NLSAM Documentation

Release 0.7.1

Samuel St-Jean

Jul 03, 2023

CONTENTS

This is the documentation detailing the internal of the non local spatial and angular matching (NLSAM) denoising algorithm for diffusion MRI, which is available at https://github.com/samuelstjean/nlsam.

You can find the original paper and full details of the algorithm as presented in

```
St-Jean, S., Coupé, P., & Descoteaux, M. (2016)
"Non Local Spatial and Angular Matching :
Enabling higher spatial resolution diffusion MRI datasets through adaptive denoising."
Medical Image Analysis, 32(2016), 115-130.
```

Which you can grab a copy from the publisher website or from arxiv.

You can find below the documentation for each modules and a few instructions on topics such as noise estimation and detailed installation instructions.

CHAPTER

ONE

NLSAM

1.1 Submodules

1.1.1 nlsam.angular_tools

Module Contents

Functions

angular_neighbors(vec, n)	Returns the indices of the n closest neighbors (excluding the vector itself)
_angle(vec)	Inner function that finds the angle between all vectors of the input.
greedy_set_finder(sets)	Returns a list of subsets that spans the input sets with a greedy algorithm
split_per_shell(bvals, bvecs, angular_size, dwis[, ...])	Process each shell separately for finding the valid angular neighbors.

nlsam.angular_tools.**angular_neighbors**(*vec*, *n*)

Returns the indices of the n closest neighbors (excluding the vector itself) given an array of m points with x, y and z coordinates.

Input : A m x 3 array, with m being the number of points, one per line. Each column has x, y and z coordinates for each vector.

Output : A m x n array. Each line has the n indices of the closest n neighbors amongst the m input vectors.

Note : Symmetries are not considered here so a vector and its opposite sign counterpart will be considered far apart, even though in dMRI we consider (x, y, z) and -(x, y, z) to be practically identical.

nlsam.angular_tools.**_angle**(*vec*)

Inner function that finds the angle between all vectors of the input. The diagonal is the angle between each vector and itself, thus 0 everytime. It should not be called as is, since it serves mainly as a shortcut for other functions.

arccos(0) = pi/2, so b0s are always far from everyone in this formulation.

nlsam.angular_tools.**greedy_set_finder**(*sets*)

Returns a list of subsets that spans the input sets with a greedy algorithm http://en.wikipedia.org/wiki/Set_cover_problem#Greedy_algorithm

nlsam.angular_tools.**split_per_shell**(*bvals*, *bvecs*, *angular_size*, *dwis*, *is_symmetric=False*, *bval_threshold=25*)

Process each shell separately for finding the valid angular neighbors. Returns a list of indexes for each shell separately

1.1.2 nlsam.bias_correction

Module Contents

Functions

stabilization(data, m_hat, sigma, N[, mask, clip_eta, ...])	
root_finder_sigma(data, sigma, N[, mask, verbose, n_cores])	Compute the local corrected standard deviation for the adaptive nonlocal

Attributes

logger

nlsam.bias_correction.**logger**

nlsam.bias_correction.**stabilization**(*data*, *m_hat*, *sigma*, *N*, *mask=None*, *clip_eta=True*, *return_eta=False*, *n_cores=-1*, *verbose=False*)

nlsam.bias_correction.**root_finder_sigma**(*data*, *sigma*, *N*, *mask=None*, *verbose=False*, *n_cores=-1*)

Compute the local corrected standard deviation for the adaptive nonlocal means according to the correction factor xi.

Input

data
[ndarray] Signal intensity

sigma
[ndarray] Noise magnitude standard deviation

N
[ndarray or double] Number of coils of the acquisition (N=1 for Rician noise)

mask
[ndarray, optional] Compute only the corrected sigma value inside the mask.

verbose
[bool, optional] displays a progress bar if True

n_cores
[int, optional] number of cores to use for parallel processing

returns
Corrected sigma value, where sigma_gaussian = sigma / sqrt(xi)

rtype
output, ndarray

1.1.3 nlsam.denoiser

Module Contents

Functions

nlsam_denoise(data, sigma, bvals, bvecs, block_size[, ...])	Main nlsam denoising function which sets up everything nicely for the local
local_denoise(data, block_size, overlap, variance[, ...])	
processer(data, mask, variance, block_size, overlap, ...)	

Attributes

logger

nlsam.denoiser.**logger**

nlsam.denoiser.**nlsam_denoise**(*data*, *sigma*, *bvals*, *bvecs*, *block_size*, *mask=None*, *is_symmetric=False*, *n_cores=-1*, *split_b0s=False*, *split_shell=False*, *subsample=True*, *n_iter=10*, *b0_threshold=10*, *bval_threshold=25*, *dtype=np.float64*, *verbose=False*)

Main nlsam denoising function which sets up everything nicely for the local block denoising.

Input

data
[ndarray] Input volume to denoise.

sigma
[ndarray] Noise standard deviation estimation at each voxel. Converted to variance internally.

bvals
[1D array] the N bvalues associated to each of the N diffusion volume.

bvecs
[N x 3 2D array] the N 3D vectors for each acquired diffusion gradients.

block_size
[tuple, length = data.ndim] Patch size + number of angular neighbors to process at once as similar data.

Optional parameters

mask
[ndarray, default None] Restrict computations to voxels inside the mask to reduce runtime.

is_symmetric
[bool, default False] If True, assumes that for each coordinate (x, y, z) in bvecs, (-x, -y, -z) was also acquired.

n_cores
[int, default -1] Number of processes to use for the denoising. Default is to use all available cores.

split_b0s
[bool, default False] If True and the dataset contains multiple b0s, a different b0 will be used for each run of the denoising. If False, the b0s are averaged and the average b0 is used instead.

split_shell
[bool, default False] If True and the dataset contains multiple bvalues, each shell is processed independently. If False, all the data is used at the same time for computing angular neighbors.

subsample
[bool, default True] If True, find the smallest subset of indices required to process each dwi at least once.

n_iter
[int, default 10] Maximum number of iterations for the reweighted l1 solver.

b0_threshold
[int, default 10] A bvalue below b0_threshold will be considered as a b0 image.

bval_threshold
[int, default 25] Any bvalue within += bval_threshold of each others will be considered on the same shell (e.g. b=990 and b=1000 are on the same shell).

dtype
[np.float32 or np.float64, default np.float64] Precision to use for inner computations. Note that np.float32 should only be used for very, very large datasets (that is, your ram starts swapping) as it can lead to numerical precision errors.

verbose
[bool, default False] print useful messages.

Output

data_denoised
[ndarray] The denoised dataset

nlsam.denoiser.**local_denoise**(*data*, *block_size*, *overlap*, *variance*, *n_iter=10*, *mask=None*, *dtype=np.float64*, *n_cores=-1*, *verbose=False*)

nlsam.denoiser.**processer**(*data*, *mask*, *variance*, *block_size*, *overlap*, *param_alpha*, *param_D*, *current_slice*, *dtype=np.float64*, *n_iter=10*, *gamma=3*, *tau=1*, *tolerance=1e-05*)

1.1.4 nlsam.smoothing

Module Contents

Functions

sh_smooth(data, bvals, bvecs[, sh_order, ...])	Smooth the raw diffusion signal with spherical harmonics.
_local_standard_deviation(arr[, current_slice])	Standard deviation estimation from local patches.
local_standard_deviation(arr[, n_cores, verbose])	Standard deviation estimation from local patches.

Attributes

logger

nlsam.smoothing.**logger**

nlsam.smoothing.**sh_smooth**(*data*, *bvals*, *bvecs*, *sh_order=4*, *b0_threshold=1.0*, *similarity_threshold=50*, *regul=0.006*)

Smooth the raw diffusion signal with spherical harmonics.

data
[ndarray] The diffusion data to smooth.

gtab
[gradient table object] Corresponding gradients table object to data.

b0_threshold
[float, default 1.0] Threshold to consider this bval as a b=0 image.

sh_order
[int, default 8] Order of the spherical harmonics to fit.

similarity_threshold
[int, default 50] All bvalues such that **|b_1 - b_2|** < similarity_threshold will be considered as identical for smoothing purpose. Must be lower than 200.

regul
[float, default 0.006] Amount of regularization to apply to sh coefficients computation.

Returns
pred_sig – The smoothed diffusion data, fitted through spherical harmonics.

Return type
ndarray

nlsam.smoothing.**_local_standard_deviation**(*arr*, *current_slice=None*)

Standard deviation estimation from local patches.

Estimates the local variance on patches by using convolutions to estimate the mean. This is the multiprocessed function.

Parameters

- **arr** (*3D or 4D ndarray*) – The array to be estimated
- **current_slice** (*numpy slice object*) – current slice to evaluate if we are running in parallel

Returns
sigma – Map of standard deviation of the noise.

Return type
ndarray

nlsam.smoothing.**local_standard_deviation**(*arr*, *n_cores=-1*, *verbose=False*)

Standard deviation estimation from local patches.

The noise field is estimated by subtracting the data from it's low pass filtered version, from which we then compute the variance on a local neighborhood basis.

Parameters

- **arr** (*3D or 4D ndarray*) – The array to be estimated
- **n_cores** (*int*) – Number of cores to use for multiprocessing, default : all of them
- **verbose** (*int*) – If True, prints progress information. A higher number prints more often

Returns
sigma – Map of standard deviation of the noise.

Return type
ndarray

CHAPTER

TWO

ADVANCED TECHNIQUES FOR ESTIMATING DEGREES OF FREEDOM IN A NON CENTRAL CHI DISTRIBUTION

This section is mostly personal recommendations based on some literature, stuff I have played with and stuff I have seen in MR physics classes. It is probably not exhaustive nor perfectly accurate, but should give the interested reader a feeling of what is happening and why.

2.1 The problem

Noise estimation in MR highly depends on the reconstruction algorithm implemented by your vendor (*Dietrich et al.*). Unfortunately, due to interference between adjacent receiver coils as used in modern parallel imaging (i.e. pretty much always unless you are doing fancy specialized acquisitions), the real noise distribution is slightly different than a pure Rician or Noncentral chi distribution (*Aja-Fernandez et al., Constantinides et al.*).

It is still possible to estimate the distribution, but the values of the 'standard deviation' and degrees of freedom of that distribution depends on the parameters of the acquisition (i.e. SENSE maps, GRAPPA weights), which are probably hard to acquire if you do not have a friendly MR physicist at hand (they are always nice guys anyway, so don't be afraid to ask for help). The authors of (*Aja-Fernandez et al.*) still offer a way to do a blind estimation of these values for those interested to dig a bit more.

2.2 Why it is hard to find a surefire correction method

Also based on my MR physics class understanding, due to the way the (closed source) algorithm in each vendor's scanner software work, they are likely to discard the signal coming from far away receiver elements from the imaged body region. As an example, near the top of the head, the coil elements placed near the neck are very likely to measure little relevant signal and mostly contribute noise.

This means that during the k-space reconstruction, the signal contribution from these coils will get thrown away, and thus the number of effectively used coils will vary per region and will be lower than the number of coils on you receiver array. Add noise correlation into that due to electrical interference and proximity of your receiver elements, and you are looking at some (hard to figure out) distribution which is different from what you expect according to the theory.

2.3 What other people suggest

The authors of (*Veraart et al.*) also provided another way to estimate those relevant parameters based on constructing synthetic noise maps for those interested in trying another method.

As a final tl;dr advice, some other studies have found that for GRAPPA reconstruction, with a 12 channels head coils N=4 (*Brion et al.*) (that's what we use in Sherbrooke also for the 1.5T Siemens scanner with a 12 channels head coil) works well and for a 32 channels head coils (*Varadarajan et al.*), a value around N=9 seems to work (remember that N varies spatially, but it seems to be fairly homogeneous/vary slowly).

The authors of (*Becker et al.*) also indicates that in the worst case, using N=1 for Rician noise is better than doing nothing.

Same observation from (*Sakaie et al.*) in real data; they fitted the background distribution and found out that for a sum of square (SoS) reconstruction with 12 coils, N = 3.76 $\pm$ 0.07 in 5 subjects (well, it should be an integer since it represents the number of degrees of freedom, so let's say N=4). As expected, it is much lower than 12 because of the correlation in each adjacent coils and produces DTI metrics (FA, MD, RD, AD) with a stronger bias than an adaptive combine (N = 1.03 $\pm$ 0.01) reconstruction.

2.4 Take home message

In all cases, the take home message would be that estimating the real value for N is still challenging, but it will most likely be lower than the number of coils present on your receiver coil due to the way MRI scanners reconstruct and combine images.

My new personal recommendation would now be to use the default option, that is to estimate both parameters of the distribution at once from the background noise. While this will underestimate most of the time the correct noise standard deviation, it seems more stable than guessworking or assuming N=1 in general, all the while providing a good ballpark estimate for the denoising process, see *St-Jean et al.* for more details.

2.5 References

Aja-Fernandez, S., Vegas-Sanchez-Ferrero, G., Tristan-Vega, A., 2014. Noise estimation in parallel MRI: GRAPPA and SENSE. Magnetic resonance imaging

Becker, S. M. A., Tabelow, K., Mohammadi, S., Weiskopf, N., & Polzehl, J. (2014). Adaptive smoothing of multi-shell diffusion weighted magnetic resonance data by msPOAS. NeuroImage

Brion, V., Poupon, C., Riff, O., Aja-Fernández, S., Tristán-Vega, A., Mangin, J.-F., Le Bihan D, Poupon, F. (2013). Noise correction for HARDI and HYDI data obtained with multi-channel coils and sum of squares reconstruction: an anisotropic extension of the LMMSE. Magnetic Resonance Imaging

Constantinides, C. D., Atalar, E., & McVeigh, E. R. (1997). Signal-to-Noise Measurements in Magnitude Images from NMR Phased Arrays. Magnetic Resonance in Medicine, 38(5), 852–857.

Dietrich, O., Raya, J. G., Reeder, S. B., Ingrisch, M., Reiser, M. F., & Schoenberg, S. O. (2008). Influence of multichannel combination, parallel imaging and other reconstruction techniques on MRI noise characteristics. Magnetic Resonance Imaging

Sakaie, M. & Lowe, M., Retrospective correction of bias in diffusion tensor imaging arising from coil combination mode, Magnetic Resonance Imaging, Volume 37, April 2017

St-Jean S, De Luca A, Tax C.M.W., Viergever M.A, Leemans A. (2020) Automated characterization of noise distributions in diffusion MRI data Medical Image Analysis, October 2020:101758. doi:10.1016/j.media.2020.101758

Varadarajan, D., & Haldar, J. (2015). A Majorize-Minimize Framework for Rician and Non-Central Chi MR Images. IEEE Transactions on Medical Imaging

Veraart, J., Rajan, J., Peeters, R. R., Leemans, A., Sunaert, S., & Sijbers, J. (2013). Comprehensive framework for accurate diffusion MRI parameter estimation. Magnetic Resonance in Medicine

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

INDEX

Symbols

A

G

L

M

N

P

R

S