvoxel T1 and extracellular volume fraction. J Cardiovasc Magn Reson 2015;17(1):W35. https://doi.org/10.1186/1532-429X-17-S1-W35.

[26] Wang D, Ostenson J, Smith D S. snapMRF repository. : http://github.com/chixindebaoyu/snapMRF; 2019 [Online; accessed 03-Mar-2019].

[27] Hargreves B. Bloch equation simulation. : http://mrsrl.stanford.edu/brian/bloch; 2019 [Online; accessed 25-January-2019].